INSIDE MACINTOSH

Mac OS 8
Control Manager Reference

Updated for Appearance 1.0.2

November 18, 1998
Technical Publications
© 1998 Apple Computer, Inc.

Apple Computer, Inc.
© 1997, 1998 Apple Computer, Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Computer, Inc., except to make a backup copy of any documentation provided on CD-ROM.

The Apple logo is a trademark of Apple Computer, Inc.
Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this book. Apple retains all intellectual property rights associated with the technology described in this book. This book is intended to assist application developers to develop applications only for Apple-labeled or Apple-licensed computers.

Every effort has been made to ensure that the information in this manual is accurate. Apple is not responsible for typographical errors.

Apple Computer, Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Mac, and Macintosh are trademarks of Apple Computer, Inc., registered in the United States and other countries.

Adobe, Acrobat, and PostScript are trademarks of Adobe Systems Incorporated or its subsidiaries and may be registered in certain jurisdictions.

Helvetica and Palatino are registered trademarks of Linotype-Hell AG and / or its subsidiaries.

ITC Zapf Dingbats is a registered trademark of International Typeface Corporation.

Simultaneously published in the United States and Canada.

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

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

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

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

Contents

11/18/98 © Apple Computer, Inc.

11/18/98 © Apple Computer, Inc.

Figures, Tables, and Listings

11/18/98 © Apple Computer, Inc.

11/18/98 © Apple Computer, Inc.

CHAPTER 1

Control Manager Reference

Contents

11/18/98 © Apple Computer, Inc.

11/18/98 © Apple Computer, Inc.

11/18/98 © Apple Computer, Inc.

11/18/98 © Apple Computer, Inc.

Your program can use the Control Manager to create and manage controls. Controls are onscreen objects that the user can manipulate with the mouse. By manipulating controls, the user can take an immediate action or change settings to modify a future action.

Portions of the Control Manager application programming interface (API) are new, changed, or not recommended with Mac OS 8 or Appearance Manager 1.0. See the following sections for descriptions of the changes to the Control Manager:

- "Control Manager Functions" (page 11)
- "Control Manager Data Types" (page 94)
- "Control Manager Constants" (page 106)
- "Result Codes" (page 140)

For descriptions of the parts of the Control Manager API that are unaffected by Appearance Manager 1.0, see *Inside Macintosh: Macintosh Toolbox Essentials*. For a description of the Mac OS 8.5 Control Manager API, see *Mac OS 8.5 Control Manager Reference*.

Control Manager Functions

Control Manager functions in the following areas have been affected by Appearance Manager 1.0:

- "Creating and Removing Controls" (page 12)
- "Embedding Controls" (page 17)
- "Manipulating Controls" (page 27)
- "Displaying Controls" (page 32)
- "Handling Events in Controls" (page 34)
- "Handling Keyboard Focus" (page 41)
- "Accessing and Changing Control Settings and Data" (page 46)
- "Defining Your Own Control Definition Function" (page 56)
- "Defining Your Own Action Functions" (page 78)

11/18/98 © Apple Computer, Inc.

- "Defining Your Own Key Filter Function" (page 80)
- "Defining Your Own User Pane Functions" (page 83)

Creating and Removing Controls

The following Control Manager functions for creating and removing controls are new, changed, or not recommended with Appearance Manager 1.0:

- `GetNewControl` (page 12) creates a control from a control resource. Changed with Appearance Manager 1.0.
- `NewControl` (page 13) creates a control based on parameter data. Changed with Appearance Manager 1.0.
- `DisposeControl` (page 15) removes a control and any of its embedded controls from a window. Changed with Appearance Manager 1.0.
- `KillControls` (page 16) removes all controls in a specified window. Changed with Appearance Manager 1.0.

GetNewControl

Creates a control from a control resource.

```
pascal ControlHandle GetNewControl (
                     SInt16 resourceID,
                     WindowPtr owningWindow);
```

`resourceID` The resource ID of the control you wish to create; see Table 1-1 (page 108).

`owningWindow` A pointer to the window in which to place the control.

function result Returns a handle to the control created from the specified control resource. If `GetNewControl` can't read the control resource from the resource file, it returns `nil`.

DISCUSSION

The `GetNewControl` function creates a control structure from the information in the specified control resource, adds the control structure to the control list for

11/18/98 © Apple Computer, Inc.

the specified window, and returns as its function result a handle to the control. You use this handle when referring to the control in most other Control Manager functions. After making a copy of the control resource, `GetNewControl` releases the memory occupied by the original control resource before returning.

The control resource specifies the rectangle for the control, its initial setting, its visibility state, its maximum and minimum settings, its control definition ID, a reference value, and its title (if any). After you use `GetNewControl` to create the control, you can change the control characteristics with other Control Manager functions.

If the control resource specifies that the control should be visible, the Control Manager draws the control. If the control resource specifies that the control should initially be invisible, you can use the function `ShowControl` (page 27) to make the control visible.

When an embedding hierarchy is established within a window, `GetNewControl` automatically embeds the newly created control in the root control of the owning window. See "Embedding Controls" (page 17).

If you are using standard system controls, default colors are used and the control color table resource is ignored. To use colors other than the default colors, you must write your own custom control definition function.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

SEE ALSO

`NewControl` (page 13).

NewControl

Creates a control based on parameter data.

```
pascal ControlHandle NewControl (
                     WindowPtr owningWindow,
                     const Rect *boundsRect,
                     ConstStr255Param controlTitle,
                     Boolean initiallyVisible,
                     SInt16 initialValue,
```


11/18/98 © Apple Computer, Inc.

```
                    SInt16 minimumValue,
                    SInt16 maximumValue,
                    SInt16 procID,
                    SInt32 controlReference);
```

`owningWindow` A pointer to the window in which you want to place the control. All coordinates pertaining to the control are interpreted in this window's local coordinate system.

`boundsRect` A pointer to a rectangle, specified in the given window's local coordinates, that encloses the control and thus determines its size and location. When specifying this rectangle, you should follow the guidelines presented in "Dialog Box Layout", in *Mac OS 8 Human Interface Guidelines*, for control placement and alignment.

`controlTitle` The title string, used for push buttons, checkboxes, radio buttons, and pop-up menus. When specifying a multiple-line title, separate the lines with the ASCII character code 0x0D (carriage return). For controls that don't use titles, pass an empty string.

`initiallyVisible`
A Boolean value specifying the visible/invisible state for the control. If you pass `true` in this parameter, `NewControl` draws the control immediately, without using your window's standard updating mechanism. If you pass `false`, you must later use `ShowControl` (page 27) to display the control.

`initialValue` The initial setting for the control; see "Settings Values for Standard Controls" (page 113).

`minimumValue` The minimum setting for the control; see "Settings Values for Standard Controls" (page 113).

`maximumValue` The maximum setting for the control; see "Settings Values for Standard Controls" (page 113).

`procID` The control definition ID; see Table 1-1 (page 108). If the control definition function isn't in memory, it is read in.

`controlReference`
The control's reference value, which is set and used only by your application.

11/18/98 © Apple Computer, Inc.

function result Returns a handle to the control described in its parameters. If `NewControl` runs out of memory or fails, it returns `nil`.

DISCUSSION

The `NewControl` function creates a control structure from the information you specify in its parameters, adds the control structure to the control list for the specified window, and returns as its function result a handle to the control. You can use this handle when referring to the control in most other Control Manager functions. Generally, you should use the function `GetNewControl` (page 12) instead of `NewControl`, because `GetNewControl` is a resource-based control-creation function that allows you to localize your application without recompiling.

When an embedding hierarchy is established within a window, `NewControl` automatically embeds the newly created control in the root control of the owning window. See "Embedding Controls" (page 17).

If you are using standard system controls, default colors are used and the control color table resource is ignored. To use colors other than the default colors, write your own custom control definition function.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

SEE ALSO

`GetNewControl` (page 12).

DisposeControl

Removes a control and any of its embedded controls from a window.

```
pascal void DisposeControl (ControlHandle theControl);
```

`theControl` A handle to the control you wish to remove.

11/18/98 © Apple Computer, Inc.

DISCUSSION

The `DisposeControl` function removes the specified control (and any embedded controls it may possess) from the screen, deletes it from the window's control list, and releases the memory occupied by the control structure and any data structures associated with the control. Passing the root control to this function is the effectively the same as calling `KillControls` (page 16). If an embedding hierarchy is present, `DisposeControl` disposes of the controls embedded within a control before disposing of the container control.

You should use `DisposeControl` when you wish to retain the window but dispose of one of its controls. The Window Manager functions `CloseWindow` and `DisposeWindow` automatically dispose of all controls associated with the given window.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

SEE ALSO

"Embedding Controls" (page 17).

KillControls

Removes all controls in a specified window.

```
pascal void KillControls (WindowPtr theWindow);
```

`theWindow` A pointer to the window whose controls you wish to remove.

DISCUSSION

The `KillControls` function disposes of all controls associated with the specified window. To remove just one control, use `DisposeControl` (page 15). If an embedding hierarchy is present, `KillControls` disposes of the controls embedded within a control before disposing of the container control.

You should use `KillControls` when you wish to retain the window but dispose of its controls. The Window Manager functions `CloseWindow` and `DisposeWindow` automatically dispose of all controls associated with the given window.

11/18/98 © Apple Computer, Inc.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

SEE ALSO

“Embedding Controls” (page 17).

Embedding Controls

This section provides functions that you can use to establish an embedding hierarchy. This can be accomplished in two steps: creating a root control and embedding controls within it.

To embed controls in a window, you must create a root control for that window. The **root control** is the container for all other window controls. You create the root control in one of two ways—by calling the `CreateRootControl` (page 19) function or by setting the appropriate dialog flag. The root control can be retrieved by calling `GetRootControl` (page 20).

The root control is implemented as a user pane control. You can attach any application-defined user pane functions to the root control to perform actions such as hit testing, drawing, handling keyboard focus, erasing to the correct background, and processing idle and keyboard events. For information on how to write these functions, see “Defining Your Own User Pane Functions” (page 83).

Once you have created a root control, newly created controls will automatically be embedded in the root control when you call `NewControl` (page 13) or `GetNewControl` (page 12). You can specify that a specific control be embedded into another by calling `EmbedControl` (page 21).

By acting on an embedder control, you can move, disable, or hide groups of items. For example, you can use a blank user pane control as the embedder control for all items in a particular “page” of a tab control. After creating as many user panes as you have tabs, you can hide one and show the next when a tab is clicked. All the controls embedded in the user pane will be hidden and shown automatically when the user pane is hidden and shown.

The Dialog Manager uses `AutoEmbedControl` (page 22) to position dialog items in an **embedding hierarchy** based on both visual containment and the item list resource order. As items are added to a dialog box during creation, controls that already exist in the window will be containers for new controls if they both

11/18/98 © Apple Computer, Inc.

visually contain the control and have set the `kControlSupportsEmbedding` feature bit. For this reason, you should place the largest embedder controls at the beginning of the item list resource. As an example, the Dialog Manager would embed radio buttons in a tab control if they visually "fit" inside the tab control, as long as the tab control was already created in a `'DITL'` resource and established as an embedder control.

In addition to calling `CreateRootControl`, you can establish an embedding hierarchy in a dialog box by either setting the feature bit `kDialogFlagsUseControlHierarchy` in the extended dialog resource or passing it in the `inFlags` parameter of the Dialog Manager function `NewFeaturesDialog`. An embedding hierarchy can be created in an alert box by setting the `kAlertFlagsUseControlHierarchy` bit in the extended alert resource. It is important to note that a preexisting alert or dialog item will become a control if it is in an alert or dialog box that now uses an embedding hierarchy.

The embedding hierarchy enforces drawing order by drawing the embedding control before its embedded controls. Using an embedding hierarchy also enforces orderly hit-testing, since it performs an "inside-out" hit test to determine the most deeply nested control that is hit by the mouse. An embedding hierarchy is also necessary for controls to make use of keyboard focus, the default focusing order for which is a linear progression that uses the order the controls were added to the window. For more details on keyboard focus, see "Handling Keyboard Focus" (page 41).

The following Control Manager functions for embedding controls are new with Appearance Manager 1.0:

- `CreateRootControl` (page 19) creates the root control for a specified window. New with Appearance Manager 1.0.
- `GetRootControl` (page 20) obtains a handle to a window's root control. New with Appearance Manager 1.0.
- `EmbedControl` (page 21) embeds one control inside another. New with Appearance Manager 1.0.
- `AutoEmbedControl` (page 22) automatically embeds a control in the smallest appropriate embedder control. New with Appearance Manager 1.0.
- `CountSubControls` (page 22) obtains the number of embedded controls within a control. New with Appearance Manager 1.0.
- `GetIndexedSubControl` (page 23) obtains a handle to a specified embedded control. New with Appearance Manager 1.0.

11/18/98 © Apple Computer, Inc.

- `GetSuperControl` (page 24) obtains a handle to the embedder control. New with Appearance Manager 1.0.
- `SetControlSupervisor` (page 25) routes mouse-down events to the embedder control. New with Appearance Manager 1.0.
- `DumpControlHierarchy` (page 26) writes a textual representation of the control hierarchy for a specified window into a file. New with Appearance Manager 1.0.

CreateRootControl

Creates the root control for a specified window.

```
pascal OSErr CreateRootControl (
                    WindowPtr inWindow,
                    ControlHandle* outControl);
```

`inWindow`	A pointer to the window in which you wish to create a root control.
`outControl`	Pass a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the root control.
function result	A result code; see "Result Codes" (page 140).

DISCUSSION

The `CreateRootControl` function creates the root control for a window if no other controls are present. If there are any controls in the window prior to calling `CreateRootControl`, an error is returned and the root control is not created.

The root control acts as the top-level container for a window and is required for embedding to occur. Once the root control is created, you can call `EmbedControl` (page 21) and `AutoEmbedControl` (page 22) to embed controls in the root control.

Note
The minimum, maximum, and initial settings for a root control are reserved and should not be changed.

11/18/98 © Apple Computer, Inc.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

"Embedding Controls" (page 17).

GetRootControl

Obtains a handle to a window's root control.

```
pascal OSErr GetRootControl (
                    WindowPtr inWindow,
                    ControlHandle* outControl);
```

`inWindow` A pointer to the window to be examined.

`outControl` Pass a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the root control.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

You can call `GetRootControl` to determine whether or not a root control (and therefore an embedding hierarchy) exists within a specified window. Once you have the root control's handle, you can pass it to functions such as `DisposeControl` (page 15), `ActivateControl` (page 29), and `DeactivateControl` (page 30) to apply specified actions to the entire embedding hierarchy.

Note
The minimum, maximum, and initial settings for a root control are reserved and should not be changed.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

SEE ALSO

"Embedding Controls" (page 17).

EmbedControl

Embeds one control inside another.

```
pascal OSErr EmbedControl (
                    ControlHandle inControl,
                    ControlHandle inContainer);
```

`inControl` A handle to the control to be embedded.

`inContainer` A handle to the embedder control.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

An embedding hierarchy must be established before your application calls the `EmbedControl` function. If the specified control does not support embedding or there is no root control in the owning window, an error is returned. If the control you wish to embed is in a different window from the embedder control, an error is returned. See "Embedding Controls" (page 17) for more details on embedding.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`AutoEmbedControl` (page 22).

11/18/98 © Apple Computer, Inc.

AutoEmbedControl

Automatically embeds a control in the smallest appropriate embedder control.

```
pascal OSErr AutoEmbedControl (
                    ControlHandle inControl,
                    WindowPtr inWindow);
```

`inControl` A handle to the control to be embedded.

`inWindow` A pointer to the window in which to embed the control.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

The Dialog Manager uses `AutoEmbedControl` (page 22) to position dialog items in an embedding hierarchy based on both visual containment and the item list resource order. For information on embedding hierarchies in dialog and alert boxes, see "Embedding Controls" (page 17).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`EmbedControl` (page 21).

CountSubControls

Obtains the number of embedded controls within a control.

```
pascal OSErr CountSubControls (
                    ControlHandle inControl,
                    SInt16* outNumChildren);
```

`inControl` A handle to a control whose embedded controls you wish to count.

11/18/98 © Apple Computer, Inc.

`outNumChildren`
Pass a pointer to a signed 16-bit integer value. On return, the value is set to the number of embedded subcontrols.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

The `CountSubControls` function is useful for iterating over the control hierarchy. You can use the count produced to determine how many subcontrols there are and then call `GetIndexedSubControl` (page 23) to get each.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

"Embedding Controls" (page 17).

GetIndexedSubControl

Obtains a handle to a specified embedded control.

```
pascal OSErr GetIndexedSubControl (
                    ControlHandle inControl,
                    SInt16 inIndex,
                    ControlHandle* outSubControl);
```

`inControl` A handle to an embedder control.

`inIndex` A 1-based index—an integer between 1 and the value returned in the `outNumChildren` parameter of `CountSubControls` (page 22)—specifying the control you wish to access.

`outSubControl` Pass a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the embedded control.

function result A result code; see "Result Codes" (page 140). If the index passed in is invalid, the `paramErr` result code is returned.

11/18/98 © Apple Computer, Inc.

DISCUSSION

The `GetIndexedSubControl` function is useful for iterating over the control hierarchy. Also, the value of a radio group control is the index of its currently selected embedded radio button control. So, passing the current value of a radio group control into `GetIndexedSubControl` will give you a handle to the currently selected radio button control.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

"Embedding Controls" (page 17).

GetSuperControl

Obtains a handle to an embedder control.

```
pascal OSErr GetSuperControl (
                    ControlHandle inControl,
                    ControlHandle* outParent);
```

`inControl` A handle to an embedded control.

`outParent` Pass a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the embedder control.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

The `GetSuperControl` function gets a handle to the parent control of the control passed in.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

SEE ALSO

"Embedding Controls" (page 17).

SetControlSupervisor

Routes mouse-down events to the embedder control.

```
pascal OSErr SetControlSupervisor (
                    ControlHandle inControl,
                    ControlHandle inBoss);
```

`inControl`	A handle to an embedded control.
`inBoss`	A handle to the embedder control to which mouse-down events are to be routed.
function result	A result code; see "Result Codes" (page 140).

DISCUSSION

The `SetControlSupervisor` function allows an embedder control to respond to mouse-down events occurring in its embedded controls.

An example of a standard control that uses this function is the radio group control. Mouse-down events in the embedded controls of a radio group are intercepted by the group control. (The embedded controls in this case must support radio behavior; if a mouse-down event occurs in an embedded control within a radio group control that does not support radio behavior, the control tracks normally and the group is not involved.) The group handles all interactions and switches the embedded control's value on and off. If the value of the radio group changes, `TrackControl` (page 41) or `HandleControlClick` (page 38) will return the `kControlRadioGroupPart` part code. If the user tracks off the radio button or clicks the current radio button, `kControlNoPart` is returned.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

SEE ALSO

"Embedding Controls" (page 17).

DumpControlHierarchy

Writes a textual representation of the control hierarchy for a specified window into a file.

```
pascal OSErr DumpControlHierarchy (
                    WindowPtr inWindow,
                    const FSSpec* inDumpFile);
```

`inWindow`	A pointer to the window whose control hierarchy you wish to examine.
`inDumpFile`	A pointer to a file specification in which to place a text description of the window's control hierarchy.
function result	A result code; see "Result Codes" (page 140).

DISCUSSION

The `DumpControlHierarchy` function places a text listing of the current control hierarchy for the window specified into the specified file, overwriting any existing file. If the specified window does not contain a control hierarchy, `DumpControlHierarchy` notes this in the text file. This function is useful for debugging embedding-related problems.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

"Embedding Controls" (page 17).

11/18/98 © Apple Computer, Inc.

Manipulating Controls

When showing, hiding, activating, or deactivating groups of controls, the state of an embedded control that is hidden or deactivated is preserved so that when the embedder control is shown or activated, the embedded control appears in the same state as the embedder. An embedded control is considered **latent** when it is deactivated or hidden due to its embedder control being deactivated or hidden. If you activate a latent embedded control whose embedder is deactivated, the embedded control becomes latent until the embedder is activated. However, if you deactivate a latent embedded control, it will not be activated when its embedder is activated.

When activating and deactivating controls in an embedding hierarchy, call `ActivateControl` (page 29) and `DeactivateControl` (page 30) instead of `HiliteControl` to ensure that latent embedded controls are displayed correctly.

The following Control Manager functions for manipulating controls are new, changed, or not recommended with Appearance Manager 1.0:

- `ShowControl` (page 27) makes an invisible control, and any latent embedded controls, visible. Changed with Appearance Manager 1.0.
- `HideControl` (page 28) makes a visible control, and any latent embedded controls, invisible. Changed with Appearance Manager 1.0.
- `ActivateControl` (page 29) activates a control and any latent embedded controls. New with Appearance Manager 1.0.
- `DeactivateControl` (page 30) deactivates a control and any latent embedded controls. New with Appearance Manager 1.0.
- `IsControlActive` (page 31) returns whether a control is active. New with Appearance Manager 1.0.
- `SendControlMessage` (page 31) sends a message to a control definition function. New with Appearance Manager 1.0.

ShowControl

Makes an invisible control, and any latent embedded controls, visible.

```
pascal void ShowControl (ControlHandle theControl);
```


11/18/98 © Apple Computer, Inc.

`theControl` A handle to the control to make visible.

DISCUSSION

If the specified control is invisible, the `ShowControl` function makes it visible and immediately draws the control within its window without using your window's standard updating mechanism. If the specified control has embedded controls, `ShowControl` makes the embedded controls visible as well. If the control is already visible, `ShowControl` has no effect.

If you call `ShowControl` on a latent embedded control whose embedder is disabled, the embedded control will be invisible until its embedder control is enabled. For a discussion of latency, see "Manipulating Controls" (page 27).

You can make a control invisible in several ways:

- Specifying its invisibility in the control resource.
- Passing a value of `false` in the `visible` parameter of `NewControl` (page 13).
- Calling `HideControl` (page 28).
- Calling `SetControlVisibility` (page 54). The setting takes effect the next time the control is drawn.

SPECIAL CONSIDERATIONS

The `ShowControl` function draws the control in its window, but the control can still be completely or partially obscured by overlapping windows or other objects.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

HideControl

Makes a visible control, and any latent embedded controls, invisible.

```
pascal void HideControl (ControlHandle theControl);
```

`theControl` A handle to the control to hide.

11/18/98 © Apple Computer, Inc.

DISCUSSION

The `HideControl` function makes the specified control invisible. This can be useful, for example, before adjusting a control's size and location. It also adds the control's rectangle to the window's update region, so that anything else that was previously obscured by the control will reappear on the screen. If the specified control has embedded controls, `HideControl` makes the embedded controls invisible as well. If the control is already invisible, `HideControl` has no effect.

If you call `HideControl` on a latent embedded control, it would not be displayed the next time `ShowControl` was called on its embedder control. For a discussion of latency, see "Manipulating Controls" (page 27).

To make the control visible again, you can use the functions `ShowControl` (page 27) or `SetControlVisibility` (page 54).

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

ActivateControl

Activates a control and any latent embedded controls.

```
pascal OSErr ActivateControl (ControlHandle inControl);
```

`inControl` A handle to the control to activate. Passing a window's root control activates all controls in that window.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

The `ActivateControl` function should be called instead of `HiliteControl` to activate a specified control and its latent embedded controls. For a discussion of latency, see "Manipulating Controls" (page 27).

You can use `ActivateControl` to activate all controls in a window by passing the window's root control in the `inControl` parameter.

If a control definition function supports activate events, it will receive a `kControlMsgActivate` message before redrawing itself in its active state.

11/18/98 © Apple Computer, Inc.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`DeactivateControl` (page 30).

"Embedding Controls" (page 17).

DeactivateControl

Deactivates a control and any latent embedded controls.

```
pascal OSErr DeactivateControl (ControlHandle inControl);
```

`inControl` A handle to the control to deactivate. Passing a window's root control deactivates all controls in that window.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

The `DeactivateControl` function should be called instead of `HiliteControl` to deactivate a specified control and its latent embedded controls. For a discussion of latency, see "Manipulating Controls" (page 27).

You can use `DeactivateControl` to deactivate all controls in a window by passing the window's root control in the `inControl` parameter.

If a control definition function supports activate events, it will receive a `kControlMsgActivate` message before redrawing itself in its inactive state.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`ActivateControl` (page 29).

"Embedding Controls" (page 17).

11/18/98 © Apple Computer, Inc.

IsControlActive

Returns whether a control is active.

```
pascal Boolean IsControlActive (ControlHandle inControl);
```

`inControl`	A handle to the control to be examined.
function result	Returns a Boolean value. If `true`, the control is active. If `false`, the control is inactive.

DISCUSSION

If you wish to determine whether a control is active, you should call `IsControlActive` instead of testing the `contrlHilite` field of the control structure.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SendControlMessage

Sends a message to a control definition function.

```
pascal SInt32 SendControlMessage (
                    ControlHandle inControl,
                    SInt16 inMessage,
                    SInt32 inParam);
```

`inControl`	A handle to the control that is to receive a low-level message.
`inMessage`	A bit field representing the message(s) you wish to send; see “Messages” (page 58).
`inParam`	The message-dependent data passed in the `param` parameter of the control definition function.
function result	Returns a signed 32-bit integer which contains varying data depending upon the message sent; see “Messages” (page 58).

11/18/98 © Apple Computer, Inc.

DISCUSSION

Your application does not normally need to call the `SendControlMessage` function. If you have a special need to call a control definition function directly, call `SendControlMessage` to access and manipulate the control's attributes.

Before calling `SendControlMessage`, you should determine whether the control supports the specific message you wish to send by calling `GetControlFeatures` (page 53) and examining the feature bit field returned. If there are no feature bits returned that correspond to the message you wish to send (for messages 0 through 12), you can assume that all system controls support that message.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`MyControlDefProc` (page 57).

Displaying Controls

The following Control Manager functions for displaying controls are new, changed, or not recommended with Appearance Manager 1.0:

- `DrawOneControl` (page 32) draws a control and any embedded controls that are currently visible in the specified window. Changed with Appearance Manager 1.0.
- `DrawControlInCurrentPort` (page 33) draws a control in the current graphics port. New with Appearance Manager 1.0.
- `SetUpControlBackground` (page 34) sets the background for a control. New with Appearance Manager 1.0.

DrawOneControl

Draws a control and any embedded controls that are currently visible in the specified window.

```
pascal void DrawOneControl (ControlHandle theControl);
```


11/18/98 © Apple Computer, Inc.

`theControl` A handle to the control to draw.

DISCUSSION

Although you should generally use the function `UpdateControls` to update controls, you can use the `DrawOneControl` function to update a single control. If an embedding hierarchy exists and the control passed in has embedded controls, `DrawOneControl` draws the control and embedded controls. If the root control for a window is passed in, the result is the same as if `DrawControls` was called.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

SEE ALSO

“Embedding Controls” (page 17).

DrawControlInCurrentPort

Draws a control in the current graphics port.

```
pascal void DrawControlInCurrentPort (ControlHandle inControl);
```

`inControl` A handle to the control to draw.

DISCUSSION

Typically, controls are automatically drawn in their owner’s graphics port with `DrawControls`, `DrawOneControl` (page 32), and `UpdateControls`. `DrawControlInCurrentPort` permits easy offscreen control drawing and printing. All standard system controls support this function.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

SetUpControlBackground

Sets the background for a control.

```
pascal OSErr SetUpControlBackground (
                    ControlHandle inControl,
                    SInt16 inDepth,
                    Boolean inIsColorDevice);
```

`inControl` A handle to the control whose background is to be set.

`inDepth` The bit depth (in pixels) of the current graphics port.

`inIsColorDevice`
A Boolean value. Set to `true` to indicate that you are drawing on a color device; set to `false` for a monochrome device.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

The `SetUpControlBackground` function allows you to set the background of a control. This function is typically called by control definition functions that are embedded in other controls. You might call `SetUpControlBackground` in response to an application-defined function installed in a user pane control; see "Defining Your Own User Pane Functions" (page 83). `SetUpControlBackground` ensures that the background color is always correct when calling `EraseRect` and `EraseRgn`. If your control spans multiple monitors, `SetUpControlBackground` should be called for each device that your control is drawing on; see "Graphics Devices" in *Imaging With QuickDraw* for more details on handling device loops.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Handling Events in Controls

The following Control Manager functions for handling events in controls are new, changed, or not recommended with Appearance Manager 1.0:

- `FindControlUnderMouse` (page 35) obtains the location of a mouse-down event in a control. New with Appearance Manager 1.0.

11/18/98 © Apple Computer, Inc.

- `FindControl` (page 36) obtains the location of a mouse-down event in a control. Not recommended with Appearance Manager 1.0.
- `HandleControlKey` (page 37) sends a keyboard event to a control with keyboard focus. New with Appearance Manager 1.0.
- `IdleControls` (page 38) performs idle event processing. New with Appearance Manager 1.0.
- `HandleControlClick` (page 38) responds to cursor movements in a control while the mouse button is down and returns the location of the next mouse-up event. New with Appearance Manager 1.0.
- `TrackControl` (page 41) responds to cursor movements in a control while the mouse button is down. Not recommended with Appearance Manager 1.0.

FindControlUnderMouse

Obtains the location of a mouse-down event in a control.

```
pascal ControlHandle FindControlUnderMouse (
                    Point inWhere,
                    WindowPtr inWindow,
                    SInt16 *outPart);
```

`inWhere` A point, specified in coordinates local to the window, where the mouse-down event occurred. Before calling `FindControlUnderMouse`, use the QuickDraw `GlobalToLocal` function to convert the point stored in the `where` field of the event structure (which describes the location of the mouse-down event) to coordinates local to the window.

`inWindow` A pointer to the window in which the mouse-down event occurred.

`outPart` Pass a pointer to a signed 16-bit integer value. On return, the value is set to the part code of the control part that was selected; see “Control Part Code Constants” (page 135).

function result Returns a handle to the control that was selected. If the mouse-down event did not occur over a control part, `FindControlUnderMouse` returns `nil`.

11/18/98 © Apple Computer, Inc.

DISCUSSION

You should call the `FindControlUnderMouse` function instead of `FindControl` (page 36) to determine whether a mouse-down event occurred in a control, particularly if an embedding hierarchy is present. `FindControlUnderMouse` will return a handle to the control even if no part was hit and can determine whether a mouse-down event has occurred even if the control is deactivated, while `FindControl` does not.

When a mouse-down event occurs, your application should call `FindControlUnderMouse` after using the Window Manager function `FindWindow` to ascertain that a mouse-down event has occurred in the content region of a window containing controls.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

“Embedding Controls” (page 17).

FindControl

Obtains the location of a mouse-down event in a control.

When the Appearance Manager is available, you should call `FindControlUnderMouse` (page 35) to determine the location of a mouse-down event in a control. `FindControlUnderMouse` will return a handle to the control even if no part was hit and can determine whether a mouse-down event has occurred even if the control is deactivated, while `FindControl` does not.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

HandleControlKey

Sends a keyboard event to a control with keyboard focus.

```
pascal SInt16 HandleControlKey (
                    ControlHandle inControl,
                    SInt16 inKeyCode,
                    SInt16 inCharCode,
                    SInt16 inModifiers);
```

`inControl` A handle to the control that currently has keyboard focus.

`inKeyCode` The virtual key code, derived from the event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.

`inCharCode` A character, derived from the event structure. The value that is generated depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource.

`inModifiers` Information from the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

function result Returns the part code that was hit during the keyboard event; see "Control Part Code Constants" (page 135).

DISCUSSION

If you have determined that a keyboard event has occurred in a given window, before calling the `HandleControlKey` function, call `GetKeyboardFocus` (page 43) to get the handle to the control that currently has keyboard focus. The `HandleControlKey` function passes the values specified in its `inKeyCode`, `inCharCode`, and `inModifiers` parameters to control definition functions that set the `kControlSupportsFocus` feature bit.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

IdleControls

Performs idle event processing.

```
pascal void IdleControls (WindowPtr inWindow);
```

`inWindow` A pointer to a window containing controls that support idle events.

DISCUSSION

Your application should call the `IdleControls` function to give idle time to any controls that want the `kControlMsgIdle` message. `IdleControls` calls the control with an idle event so the control can do idle-time processing. You should call `IdleControls` at least once in your event loop. See "Performing Idle Processing" (page 74) for more details on how a control definition function should handle idle processing.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

HandleControlClick

Responds to cursor movements in a control while the mouse button is down and returns the location of the next mouse-up event.

```
pascal ControlPartCode HandleControlClick (
                    ControlHandle inControl,
                    Point inWhere,
                    SInt16 inModifiers,
                    ControlActionUPP inAction);
```

`inControl` A handle to the control in which the mouse-down event occurred. Pass the control handle returned by `FindControl` or `FindControlUnderMouse`.

`inWhere` A point, specified in local coordinates, where the mouse-down event occurred. Supply the same point you passed to `FindControl` or `FindControlUnderMouse`.

11/18/98 © Apple Computer, Inc.

`inModifiers`	Information from the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.
`inAction`	A universal procedure pointer to an action function defining what action your application takes while the user holds down the mouse button. The value of the `inAction` parameter can be a valid `procPtr`, `nil`, or -1. A value of -1 indicates that the control should either perform auto tracking, or if it is incapable of doing so, do nothing (like `nil`). For custom controls, what you pass in this parameter depends on how you define the control. If the part index is greater than 128, the pointer must be of type `DragGrayRegionUPP` unless the control supports live feedback, in which case it should be a `ControlActionUPP`.
function result	Returns a value of type `ControlPartCode` identifying the control's part; see "Control Part Code Constants" (page 135).

DISCUSSION

Call the `HandleControlClick` function after a call to `FindControl` or `FindControlUnderMouse`. The `HandleControlClick` function should be called instead of `TrackControl` (page 41) to follow the user's cursor movements in a control and provide visual feedback until the user releases the mouse button. Unlike `TrackControl`, `HandleControlClick` allows modifier keys to be passed in so that the control may use these if the control (such as a list box or editable text field) is set up to handle its own tracking.

The visual feedback given by `HandleControlClick` depends on the control part in which the mouse-down event occurs. When highlighting is appropriate, for example, `HandleControlClick` highlights the control part (and removes the highlighting when the user releases the mouse button). When the user holds down the mouse button while the cursor is in an indicator (such as the scroll box of a scroll bar) and moves the mouse, `HandleControlClick` responds by dragging a dotted outline or a ghost image of the indicator. If the user releases the mouse button when the cursor is in an indicator such as the scroll box, `HandleControlClick` calls the control definition function to reposition the indicator.

While the user holds down the mouse button with the cursor in one of the standard controls, `HandleControlClick` performs the following actions, depending on the value you pass in the parameter `inAction`.

11/18/98 © Apple Computer, Inc.

- If you pass `nil` in the `inAction` parameter, `HandleControlClick` uses no action function and therefore performs no additional actions beyond highlighting the control or dragging the indicator. This is appropriate for push buttons, checkboxes, radio buttons, and the scroll box of a scroll bar.
- If you pass a pointer to an action function in the `inAction` parameter, it must define some action that your application repeats as long as the user holds down the mouse button. This is appropriate for the scroll arrows and gray areas of a scroll bar.
- If you pass `(ControlActionUPP)-1L` in the `inAction` parameter, `HandleControlClick` looks in the `contrlAction` field of the control structure for a pointer to the control's action function. This is appropriate when you are tracking the cursor in a pop-up menu. You can call `GetControlAction` to determine the value of this field, and you can call `SetControlAction` (page 48) to change this value. If the `contrlAction` field of the control structure contains a function pointer, `HandleControlClick` uses the action function it points to; if the field of the control structure also contains the value `(ControlActionUPP)-1L`, `HandleControlClick` calls the control definition function to perform the necessary action; you may wish to do this if you define your own control definition function for a custom control. If the field of the control structure contains the value `nil`, `HandleControlClick` performs no action.

Note
For `'CDEF'` resources that implement custom dragging, you usually call `HandleControlClick`, which returns 0 regardless of the user's changes of the control setting. To avoid this, you should use another method to determine whether the user has changed the control setting, for instance, comparing the control's value before and after your call to `HandleControlClick`.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`MyActionProc` (page 78).

11/18/98 © Apple Computer, Inc.

TrackControl

Responds to cursor movements in a control while the mouse button is down.

When the Appearance Manager is available, call `HandleControlClick` (page 38) instead of `TrackControl` to follow the user's cursor movements in a control and provide visual feedback until the user releases the mouse button. Unlike the `TrackControl` function, `HandleControlClick` also accepts modifier key information so that the control may take into account the current modifier key state if the control is set up to handle its own tracking.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

Handling Keyboard Focus

A control with **keyboard focus** receives keyboard events. The Dialog Manager tests to see which control has keyboard focus when a keyboard event is processed and sends the event to that control. If no control has keyboard focus, the keyboard event is discarded. Currently, the list box, clock, and editable text controls are the only standard system controls that support keyboard focus. A control retains keyboard focus if it is hidden or deactivated.

A **focus ring** is drawn around the control with keyboard focus. When creating your own controls, allow space for the focus ring. For more details on designing with focus rings, see *Mac OS 8 Human Interface Guidelines*.

Keyboard focus is only available if an embedding hierarchy has been established in the focusable control's window. The default focusing order is based on the order in which controls are added to the window. For more details on embedding hierarchies, see "Embedding Controls" (page 17).

The following Control Manager functions for handling keyboard focus are new with Appearance Manager 1.0:

- `SetKeyboardFocus` (page 42) sets the current keyboard focus to a specified control part for a window. New with Appearance Manager 1.0.
- `GetKeyboardFocus` (page 43) obtains a handle to the control with the current keyboard focus for a specified window. New with Appearance Manager 1.0.
- `AdvanceKeyboardFocus` (page 43) advances the keyboard focus to the next focusable control in a window. New with Appearance Manager 1.0.

11/18/98 © Apple Computer, Inc.

- `ReverseKeyboardFocus` (page 44) returns keyboard focus to the prior focusable control in a window. New with Appearance Manager 1.0.
- `ClearKeyboardFocus` (page 45) removes the keyboard focus for the currently focused control in a window. New with Appearance Manager 1.0.

SetKeyboardFocus

Sets the current keyboard focus to a specified control part for a window.

```
pascal OSErr SetKeyboardFocus (
                    WindowPtr inWindow,
                    ControlHandle inControl,
                    ControlFocusPart inPart);
```

`inWindow` A pointer to the window containing the control that is to receive keyboard focus.

`inControl` A handle to the control that is to receive keyboard focus.

`inPart` A part code specifying the part of a control to receive keyboard focus. To clear a control's keyboard focus, pass `kControlFocusNoPart`. See "Handling Keyboard Focus" (page 72).

function result A result code; see "Result Codes" (page 140).

DISCUSSION

The `SetKeyboardFocus` function sets the keyboard focus to a specified control part. The control to receive keyboard focus can be deactivated or invisible. This permits you to set the focus for an item in a dialog box before the dialog box is displayed.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetKeyboardFocus` (page 43).

11/18/98 © Apple Computer, Inc.

"Handling Keyboard Focus" (page 41).

GetKeyboardFocus

Obtains a handle to the control with the current keyboard focus for a specified window.

```
pascal OSErr GetKeyboardFocus (
                    WindowPtr inWindow,
                    ControlHandle* outControl);
```

`inWindow`	A pointer to the window for which to obtain keyboard focus.
`outControl`	Pass a pointer to a `ControlHandle` value. On return, the `ControlHandle` value is set to a handle to the control that currently has keyboard focus. Produces `nil` if no control has focus.
function result	A result code; see "Result Codes" (page 140).

DISCUSSION

The `GetKeyboardFocus` function returns the handle of the control with current keyboard focus within a specified window.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetKeyboardFocus` (page 42).

"Handling Keyboard Focus" (page 41).

AdvanceKeyboardFocus

Advances the keyboard focus to the next focusable control in a window.

```
pascal OSErr AdvanceKeyboardFocus (WindowPtr inWindow);
```


11/18/98 © Apple Computer, Inc.

`inWindow`	A pointer to the window for which to advance keyboard focus.
function result	A result code; see “Result Codes” (page 140).

DISCUSSION

The `AdvanceKeyboardFocus` function skips over deactivated and hidden controls until it finds the next focusable control in the window. If it does not find a focusable item, it simply returns.

When `AdvanceKeyboardFocus` is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in its `message` parameter and `kControlFocusNextPart` in its `param` parameter. In response to this message, your control definition function should change keyboard focus to its next part, the entire control, or remove keyboard focus from the control, depending upon the circumstances. See “Handling Keyboard Focus” (page 72) for a discussion of possible responses to this message.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`ReverseKeyboardFocus` (page 44).

“Handling Keyboard Focus” (page 41).

ReverseKeyboardFocus

Returns keyboard focus to the prior focusable control in a window.

```
pascal OSErr ReverseKeyboardFocus (WindowPtr inWindow);
```

`inWindow`	A pointer to the window for which to reverse keyboard focus.
function result	A result code; see “Result Codes” (page 140).

11/18/98 © Apple Computer, Inc.

DISCUSSION

The `ReverseKeyboardFocus` function reverses the progression of keyboard focus, skipping over deactivated and hidden controls until it finds the previous control to receive keyboard focus in the window.

When `ReverseKeyboardFocus` is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in its `message` parameter and `kControlFocusPrevPart` in its `param` parameter. In response to this message, your control definition function should change keyboard focus to its previous part, the entire control, or remove keyboard focus from the control, depending upon the circumstances. See "Handling Keyboard Focus" (page 72) for a discussion of possible responses to this message.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`AdvanceKeyboardFocus` (page 43).

"Handling Keyboard Focus" (page 41).

ClearKeyboardFocus

Removes the keyboard focus for the currently focused control in a window.

```
pascal OSErr ClearKeyboardFocus (WindowPtr inWindow);
```

`inWindow` A pointer to the window in which to clear keyboard focus.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

When the `ClearKeyboardFocus` function is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in its `message` parameter and `kControlFocusNoPart` in its `param` parameter. See "Handling Keyboard Focus" (page 72) for a discussion of possible responses to this message.

11/18/98 © Apple Computer, Inc.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

"Handling Keyboard Focus" (page 41).

Accessing and Changing Control Settings and Data

The following Control Manager functions for accessing and changing control settings and data are new, changed, or not recommended with Appearance Manager 1.0:

- `GetBestControlRect` (page 47) obtains a control's optimal size and text placement. New with Appearance Manager 1.0.
- `SetControlAction` (page 48) sets or changes the action function for a control. Changed with Appearance Manager 1.0.
- `SetControlColor` (page 48) customizes the color table for a control. Not recommended with Appearance Manager 1.0.
- `SetControlData` (page 49) sets control-specific data. New with Appearance Manager 1.0.
- `GetControlData` (page 50) obtains control-specific data. New with Appearance Manager 1.0.
- `GetControlDataSize` (page 52) obtains the size of a control's tagged data. New with Appearance Manager 1.0.
- `GetControlFeatures` (page 53) obtains the features a control supports. New with Appearance Manager 1.0.
- `SetControlFontStyle` (page 53) sets the font style for a control. New with Appearance Manager 1.0.
- `SetControlVisibility` (page 54) sets the visibility of a control, and any embedded controls, and specifies whether it should be drawn. New with Appearance Manager 1.0.
- `IsControlVisible` (page 55) returns whether a control is visible. New with Appearance Manager 1.0.

11/18/98 © Apple Computer, Inc.

GetBestControlRect

Obtains a control's optimal size and text placement.

```
pascal OSErr GetBestControlRect (
                    ControlHandle inControl,
                    Rect *outRect,
                    SInt16 *outBaseLineOffset);
```

`inControl` A handle to the control to be examined.

`outRect` Pass a pointer to an empty rectangle (0, 0, 0, 0). On return, the rectangle is set to the optimal size for the control. If the control doesn't support getting an optimal size rectangle, the control's bounding rectangle is passed back.

`outBaseLineOffset`
Pass a pointer to a signed 16-bit integer value. On return, the value is set to the offset from the bottom of control to the base of the text (usually a negative value). If the control doesn't support optimal sizing or has no text, 0 is passed back.

function result A result code; see "Result Codes" (page 140).

DISCUSSION

You can call the `GetBestControlRect` function to automatically position and size controls in accordance with human interface guidelines. This function is particularly helpful in determining the correct placement of control text whose length is not known until run-time. For example, the `StandardAlert` function uses `GetBestControlRect` to automatically size and position buttons in a newly created alert box.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

SetControlAction

Sets or changes the action function for a control.

```
pascal void SetControlAction (
                    ControlHandle theControl,
                    ControlActionUPP actionProc);
```

`theControl` A handle to the control whose action function is to be changed.

`actionProc` A universal procedure pointer to an action function defining what action your application takes while the user holds down the mouse button.

DISCUSSION

The `SetControlAction` function changes the `contrlAction` field of the control structure to point to the action function specified in the `actionProc` parameter. If the cursor is in the specified control, `HandleControlClick` (page 38) or `TrackControl` (page 41) call this action function when the user holds down the mouse button. You must provide the action function, and it must define some action to perform repeatedly as long as the user holds down the mouse button. `HandleControlUnderClick` and `TrackControl` always highlight and drag the control as appropriate.

Note
`SetControlAction` should be used to set the application-defined action function for providing live feedback for standard system scroll bar controls.

VERSION NOTES

Changed with Appearance Manager 1.0 to support live feedback.

SEE ALSO

`MyActionProc` (page 78).

SetControlColor

Customizes the color table for a control.

11/18/98 © Apple Computer, Inc.

When the Appearance Manager is available and you are using standard controls, colors are determined by the current theme. If you are creating your own control definition function, you can still set your own colors with the `SetControlColor` function.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

SetControlData

Sets control-specific data.

```
pascal OSErr SetControlData (
                    ControlHandle inControl,
                    ControlPartCode inPart,
                    ResType inTagName,
                    Size inSize,
                    Ptr inData);
```

`inControl` A handle to the control for which data is to be set.

`inPart` The part code of the control part for which data is to be set; see "Control Part Code Constants" (page 135). Passing `kControlEntireControl` indicates that either the control has no parts or the data is not tied to any specific part of the control.

`inTagName` A constant representing the control-specific data you wish to set; see "Control Data Tag Constants" (page 118).

`inSize` The size (in bytes) of the data pointed to by the `inData` parameter. For variable-length control data, pass the value returned in the `outMaxSize` parameter of `GetControlDataSize` (page 52) in the `inSize` parameter. The number of bytes must match the actual data size.

`inData` A pointer to a buffer allocated by your application. This buffer contains the data that you are sending to the control. After calling `SetControlData`, your application is responsible for disposing of this buffer, if necessary, as information is copied by control.

11/18/98 © Apple Computer, Inc.

function result A result code; see "Result Codes" (page 140). The result code `errDataNotSupported` indicates that the `inTagName` parameter is not valid.

DISCUSSION

The `SetControlData` function sets control-specific data represented by the value in the `inTagName` parameter to the data pointed to by the `inData` parameter. `SetControlData` could be used, for example, to switch a progress indicator from a determinate to indeterminate state. For a list of the control attributes that can be set, see "Control Data Tag Constants" (page 118).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetControlData` (page 50).

GetControlData

Obtains control-specific data.

```
pascal OSErr GetControlData (
                    ControlHandle inControl,
                    ControlPartCode inPart,
                    ResType inTagName,
                    Size inBufferSize,
                    Ptr inBuffer,
                    Size *outActualSize);
```

`inControl` A handle to the control to be examined.

`inPart` The part code of the control part from which data is to be obtained; see "Control Part Code Constants" (page 135). Passing `kControlEntireControl` indicates that either the control has no parts or the data is not tied to any specific part of the control.

11/18/98 © Apple Computer, Inc.

`inTagName`	A constant representing the control-specific data you wish to obtain; see "Control Data Tag Constants" (page 118).
`inBufferSize`	The size (in bytes) of the data pointed to by the `inBuffer` parameter. For variable-length control data, pass the value returned in the `outMaxSize` parameter of `GetControlDataSize` (page 52) in the `inBufferSize` parameter. The number of bytes must match the actual data size.
`inBuffer`	Pass a pointer to a buffer allocated by your application. On return, the buffer contains a copy of the control-specific data. If you pass `nil` on input, it is equivalent to calling `GetControlDataSize` (page 52). The actual size of the control-specific data will be returned in the `outActualSize` parameter. For variable-length data, the number of bytes must match the actual data size.
`outActualSize`	Pass a pointer to a `Size` value. On return, the value is set to the actual size of the data.
function result	A result code; see "Result Codes" (page 140). The result code `errDataNotSupported` indicates that the `inTagName` parameter is not valid.

DISCUSSION

The `GetControlData` function will only copy the amount of data specified in the `inBufferSize` parameter, but will tell you the actual size of the buffer so you will know if the data was truncated.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetControlData` (page 49).

11/18/98 © Apple Computer, Inc.

GetControlDataSize

Obtains the size of a control's tagged data.

```
pascal OSErr GetControlDataSize (
                    ControlHandle inControl,
                    ControlPartCode inPart,
                    ResType inTagName,
                    Size *outMaxSize);
```

`inControl` A handle to the control to be examined.

`inPart` The part code of the control part with which the data is associated; see "Control Part Code Constants" (page 135). Passing `kControlEntireControl` indicates that either the control has no parts or the data is not tied to any specific part of the control.

`inTagName` A constant representing the control-specific data whose size is to be obtained; see "Control Data Tag Constants" (page 118).

`outMaxSize` Pass a pointer to a `Size` value. On return, the value is set to the size (in bytes) of the control's tagged data. This value should be passed to `SetControlData` (page 49) and `GetControlData` (page 50) to allocate a sufficiently large buffer for variable-length data.

function result A result code; see "Result Codes" (page 140). The result code `errDataNotSupported` indicates that the `inTagName` parameter is not valid.

DISCUSSION

Pass the value returned in the `outMaxSize` parameter of `GetControlDataSize` in the `inBufferSize` parameter of `SetControlData` (page 49) and `GetControlData` (page 50) to allocate an adequate buffer for variable-length data.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

GetControlFeatures

Obtains the features a control supports.

```
pascal OSErr GetControlFeatures (
                    ControlHandle inControl,
                    UInt32 *outFeatures);
```

`inControl` A handle to the control to be examined.

`outFeatures` Pass a pointer to an unsigned 32-bit integer value. On return, the value contains a bit field specifying the features the control supports. For a list of the features a control may support, see "Specifying Which Appearance-Compliant Messages Are Supported" (page 68).

function result A result code; see "Result Codes" (page 140). The result code `errMsgNotSupported` indicates that the control does not support Appearance-compliant features.

DISCUSSION

The `GetControlFeatures` function obtains the Appearance-compliant features a control definition function supports, in response to a `kControlMsgGetFeatures` message.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SetControlFontStyle

Sets the font style for a control.

```
pascal OSErr SetControlFontStyle (
                    ControlHandle inControl,
                    const ControlFontStyleRec *inStyle);
```

`inControl` A handle to the control whose font style is to be set.

11/18/98 © Apple Computer, Inc.

`inStyle`	Pass a pointer to a `ControlFontStyleRec` (page 95) structure. If the `flags` field is cleared, the control uses the system font unless the control variant `kControlUsesOwningWindowsFontVariant` has been specified (control uses window font).
function result	A result code; see "Result Codes" (page 140).

DISCUSSION

The `SetControlFontStyle` function sets the font style for a given control. To specify the font for controls in a dialog box, it is generally easier to use the dialog font table resource. `SetControlFontStyle` allows you to override a control's default font (system or window font, depending upon whether the control variant `kControlUsesOwningWindowsFontVariant` has been specified). Once you have set a control's font with this function, you can cause the control to revert to its default font by passing a control font style structure with a cleared `flags` field in the `inStyle` parameter.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SetControlVisibility

Sets the visibility of a control, and any embedded controls, and specifies whether it should be drawn.

```
pascal OSErr SetControlVisibility (
                    ControlHandle inControl,
                    Boolean inIsVisible,
                    Boolean inDoDraw);
```

`inControl`	A handle to the control whose visibility is to be set.
`inIsVisible`	A Boolean value indicating whether the control is visible or invisible. If you set this value to `true`, the control will be visible. If `false`, the control will be invisible. If you wish to show a control (and latent embedded subcontrols) but do not want to cause screen drawing, pass `true` for this parameter and false in the `inDoDraw` parameter.

11/18/98 © Apple Computer, Inc.

`inDoDraw`	A Boolean value indicating whether the control should be drawn or erased. If `true`, the control's display on the screen should be updated (drawn or erased) based on the value passed in the `inIsVisible` parameter. If `false`, the display will not be updated.
function result	A result code; see "Result Codes" (page 140).

DISCUSSION

You should call the `SetControlVisibility` function instead of setting the `contrlVis` field of the control structure to set the visibility of a control and specify whether it will be drawn. If the control has embedded controls, `SetControlVisibility` allows you to set their visibility and specify whether or not they will be drawn. If you wish to show a control but do not want it to be drawn onscreen, pass `true` in the `inIsVisible` parameter and false in the `inDoDraw` parameter.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

IsControlVisible

Returns whether a control is visible.

```
pascal Boolean IsControlVisible (ControlHandle inControl);
```

`inControl`	A handle to the control to be examined.
function result	Returns a Boolean value. If `true`, the control is visible. If `false`, the control is hidden.

DISCUSSION

If you wish to determine whether a control is visible, call `IsControlVisible` instead of testing the `contrlVis` field of the control structure.

11/18/98 © Apple Computer, Inc.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Defining Your Own Control Definition Function

A control definition function determines how a control generally looks and behaves. Various Control Manager functions call a control definition function whenever they need to perform a control-dependent action, such as drawing the control on the screen. In addition to standard control definition functions, defined by the system, you can make your own custom control definition functions.

The Control Manager calls the Resource Manager to access a control definition function with the given resource ID; for a description of how to derive a control definition function ID, see “Control Definition IDs” (page 106). The Resource Manager reads a control definition function into memory and returns a handle to it. The Control Manager stores this handle in the `contrlDefProc` field of the control structure.

When various Control Manager functions need to perform a type-dependent action on the control, they call the control definition function and pass it the variation code for its type as a parameter. You can define your own variation codes; this allows you to use one `'CDEF'` resource to handle several variations of the same general control. See `'CNTL'` (page 100) for further discussion of controls, their resources, and their IDs.

If you choose to provide your own control definition functions, these functions should apply the user’s desktop color choices the same way the standard control definition functions do. You can use control color tables of any desired size and define their contents in any way you wish, except that part indices and messages 0 through 127 are reserved for system definition.

The following Control Manager function for defining your own control definition function is changed with Appearance Manager 1.0:

- `MyControlDefProc` (page 57) defines a custom control. Changed with Appearance Manager 1.0.

11/18/98 © Apple Computer, Inc.

MyControlDefProc

If you wish to define new, nonstandard controls for your application, you must write a control definition function and store it in a resource file as a resource of type 'CDEF'.

The Control Manager declares the type for an application-defined control definition function as follows:

```
typedef pascal SInt32 (*ControlDefProcPtr)(
                    SInt16 varCode,
                    ControlHandle theControl,
                    ControlDefProcMessage message,
                    SInt32 param);
```

The Control Manager defines the data type `ControlDefUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlDefUPP;
```

You typically use the `NewControlDefProc` macro like this:

```
ControlDefUPP myControlDefUPP;
myControlDefUPP = NewControlDefProc (MyControl);
```

You typically use the `CallControlDefProc` macro like this:

```
CallControlDefProc(myControlDefUPP, varCode, theControl, message, param);
```

Here's how to declare the function `MyControlDefProc`:

```
pascal SInt32 MyControlDefProc (
                    SInt16 varCode,
                    ControlHandle theControl,
                    ControlDefProcMessage message,
                    SInt32 param);
```

`varCode` The control's variation code.

`theControl` A handle to the control that the operation will affect.

`message` A code for the task to be performed. The `message` parameter contains one of the task codes defined in "Messages" (page 58). The subsections that follow explain each of these tasks in detail.

11/18/98 © Apple Computer, Inc.

`param`	Data associated with the task specified by the `message` parameter. If the task requires no data, this parameter is ignored.
function result	The function results that your control definition function returns depend on the value that the Control Manager passes in the `message` parameter.

DISCUSSION

The Control Manager calls your control definition function under various circumstances; the Control Manager uses the `message` parameter to inform your control definition function what action it must perform. The data that the Control Manager passes in the `param` parameter, the action that your control definition function must undertake, and the function results that your control definition function returns all depend on the value that the Control Manager passes in the `message` parameter. The rest of this section describes how to respond to the various values that the Control Manager passes in the `message` parameter.

VERSION NOTES

Changed with Appearance Manager 1.0 to support new control definition messages.

Messages

The Control Manager passes constants of type `ControlDefProcMessage` to indicate the action your control definition function must perform.

```
enum {
    drawCntl                    = 0,
    testCntl                    = 1,
    calcCRgns                   = 2,
    initCntl                    = 3,
    dispCntl                    = 4,
    posCntl                     = 5,
    thumbCntl                   = 6,
    dragCntl                    = 7,
    autoTrack                   = 8,
    calcCntlRgn                 = 10,
```


11/18/98 © Apple Computer, Inc.

```
    calcThumbRgn               = 11,
    kControlMsgDrawGhost       = 13,
    kControlMsgCalcBestRect    = 14,
    kControlMsgHandleTracking  = 15,
    kControlMsgFocus           = 16,
    kControlMsgKeyDown         = 17,
    kControlMsgIdle            = 18,
    kControlMsgGetFeatures     = 19,
    kControlMsgSetData         = 20,
    kControlMsgGetData         = 21,
    kControlMsgActivate        = 22,
    kControlMsgSetUpBackground = 23,
    kControlMsgSubValueChanged = 25,
    kControlMsgCalcValueFromPos = 26,
    kControlMsgTestNewMsgSupport= 27,
    kControlMsgSubControlAdded = 28,
    kControlMsgSubControlRemoved= 29
};
typedef SInt16 ControlDefProcMessage;
```

Constant descriptions

`drawCntl`	Draw the entire control or part of a control.
`testCntl`	Test where the mouse has been pressed.
`calcCRgns`	Calculate the region for the control or the indicator in 24-bit systems. This message is obsolete in Mac OS 7.6 and later.
`initCntl`	Perform additional control initialization.
`dispCntl`	Perform additional control disposal actions.
`posCntl`	Move and update the indicator setting.
`thumbCntl`	Calculate the parameters for dragging the indicator.
`dragCntl`	Perform customized dragging (of the control or its indicator).
`autoTrack`	Execute the specified action function.
`calcCntlRgn`	Calculate the control region in 32-bit systems.
`calcThumbRgn`	Calculate the indicator region in 32-bit systems.
`kControlMsgDrawGhost`	Draw a ghost image of the indicator. Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

`kControlMsgCalcBestRect`
: Calculate the optimal control rectangle. Available with Appearance Manager 1.0 and later.

`kControlMsgHandleTracking`
: Perform custom tracking. Available with Appearance Manager 1.0 and later.

`kControlMsgFocus` Handle keyboard focus. Available with Appearance Manager 1.0 and later.

`kControlMsgKeyDown` Handle keyboard events. Available with Appearance Manager 1.0 and later.

`kControlMsgIdle` Perform idle processing. Available with Appearance Manager 1.0 and later.

`kControlMsgGetFeatures`
: Specify which Appearance-compliant messages are supported. Available with Appearance Manager 1.0 and later.

`kControlMsgSetData`
: Set control-specific data. Available with Appearance Manager 1.0 and later.

`kControlMsgGetData`
: Get control-specific data. Available with Appearance Manager 1.0 and later.

`kControlMsgActivate`
: Handle activate and deactivate events. Available with Appearance Manager 1.0 and later.

`kControlMsgSetUpBackground`
: Set the control's background color or pattern (only available if the control supports embedding). Available with Appearance Manager 1.0 and later.

`kControlMsgSubValueChanged`
: Be informed that the value of a subcontrol embedded in the control has changed; this message is useful for radio groups. Available with Appearance 1.0.1 and later.

`kControlMsgCalcValueFromPos`
: Support live feedback while dragging the indicator and calculate the control value based on the new indicator region. Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

`kControlMsgTestNewMsgSupport`
: Specify whether Appearance-compliant messages are supported. Available with Appearance Manager 1.0 and later.

`kControlMsgSubControlAdded`
: Be informed that a subcontrol has been embedded in the control. Available with Appearance 1.0.1 and later.

`kControlMsgSubControlRemoved`
: Be informed that a subcontrol is about to be removed from the control. Available with Appearance 1.0.1 and later.

Drawing the Control or Its Part

When the Control Manager passes the value `drawCntl` in the `message` parameter, your control definition function should respond by drawing the indicator or the entire control.

The Control Manager passes one of the following drawing constants in the low word of the `param` parameter to specify whether the user is drawing an indicator or the whole control. The high-order word of the `param` parameter may contain undefined data; therefore, evaluate only the low-order word of this parameter.

```
enum {
    kDrawControlEntireControl   = 0,
    kDrawControlIndicatorOnly   = 129
};
```

Constant descriptions

`kDrawControlEntireControl`
: Draw the entire control.

`kDrawControlIndicatorOnly`
: Draw the indicator only.

With the exception of part code 128, which is reserved for future use and should not be used, any other value indicates a part code for the control.

If the specified control is visible, your control definition function should draw the control (or the part specified in the `param` parameter) within the control's rectangle. If the control is invisible (that is, if its `contrlVis` field is set to 0), your control definition function does nothing.

11/18/98 © Apple Computer, Inc.

When drawing the control or its part, take into account the current values of its `contrlHilite` and `contrlValue` fields in the control structure.

If the part code for your control's indicator is passed in `param`, assume that the indicator hasn't moved; the Control Manager, for example, may be calling your control definition function so that you may simply highlight the indicator. However, when your application calls `SetControlValue`, `SetControlMinimum`, and `SetControlMaximum`, they in turn may call your control definition function with the `drawCntl` message to redraw the indicator. Since these functions have no way of determining what part code you chose for your indicator, they all pass 129 in `param`, meaning that you should move your indicator. Your control definition function must detect this part code as a special case and remove the indicator from its former location before drawing it. If your control has more than one indicator, you should interpret 129 to mean all indicators.

When sent the message `drawCntl`, your control definition function should return 0 as its function result.

Testing Where the Mouse-Down Event Occurs

When the Control Manager passes the value for the `testCntl` constant in the `message` parameter, your control definition function should respond by determining whether a specified point is in a visible control.

The Control Manager passes a point (in local coordinates) in the `param` parameter. The point's vertical coordinate is contained in the high-order word of the long integer, and horizontal coordinate is contained in the low-order word.

Your control definition function should return the part code of the part that contains the specified point; it should return 0 if the point is outside the control or if the control is inactive.

Calculating the Control and Indicator Regions on 24-Bit Systems

When the Control Manager passes the value for the `calcCRgns` constant in the `message` parameter, your control definition function should calculate the region passed in the `param` parameter for the specified control or its indicator.

The Control Manager passes a QuickDraw region handle in the `param` parameter. If the high-order bit of `param` is set, the region requested is that of the control's indicator; otherwise, the region requested is that of the entire control. Your control definition function should clear the high bit of the region handle before calculating the region.

11/18/98 © Apple Computer, Inc.

When passed this message, your control definition function should always return 0, and it should express the region in the local coordinate system of the control's window.

IMPORTANT

The `calcCRgns` message will never be sent to any system running on 32-bit mode and is therefore obsolete in Mac OS 7.6 and later. The `calcCntlRgn` and `calcThumbRgn` messages will be sent instead.

Calculating the Control and Indicator Regions on 32-Bit Systems

When the Control Manager passes the values for the `calcCntlRgn` or `calcThumbRgn` constants in the `message` parameter, your control definition function should calculate the region for the specified control or its indicator using the QuickDraw region handle passed in the `param` parameter .

If the Control Manager passes the value for the `calcThumbRgn` constant in the `message` parameter, calculate the region occupied by the indicator. If the Control Manager passes the value for the `calcCntlRgn` constant in the `message` parameter, calculate the region for the entire control.

When passed this message, your control definition function should always return 0, and it should express the region in the local coordinate system of the control's window.

Performing Additional Control Initialization

After initializing fields of a control structure as appropriate when creating a new control, the Control Manager passes `initCntl` in the `message` parameter to give your control definition function the opportunity to perform any type-specific initialization you may require. For example, the standard control definition function for scroll bars allocates space for a region to hold the scroll box and stores the region handle in the `contrlData` field of the new control structure.

When passed the value for the `initCntl` constant in the `message` parameter, your control definition function should ignore the `param` parameter and return 0 as a function result.

11/18/98 © Apple Computer, Inc.

Performing Additional Control Disposal Actions

The function `DisposeControl` (page 15) passes `dispCntl` in the `message` parameter to give your control definition function the opportunity to carry out any additional actions when disposing of a control. For example, the standard definition function for scroll bars releases the memory occupied by the scroll box region, whose handle is kept in the `contrlData` field of the control structure.

When passed the value for the `dispCntl` constant in the `message` parameter, your control definition function should ignore the `param` parameter and return 0 as a function result.

Dragging the Control or Its Indicator

When a mouse-up event occurs in the indicator of a control, the `HandleControlClick` (page 38) or `TrackControl` (page 41) functions call your control definition function and pass `posCntl` in the `message` parameter. In this case, the Control Manager passes a point (in coordinates local to the control's window) in the `param` parameter that specifies the vertical and horizontal offset, in pixels, by which your control definition function should move the indicator from its current position. Typically, this is the offset between the points where the cursor was when the user pressed and released the mouse button while dragging the indicator. The point's vertical offset is contained in the high-order word of the `param` parameter, and its horizontal offset is contained in the low-order word.

Your definition function should calculate the control's new setting based on the given offset and then, to reflect the new setting, redraw the control and update the `contrlValue` field in the control structure. Your control definition function should ignore the `param` parameter and return 0 as a function result.

Calculating Parameters for Dragging the Indicator

When the Control Manager passes the value for `thumbCntl` in the `message` parameter, your control definition function should respond by calculating values analogous to the `limitRect`, `slopRect`, and `axis` parameters of `DragControl` that constrain how the indicator is dragged. On entry, the fields `param->limitRect.top` and `param->limitRect.left` contain the point where the mouse-down event first occurred.

The Control Manager passes a pointer to a structure of type `IndicatorDragConstraint` in the `param` parameter:

11/18/98 © Apple Computer, Inc.

```
struct IndicatorDragConstraint {
    Rect            limitRect;
    Rect            slopRect;
    DragConstraint  axis;
};
typedef struct IndicatorDragConstraint IndicatorDragConstraint;
typedef IndicatorDragConstraint *IndicatorDragConstraintPtr;
typedef IndicatorDragConstraintPtr *IndicatorDragConstraintHandle;
```

Field descriptions

`limitRect` A pointer to a rectangle—whose coordinates should normally coincide with or be contained in the window's content region—delimiting the area in which the user can drag the control's outline.

`slopRect` A pointer to a rectangle that allows some extra space for the user to move the mouse while still constraining the control within the rectangle specified in the `limitRect` parameter.

`axis` The axis along which the user may drag the control's outline.

Your definition function should store the appropriate values into the fields of the structure pointed to by the `param` parameter; they're analogous to the similarly named parameters of the Window Manager function `DragGrayRgn`.

Your control definition function should return 0 as function result.

Performing Custom Dragging

When the Control Manager passes the value for the `dragCntl` constant in the `message` parameter, the `param` parameter typically contains a custom dragging constant with one of the following values to specify whether the user is dragging an indicator or the whole control:

```
enum {
    kDragControlEntireControl   = 0,
    kDragControlIndicator       = 1
};
```


11/18/98 © Apple Computer, Inc.

Constant descriptions

`kDragControlEntireControl`
Dragging the entire control.

`kDragControlIndicator`
Dragging the indicator.

Note
When the Appearance Manager is present, the message `kControlMsgHandleTracking` should be sent instead of `dragCntl` to handle any custom tracking; see "Performing Custom Tracking" (page 71).

If you want to use the Control Manager's default method of dragging, which is to call `DragControl` to drag the control or the Window Manager function `DragGrayRgn` to drag its indicator, return 0 as the function result for your control definition function.

If your control definition function returns a nonzero value, your control definition function (not the Control Manager) must drag the specified control (or its indicator) to follow the cursor until the user releases the mouse button. If the user drags the entire control, your definition function should use the function `MoveControl` to reposition the control to its new location after the user releases the mouse button. If the user drags the indicator, your definition function must calculate the control's new setting (based on the pixel offset between the points where the cursor was when the user pressed and released the mouse button while dragging the indicator) and then, to reflect the new setting, redraw the control and update the `contrlValue` field in the control structure. Note that, in this case, the functions `HandleControlClick` (page 38) and `TrackControl` (page 41) return 0 whether or not the user changes the indicator's position. Thus, you must determine whether the user has changed the control's setting by another method, for instance, by comparing the control's value before and after the call to `HandleControlClick`.

Executing an Action Function

The only way to specify actions in response to all mouse-down events in a control or its indicator is to define your own control definition function that specifies an action function. When you create the control, your control definition function must first respond to the `initCntl` message by storing `(ControlDefUPP)-1L` in the `contrlAction` field of the control structure. (The Control Manager sends the `initCntl` message to your control definition function after initializing the fields of a new control structure.) Then, when your

11/18/98 © Apple Computer, Inc.

application passes `(ControlActionUPP)-1L` in the `actionProc` parameter of `HandleControlClick` (page 38) or `TrackControl` (page 41), `HandleControlClick` calls your control definition function with the `autoTrack` message. The Control Manager passes the part code of the part where the mouse-down event occurs in the `param` parameter. Your control definition function should then use this information to respond as an action function would.

Note
For the `autoTrack` message, the high-order word of the `param` parameter may contain undefined data; therefore, evaluate only the low-order word of this parameter.

If the mouse-down event occurs in an indicator of a control that supports live feedback, your action function should take two parameters (a handle to the control and the part code of the control where the mouse-down event first occurred). This action function is the same one you would use to define actions to be performed in control part codes in response to a mouse-down event; see `MyActionProc` (page 78).

If the mouse-down event occurs in an indicator of a control that does not support live feedback, your action function should take no parameters, because the user may move the cursor outside the indicator while dragging it; see `MyIndicatorActionProc` (page 80).

Specifying Whether Appearance-Compliant Messages Are Supported

If your control definition function supports Appearance-compliant messages, it should return `kControlSupportsNewMessages` as a function result when the Control Manager passes `kControlMsgTestNewMsgSupport` in the `message` parameter.

```
enum{
    kControlSupportsNewMessages = ' ok '
};
```

Constant description

`kControlSupportsNewMessages`
The control definition function supports new messages introduced with Mac OS 8 and the Appearance Manager.

11/18/98 © Apple Computer, Inc.

Specifying Which Appearance-Compliant Messages Are Supported

If your control definition function supports Appearance-compliant messages, it should return a bit field of the features it supports in response to the `kControlMsgGetFeatures` message. Your control definition function should ignore the `param` parameter.

The bit field returned by your control definition function should be composed of one or more of the following bits:

```
enum{
    kControlSupportsGhosting    = 1 << 0,
    kControlSupportsEmbedding   = 1 << 1,
    kControlSupportsFocus       = 1 << 2,
    kControlWantsIdle           = 1 << 3,
    kControlWantsActivate       = 1 << 4,
    kControlHandlesTracking     = 1 << 5,
    kControlSupportsDataAccess  = 1 << 6,
    kControlHasSpecialBackground= 1 << 7,
    kControlGetsFocusOnClick    = 1 << 8,
    kControlSupportsCalcBestRect= 1 << 9,
    kControlSupportsLiveFeedback= 1 << 10,
    kControlHasRadioBehavior    = 1 << 11
};
```

Constant descriptions

`kControlSupportsGhosting`
: If this bit (bit 0) is set, the control definition function supports the `kControlMsgDrawGhost` message.

`kControlSupportsEmbedding`
: If this bit (bit 1) is set, the control definition function supports the `kControlMsgSubControlAdded` and `kControlMsgSubControlRemoved` messages.

`kControlSupportsFocus`
: If this bit (bit 2) is set, the control definition function supports the `kControlMsgKeyDown` message. If this bit and the `kControlGetsFocusOnClick` bit are set, the control definition function supports the `kControlMsgFocus` message.

`kControlWantsIdle`
: If this bit (bit 3) is set, the control definition function supports the `kControlMsgIdle` message.

11/18/98 © Apple Computer, Inc.

`kControlWantsActivate`
: If this bit (bit 4) is set, the control definition function supports the `kControlMsgActivate` message.

`kControlHandlesTracking`
: If this bit (bit 5) is set, the control definition function supports the `kControlMsgHandleTracking` message.

`kControlSupportsDataAccess`
: If this bit (bit 6) is set, the control definition function supports the `kControlMsgGetData` and `kControlMsgSetData` messages.

`kControlHasSpecialBackground`
: If this bit (bit 7) is set, the control definition function supports the `kControlMsgSetUpBackground` message.

`kControlGetsFocusOnClick`
: If this bit (bit 8) and the `kControlSupportsFocus` bit are set, the control definition function supports the `kControlMsgFocus` message.

`kControlSupportsCalcBestRect`
: If this bit (bit 9) is set, the control definition function supports the `kControlMsgCalcBestRect` message.

`kControlSupportsLiveFeedback`
: If this bit (bit 10) is set, the control definition function supports the `kControlMsgCalcValueFromPos` message.

`kControlHasRadioBehavior`
: If this bit (bit 11) is set, the control definition function supports radio button behavior and can be embedded in a radio group control. This constant is available with Appearance 1.0.1 and later.

Drawing a Ghost Image of the Indicator

If your control definition function supports indicator ghosting, it should return `kControlSupportsGhosting` as one of the feature bits in response to a `kControlMsgGetFeatures` message. If this bit is set and the control indicator is being tracked, the Control Manager calls your control definition function and passes `kControlMsgDrawGhost` in the `message` parameter. A handle to the region where the ghost should be drawn will be passed in the `param` parameter.

11/18/98 © Apple Computer, Inc.

Your control definition function should respond by redrawing the control with the ghosted indicator at the specified location and should return 0 as its function result.

Note
The ghost indicator should always be drawn before the actual indicator so that it appears underneath the actual indicator.

Calculating the Optimal Control Rectangle

If your control definition function supports calculating the optimal dimensions of the control rectangle, it should return `kControlSupportsCalcBestRect` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set and `GetBestControlRect` (page 47) is called, the Control Manager will call your control definition function and pass `kControlMsgCalcBestRect` in the `message` parameter. The Control Manager passes a pointer to a control size calculation structure in the `param` parameter.

Your control definition function should respond by calculating the width and height of the optimal control rectangle and adjusting the rectangle by setting the `height` and `width` fields of the control size calculation structure to the appropriate values. If your control definition function displays text, it should pass in the offset from the bottom of control to the base of the text in the `baseLine` field of the structure. Your control definition function should return the offset value stored in the structure's `baseLine` field.

The control size calculation structure is a structure of type `ControlCalcSizeRec`:

```
struct ControlCalcSizeRec {
    SInt16  height;
    SInt16  width;
    SInt16  baseLine;
};
typedef struct ControlCalcSizeRec ControlCalcSizeRec;
typedef ControlCalcSizeRec *ControlCalcSizePtr;
```

Field descriptions

`height`	The optimal height (in pixels) of the control's bounding rectangle.
`width`	The optimal width (in pixels) of the control's bounding rectangle.

11/18/98 © Apple Computer, Inc.

baseLine	The offset from the bottom of the control to the base of the text. This value is generally negative.

Performing Custom Tracking

If your control definition function supports custom tracking, it should return kControlHandlesTracking as one of the feature bits in response to a kControlMsgGetFeatures message. If this bit is set and a mouse-down event occurs in your control, TrackControl (page 41) or HandleControlClick (page 38) calls your control definition function and passes kControlMsgHandlesTracking in the message parameter. The Control Manager passes a pointer to a control tracking structure in the param parameter. Your control definition function should respond appropriately and return the part code that was hit, or kControlNoPart if the mouse-down event occurred outside the control; see "Control Part Code Constants" (page 135).

The control tracking structure is a structure of type ControlTrackingRec:

```
struct ControlTrackingRec {
    Point               startPt;
    SInt16              modifiers;
    ControlActionUPP    action;
};
typedef struct ControlTrackingRec ControlTrackingRec;
typedef ControlTrackingRec *ControlTrackingPtr;
```

Field descriptions

startPt	The location of the cursor at the time the mouse button was first pressed, in local coordinates. Your application retrieves this point from the where field of the event structure.
modifiers	The constant in the modifiers field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.
action	A pointer to an action function defining what action your application takes while the user holds down the mouse button. The value of the actionProc parameter can be a valid procPtr, nil, or -1. A value of -1 indicates that the control should either perform auto tracking, or if it is incapable of doing so, do nothing (like nil).

11/18/98 © Apple Computer, Inc.

Handling Keyboard Focus

If your control definition function can change its keyboard focus, it should set `kControlSupportsFocus` and `kControlGetsFocusOnClick` as feature bits in response to a `kControlMsgGetFeatures` message. If these bits are set and the `AdvanceKeyboardFocus` (page 43), `ReverseKeyboardFocus` (page 44), `ClearKeyboardFocus` (page 45), or `SetKeyboardFocus` (page 42) function is called, the Control Manager calls your control definition function and passes `kControlMsgFocus` in the `message` parameter.

The Control Manager passes one of the control focus part code constants described below or a valid part code in the `param` parameter. Your control definition function should respond by adjusting the focus accordingly.

Your control definition function should return the control focus part code or actual control part that was focused on. Return `kControlFocusNoPart` if your control does not accept focus or has just relinquished it. Return a nonzero part code to indicate that your control received keyboard focus. Your control definition function is responsible for maintaining which part is focused.

```
enum {
    kControlFocusNoPart     = 0,
    kControlFocusNextPart   = -1,
    kControlFocusPrevPart   = -2
};
typedef SInt16 ControlFocusPart;
```

Constant descriptions

`kControlFocusNoPart`

Your control definition function should relinquish its focus and return `kControlFocusNoPart`. It might respond by deactivating its text edit handle and erasing its focus ring. If the control is at the end of its subparts, it should return `kControlFocusNoPart`. This tells the focusing mechanism to jump to the next control that supports focus.

`kControlFocusNextPart`

Your control definition function should change keyboard focus to its next part, the entire control, or remove keyboard focus from the control, depending upon the circumstances.
For multiple part controls that already had keyboard focus, the next part of the control would receive keyboard focus

11/18/98 © Apple Computer, Inc.

when `kControlFocusNextPart` was passed in the `param` parameter. For example, a clock control with keyboard focus would change its focus to the left-most element of the control (the month field).
For single-part controls that did not have keyboard focus and are now receiving it, the entire control would receive keyboard focus when `kControlFocusNextPart` was passed in the `param` parameter.
For single-part controls that already had keyboard focus and are now losing it, the entire control would lose keyboard focus.
If you are passed `kControlFocusNextPart` and have run out of parts, return `kControlFocusNoPart` to indicate that the user tabbed past the control.

`kControlFocusPrevPart`

Your control definition function should change keyboard focus to its previous part, the entire control, or remove keyboard focus from the control, depending upon the circumstances.
For multiple part controls that already had keyboard focus, the previous part of the control would receive keyboard focus when `kControlFocusPrevPart` was passed in the `param` parameter. For example, a clock control with keyboard focus would change its focus to the right-most element of the control (the year field).
For single-part controls that did not have keyboard focus and are now receiving it, the entire control would receive keyboard focus when `kControlFocusNextPart` was passed in the `param` parameter.
For single-part controls that already had keyboard focus and are now losing it, the entire control would lose keyboard focus.
If you are passed `kControlFocusPrevPart` and have run out of parts, return `kControlFocusNoPart` to indicate that the user tabbed past the control.

`<part code>` Your control definition function should focus on the specified part code. Your function can interpret this in any way it wishes.

11/18/98 © Apple Computer, Inc.

Handling Keyboard Events

If your control definition function can handle keyboard events, it should return `kControlSupportsFocus`—every control that supports keyboard focus must also be able to handle keyboard events—as one of the feature bits in response to a `kControlMsgGetFeatures` message. If this bit is set, the Control Manager will pass `kControlMsgKeyDown` in the `message` parameter. The Control Manager passes a pointer to a control key down structure in the `param` parameter. Your control definition function should respond by processing the keyboard event as appropriate and return 0 as the function result.

The control key down structure is a structure of type `ControlKeyDownRec`:

```
struct ControlKeyDownRec {
    SInt16  modifiers;
    SInt16  keyCode;
    SInt16  charCode;
};
typedef struct ControlKeyDownRec ControlKeyDownRec;
typedef ControlKeyDownRec *ControlKeyDownPtr;
```

Field descriptions

`modifiers` The constant in the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.

`keyCode` The virtual key code derived from the event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.

`charCode` A particular character derived from the event structure. This value depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource.

Performing Idle Processing

If your control definition function can perform idle processing, it should return `kControlWantsIdle` as one of the feature bits in response to a `kControlMsgGetFeatures` message. If this bit is set and `IdleControls` (page 38) is called for the window your control is in, the Control Manager will pass `kControlMsgIdle` in the `message` parameter. Your control definition function should ignore the `param` parameter and respond appropriately. For example,

11/18/98 © Apple Computer, Inc.

indeterminate progress indicators and asynchronous arrows use idle time to perform their animation.

Your control definition function should return 0 as the function result.

Getting and Setting Control-Specific Data

If your control definition function supports getting and setting control-specific data, it should return `kControlSupportsDataAccess` as one of its features bits in response to the `kControlMsgGetFeatures` message. If this bit is set, the Control Manager will call your control definition function and pass `kControlMsgSetData` in the `message` parameter when `SetControlData` (page 49) is called, and will pass `kControlMsgGetData` in the `message` parameter when `GetControlData` (page 50) and `GetControlDataSize` (page 52) are called. The Control Manager passes a pointer to a control data access structure in the `param` parameter. Your definition function should respond by filling out the structure and returning an operating system status message as the function result.

The control data access structure is a structure of type `ControlDataAccessRec`:

```
struct ControlDataAccessRec{
    ResType     tag;
    ResType     part;
    Size        size;
    Ptr         dataPtr;
};
typedef struct ControlDataAccessRec ControlDataAccessRec;
typedef ControlDataAccessRec *ControlDataAccessPtr;
```

Field descriptions

`tag` A constant representing a piece of data that is passed in (in response to a `kControlMsgSetData` message) or returned (in response to a `kControlMsgGetData` message); see “Control Data Tag Constants” (page 118) for a description of these constants. The control definition function should return `errDataNotSupported` if the value in the `tag` parameter is unknown or invalid.

`part` The part of the control that this data should be applied to. If the information is not tied to a specific part of the control or the control has no parts, pass 0.

11/18/98 © Apple Computer, Inc.

`size`	On entry, the size of the buffer pointed to by the `dataPtr` field. In response to a `kControlMsgGetData` message, this field should be adjusted to reflect the actual size of the data that the control is maintaining. If the size of the buffer being passed in is smaller than the actual size of the data, the control definition function should return `errDataSizeMismatch`.
`dataPtr`	A pointer to a buffer to read or write the information requested. In response to a `kControlMsgGetData` message, this field could be `nil`, indicating that you wish to return the size of the data in the `size` field.

Handling Activate and Deactivate Events

If your control definition function wants to be informed whenever it is being activated or deactivated, it should return `kControlWantsActivate` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set and your control definition function is being activated or deactivated, the Control Manager calls it and passes `kControlMsgActivate` in the `message` parameter. The Control Manager passes a 0 or 1 in the `param` parameter. A value of 0 indicates that the control is being deactivated; 1 indicates that it is being activated.

Your control definition function should respond by performing any special processing before the user pane becomes activated or deactivated, such as deactivating its `TEHandle` or `ListHandle` if it is about to be deactivated.

Your control definition function should return 0 as the function result.

Setting a Control's Background Color or Pattern

If your control definition function supports embedding and draws its own background, it should return `kControlHasSpecialBackground` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set and an embedding hierarchy of controls is being drawn in your control, the Control Manager passes `kControlMsgSetUpBackground` in the `message` parameter of your control definition function. The Control Manager passes a pointer to a filled-in control background structure in the `param` parameter. Your control definition function should respond by setting its background color or pattern to whatever is appropriate given the bit depth and device type passed in. Your control definition function should return 0 as the function result.

11/18/98 © Apple Computer, Inc.

The control background structure is a structure of type `ControlBackgroundRec`:

```
struct ControlBackgroundRec {
    SInt16  depth;
    Boolean colorDevice;
};
typedef struct ControlBackgroundRec ControlBackgroundRec;
typedef ControlBackgroundRec *ControlBackgroundPtr;
```

Field descriptions

`depth`	A signed 16-bit integer indicating the bit depth (in pixels) of the current graphics port.
`colorDevice`	A Boolean value. If `true`, you are drawing on a color device. If `false`, you are drawing on a monochrome device.

Supporting Live Feedback

If your control definition function supports live feedback while tracking the indicator, it should return `kControlSupportsLiveFeedback` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set, the Control Manager will call your control definition function when it tracks the indicator and pass `kControlMsgCalcValueFromPos` in the `message` parameter. The Control Manager passes a handle to the indicator region being dragged in the `param` parameter.

Your control definition function should respond by calculating its value and drawing the control based on the new indicator region passed in. Your control definition function should not recalculate its indicator position. After the user is done dragging the indicator, your control definition function will be called with a `posCntl` message at which time you can recalculate the position of the indicator. Not recalculating the indicator position each time your control definition function is called creates a smooth dragging experience for the user.

Your control definition function should return 0 as the function result.

Being Informed When Subcontrols Are Added or Removed

If your control definition function wishes to be informed when subcontrols are added or removed, it should return `kControlSupportsEmbedding` as one of the feature bits in response to the `kControlMsgGetFeatures` message. If this bit is set, the Control Manager passes `ControlMsgSubControlAdded` in the `message` parameter immediately after a subcontrol is added, or it passes

11/18/98 © Apple Computer, Inc.

kControlMsgSubControlRemoved just before a subcontrol is removed from your embedder control. A handle to the control being added or removed from the embedding hierarchy is passed in the param parameter. Your control definition function should respond appropriately and return 0 as the function result.

Typically, a control definition function only supports this message if it wants to do extra processing in response to changes in its embedded controls. Radio groups use these messages to perform necessary processing for handling embedded controls. For example, if a currently selected radio button is deleted, the group can adjust itself accordingly.

Defining Your Own Action Functions

When your action function is called for a control part, your action function is passed a handle to the control and the control's part code. Your action function should then respond as is appropriate. For an example of such an action function, see MyActionProc (page 78). The only exception to this is for indicators that don't support live feedback.

If the mouse-down event occurs in an indicator of a control that does not support live feedback, your action function should take no parameters, because the user may move the cursor outside the indicator while dragging it. For an example of such an action function, see MyIndicatorActionProc (page 80).

The following Control Manager functions for defining your own control action functions are new, changed, or not recommended with Appearance Manager 1.0:

- MyActionProc (page 78) defines actions to be performed repeatedly in response to a mouse-down event in a control part. Changed with Appearance Manager 1.0.
- MyIndicatorActionProc (page 80) defines actions to be performed while the user holds down the mouse button when the cursor is over a control's indicator part. Not recommended with Appearance Manager 1.0.

MyActionProc

Defines actions to be performed repeatedly in response to a mouse-down event in a control part.

11/18/98 © Apple Computer, Inc.

The Control Manager declares the type for an application-defined action function as follows:

```
typedef pascal void (*ControlActionProcPtr)(
                    ControlHandle theControl,
                    ControlPartCode partCode);
```

The Control Manager defines the data type `ControlActionUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlActionUPP;
```

You typically use the `NewControlActionProc` macro like this:

```
ControlActionUPP myActionUPP;
myActionUPP = NewControlActionProc(MyAction);
```

You typically use the `CallControlActionProc` macro like this:

```
CallControlActionProc(MyActionUPP, theControl, partCode);
```

Here's how to declare an action function for a control part if you were to name the function `MyActionProc`:

```
pascal void MyActionProc (
                    ControlHandle theControl,
                    ControlPartCode partCode);
```

`theControl` A handle to the control in which the mouse-down event occurred.

`partCode` A control part code; see "Control Part Code Constants" (page 135). When the cursor is still in the control part where the mouse-down event first occurred, this parameter contains that control's part code. When the user drags the cursor outside the original control part, this parameter contains 0.

DISCUSSION

When a mouse-down event occurs in a control, `HandleControlClick` (page 38) and `TrackControl` (page 41) respond as is appropriate by highlighting the control or dragging the indicator as long as the user holds down the mouse button. You can define other actions to be performed repeatedly during this interval. To do so, define your own action function and point to it in the

11/18/98 © Apple Computer, Inc.

actionProc parameter of the TrackControl function or the inAction parameter of HandleControlClick. This is the only way to specify actions in response to all mouse-down events in a control or indicator.

IMPORTANT

You should use the MyIndicatorActionProc function while tracking indicators of controls that don't support live feedback.

VERSION NOTES

Changed with Appearance Manager 1.0 to support live feedback.

SEE ALSO

SetControlAction (page 48).

MyIndicatorActionProc

Defines actions to be performed while the user holds down the mouse button when the cursor is over a control's indicator.

When the Appearance Manager is available, you should use MyActionProc (page 78) to define actions to be performed in response to a mouse-down event in an indicator of a control that supports live feedback. You should only use MyIndicatorActionProc if the control does not support live feedback.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

Defining Your Own Key Filter Function

The following Control Manager function for defining your own key filter function is new with Appearance Manager 1.0:

- MyControlKeyFilterProc (page 81) allows for the interception and possible changing of keystrokes destined for a control. New with Appearance Manager 1.0.

11/18/98 © Apple Computer, Inc.

MyControlKeyFilterProc

The key filter function allows for the interception and possible changing of keystrokes destined for a control.

Controls that support text input (such as editable text and list box controls) can attach a key filter function to filter key strokes and modify them on return.

The Control Manager declares the type for an application-defined key filter function as follows:

```
typedef pascal KeyFilterResult (*ControlKeyFilterProcPtr)(
                    ControlHandle theControl,
                    SInt16* keyCode,
                    SInt16* charCode,
                    SInt16* modifiers);
```

The Control Manager defines the data type `ControlKeyFilterUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlKeyFilterUPP;
```

You typically use the `NewControlKeyFilterProc` macro like this:

```
ControlKeyFilterUPP myControlKeyFilterUPP;
myControlKeyFilterUPP = NewControlKeyFilterProc(MyKeyFilter);
```

You typically use the `CallControlKeyFilterProc` macro like this:

```
CallControlKeyFilterProc(myControlKeyFilterUPP, theControl, keyCode,
charCode, modifiers);
```

Here's how to declare a key filter function if you were to name the function `MyControlKeyFilterProc`:

```
pascal ControlKeyFilterResult MyControlKeyFilterProc (
                    ControlHandle theControl,
                    SInt16* keyCode,
                    SInt16* charCode,
                    SInt16* modifiers);
```

`theControl` A handle to the control in which the mouse-down event occurred.

11/18/98 © Apple Computer, Inc.

`keyCode`	The virtual key code derived from the event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.
`charCode`	A particular character derived from the event structure. This value depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource.
`modifiers`	The constant in the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.
function result	Returns a value indicating whether or not it allowed or blocked keystrokes; see "Key Filter Result Codes" (page 82).

DISCUSSION

Your key filter function can intercept and change keystrokes destined for a control. Your key filter function can change the keystroke, leave it alone, or block your control definition function from receiving it. For example, an editable text control can use a key filter function to allow only numeric values to be input in its field.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Key Filter Result Codes

Your key filter function returns these constants to specify whether or not a keystroke is filtered or blocked.

```
enum {
    kControlKeyFilterBlockKey   = 0,
    kControlKeyFilterPassKey    = 1
};
typedef SInt16 ControlKeyFilterResult;
```

Constant descriptions

`kControlKeyFilterBlockKey`
The keystroke is blocked and not received by the control.

11/18/98 © Apple Computer, Inc.

`kControlKeyFilterPassKey`
The keystroke is filtered and received by the control.

Defining Your Own User Pane Functions

This section describes the application-defined user pane functions that provide you with the ability to create a custom Appearance-compliant control without writing your own control definition function. A **user pane** is a general purpose stub control; it can be used as the root control for a window, as well as providing a way to hook in application-defined functions such as those described below. When Appearance is available, user panes should be used in dialog boxes instead of user items.

Once you have provided a user pane application-defined function, pass the tag constant representing the user pane function you wish to get or set in the `tagName` parameter of `SetControlData` (page 49). For a description of the tag constants, see "Control Data Tag Constants" (page 118). For example, to set a user pane draw function, pass the constant `kControlUserPaneDrawProcTag` of type `ControlUserPaneDrawingUPP` in the `tagName` parameter of `SetControlData` (page 49). The Control Manager then draws the control using a universal procedure pointer to your user pane draw function.

The following Control Manager functions for defining your own user pane functions are new with Appearance Manager 1.0:

- `MyUserPaneDrawProc` (page 84) draws the content of your user pane control in the rectangle of user pane control. New with Appearance Manager 1.0.
- `MyUserPaneHitTestProc` (page 85) returns the part code of the control that the point was in when the mouse-down event occurred. New with Appearance Manager 1.0.
- `MyUserPaneTrackingProc` (page 86) tracks a control while the user holds down the mouse button. New with Appearance Manager 1.0.
- `MyUserPaneIdleProc` (page 88) performs idle processing. New with Appearance Manager 1.0.
- `MyUserPaneKeyDownProc` (page 88) handles keyboard event processing. New with Appearance Manager 1.0.
- `MyUserPaneActivateProc` (page 90) handles activate and deactivate event processing. New with Appearance Manager 1.0.

11/18/98 © Apple Computer, Inc.

- `MyUserPaneFocusProc` (page 91) handles keyboard focus. New with Appearance Manager 1.0.
- `MyUserPaneBackgroundProc` (page 93) sets the background color or pattern for user panes that support embedding. New with Appearance Manager 1.0.

MyUserPaneDrawProc

Draws the content of your user pane control in the rectangle of user pane control.

The Control Manager declares the type for an application-defined user pane draw function as follows:

```
typedef pascal void (*ControlUserPaneDrawProc)(
                    ControlHandle control,
                    SInt16 part);
```

The Control Manager defines the data type `ControlUserPaneDrawUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneDrawUPP;
```

You typically use the `NewControlUserPaneDrawProc` macro like this:

```
ControlUserPaneDrawUPP myControlUserPaneDrawUPP;
myControlUserPaneDrawUPP = NewControlUserPaneDrawProc(MyUserPaneDraw);
```

You typically use the `CallControlUserPaneDrawProc` macro like this:

```
CallControlUserPaneDrawProc(myControlUserPaneDrawUPP, control, part);
```

Here's how to declare the function `MyUserPaneDrawProc`:

```
pascal void MyUserPaneDrawProc (
                    ControlHandle control,
                    SInt16 part);
```

`control` A handle to the user pane control in which you wish drawing to occur.

`part` The part code of the control you should draw. If 0, draw the entire control.

11/18/98 © Apple Computer, Inc.

DISCUSSION

Once you have created the function `MyUserPaneDrawProc`, pass `kControlUserPaneDrawProcTag` in the `tagName` parameter of `SetControlData` (page 49). The Control Manager will draw the user pane control with a universal procedure pointer to `MyUserPaneDrawProc`.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

MyUserPaneHitTestProc

Returns the part code of the control that the point was in when the mouse-down event occurred.

The Control Manager declares the type for an application-defined user pane hit test function as follows:

```
typedef pascal ControlPartCode (*ControlUserPaneHitTestProc) (
                    ControlHandle control,
                    Point where);
```

The Control Manager defines the data type `ControlUserPaneHitTestUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneHitTestUPP;
```

You typically use the `NewControlUserPaneHitTestProc` macro like this:

```
ControlUserPaneHitTestUPP myControlUserPaneHitTestUPP;
myControlUserPaneHitTestUPP = NewControlUserPaneHitTestProc
(MyUserPaneHitTest);
```

You typically use the `CallControlUserPaneHitTestProc` macro like this:

```
CallControlUserPaneHitTestProc(myControlUserPaneHitTestUPP, control,
where);
```

Here's how to declare the function `MyUserPaneHitTestProc`:

```
pascal ControlPartCode MyUserPaneHitTestProc (
                    ControlHandle control,
                    Point where);
```


11/18/98 © Apple Computer, Inc.

`control`	A handle to the control in which the mouse-down event occurred.
`where`	The point, in a window's local coordinates, where the mouse-down event occurred.
function result	Returns the part code of the control where the mouse-down event occurred. If the point was not over a control, your function should return `kControlNoPart`.

DISCUSSION

Once you have created the function `MyUserPaneHitTestProc`, pass `kControlUserPaneHitTestProcTag` in the `tagName` parameter of `SetControlData` (page 49).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

MyUserPaneTrackingProc

Tracks a control while the user holds down the mouse button.

The Control Manager declares the type for an application-defined user pane tracking function as follows:

```
typedef pascal ControlPartCode (*ControlUserPaneTrackingProc)(
                    ControlHandle control,
                    Point startPt,
                    ControlActionUPP actionProc);
```

The Control Manager defines the data type `ControlUserPaneTrackingUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneTrackingUPP;
```

You typically use the `NewControlUserPaneTrackingingProc` macro like this:

```
ControlUserPaneTrackingUPP myControlUserPaneTrackingUPP;
myControlUserPaneTrackingUPP = NewControlUserPaneTrackingProc
(MyUserPaneTracking);
```


11/18/98 © Apple Computer, Inc.

You typically use the `CallControlUserPaneTrackingingProc` macro like this:

```
CallControlUserPaneTrackingProc(myControlUserPaneTrackingUPP, control,
startPt, actionProc);
```

Here's how to declare the function `MyUserPaneTrackingProc`:

```
pascal ControlPartCode MyUserPaneTrackingProc (
                    ControlHandle control,
                    Point startPt,
                    ControlActionUPP actionProc);
```

`control` A handle to the control in which the mouse-down event occurred.

`startPt` The location of the cursor at the time the mouse button was first pressed, in local coordinates. Your application retrieves this point from the `where` field of the event structure.

`actionProc` A pointer to an action function defining what action your application takes while the user holds down the mouse button. The value of the `actionProc` parameter can be a valid `procPtr`, `nil`, or -1. A value of -1 indicates that the control should either perform auto tracking, or if it is incapable of doing so, do nothing (like `nil`).

function result Returns the part code of the control part that was tracked. If tracking was unsuccessful, `kControlNoPartCode` is returned.

DISCUSSION

Your `MyUserPaneTrackingProc` function should track the control by repeatedly calling the action function specified in the `actionProc` parameter until the mouse button is released. When the mouse button is released, your function should return the part code of the control part that was tracked.

This function will only get called if you've set the `kControlHandlesTracking` feature bit on creation of the user pane control. Once you have created the function `MyUserPaneTrackingProc`, pass `kControlUserPaneTrackingProcTag` in the `tagName` parameter of `SetControlData` (page 49).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

11/18/98 © Apple Computer, Inc.

MyUserPaneIdleProc

Performs idle processing.

The Control Manager declares the type for an application-defined user pane idle function as follows:

```
typedef pascal void (*ControlUserPaneIdleProc)(ControlHandle control);
```

The Control Manager defines the data type `ControlUserPaneIdleUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneIdleUPP;
```

You typically use the `NewControlUserPaneIdleProc` macro like this:

```
ControlUserPaneIdleUPP myControlUserPaneIdleUPP;
myControlUserPaneIdleUPP = NewControlUserPaneIdleProc(MyUserPaneIdle);
```

You typically use the `CallControlUserPaneIdleProc` macro like this:

```
CallControlUserPaneIdleProc(myControlUserPaneIdleUPP, control);
```

Here's how to declare the function `MyUserPaneIdleProc`:

```
pascal void MyUserPaneIdleProc (ControlHandle control);
```

`control` A handle to the control for which you wish to perform idle processing.

DISCUSSION

This function will only get called if you've set the `kControlWantsIdle` feature bit on creation of the user pane control. Once you have created the function `MyUserPaneIdleProc`, pass `kControlUserPaneIdleProcTag` in the `tagName` parameter of `SetControlData` (page 49).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

MyUserPaneKeyDownProc

Handles keyboard event processing.

11/18/98 © Apple Computer, Inc.

The Control Manager declares the type for an application-defined user pane key down function as follows:

```
typedef pascalControlPartCode(*ControlUserPaneKeyDownProc)(
                    ControlHandle control
                    SInt16 keyCode,
                    SInt16 charCode,
                    SInt16 modifiers);
```

The Control Manager defines the data type `UserPaneKeyDownUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneKeyDownUPP;
```

You typically use the `NewControlUserPaneKeyDownProc` macro like this:

```
ControlUserPaneKeyDownUPP myControlUserPaneKeyDownUPP;
myControlUserPaneKeyDownUPP = NewControlUserPaneKeyDownProc
(MyUserPaneKeyDown);
```

You typically use the `CallControlUserPaneKeyDownProc` macro like this:

```
CallControlUserPaneKeyDownProc(myControlUserPaneKeyDownUPP, control,
keyCode, charCode, modifiers);
```

Here's how to declare the function `MyUserPaneKeyDownProc`:

```
pascal ControlPartCode MyUserPaneKeyDownProc (
                    ControlHandle control,
                    SInt16 keyCode,
                    SInt16 charCode,
                    SInt16 modifiers);
```

`control` A handle to the control in which the keyboard event occurred.

`keyCode` The virtual key code derived from event structure. This value represents the key pressed or released by the user. It is always the same for a specific physical key on a particular keyboard regardless of which modifier keys were also pressed.

`charCode` A particular character derived from the event structure. This value depends on the virtual key code, the state of the modifier keys, and the current `'KCHR'` resource.

11/18/98 © Apple Computer, Inc.

`modifiers`	The constant in the `modifiers` field of the event structure specifying the state of the modifier keys and the mouse button at the time the event was posted.
function result	Returns the part code of the control where the keyboard event occurred. If the keyboard event did not occur in a control, your function should return `kControlNoPart`.

DISCUSSION

Your `MyUserPaneKeyDownProc` function should handle the key pressed or released by the user and return the part code of the control where the keyboard event occurred. This function will only get called if you've set the `kControlSupportsFocus` feature bit on creation of the user pane control. Once you have created the function `MyUserPaneKeyDownProc`, pass `kControlUserPaneKeyDownProcTag` in the `tagName` parameter of `SetControlData` (page 49).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

MyUserPaneActivateProc

Handles activate and deactivate event processing.

The Control Manager declares the type for an application-defined user pane activate function as follows:

```
typedef pascal void (*ControlUserPaneActivateProc)(
                    ControlHandle control,
                    Boolean activating);
```

The Control Manager defines the data type `UserPaneActivateUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneActivateUPP;
```

You typically use the `NewControlUserPaneActivateProc` macro like this:

11/18/98 © Apple Computer, Inc.

```
ControlUserPaneActivateUPP myControlUserPaneActivateUPP;
myControlUserPaneActivateUPP = NewControlUserPaneActivateProc
(MyUserPaneActivate);
```

You typically use the `CallControlUserPaneActivateProc` macro like this:

```
CallControlUserPaneActivateProc(myControlUserPaneActivateUPP, control,
activating);
```

Here's how to declare the function `MyUserPaneActivateProc`:

```
pascal void MyUserPaneActivateProc (
                    ControlHandle control
                    Boolean activating);
```

`control` A handle to the control in which the activate event occurred.

`activating` A Boolean value indicating whether or not the control is being activated. If `true`, the control is being activated. If `false`, the control is being deactivated.

DISCUSSION

Your `MyUserPaneActivateProc` function should perform any special processing before the user pane becomes activated or deactivated. For example, it should deactivate its `TEHandle` or `ListHandle` if the user pane is about to be deactivated.

This function will only get called if you've set the `kControlWantsActivate` feature bit on creation of the user pane control. Once you have created the function `MyUserPaneActivateProc`, pass `kControlUserPaneActivateProcTag` in the `tagName` parameter of `SetControlData` (page 49).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

MyUserPaneFocusProc

Handles keyboard focus.

11/18/98 © Apple Computer, Inc.

The Control Manager declares the type for an application-defined user pane focus function as follows:

```
typedef pascal ControlPartCode (*ControlUserPaneFocusProc)(
                    ControlHandle control,
                    ControlFocusPart action);
```

The Control Manager defines the data type `ControlUserPaneFocusUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneFocusUPP;
```

You typically use the `NewControlUserPaneFocusProc` macro like this:

```
ControlUserPaneFocusUPP myControlUserPaneFocusUPP;
myControlUserPaneFocusUPP = NewControlUserPaneFocusProc
(MyUsePaneFocus);
```

You typically use the `CallControlUserPaneFocusProc` macro like this:

```
CallControlUserPaneFocusProc(myControlUserPaneFocusUPP, control, action);
```

Here's how to declare the function `MyUserPaneFocusProc`:

```
pascal ControlPartCode MyUserPaneFocusProc (
                    ControlHandle control
                    ControlFocusPart action);
```

`control` A handle to the control that is to adjust its focus.

`action` The part code of the user pane to receive keyboard focus; see "Handling Keyboard Focus" (page 72).

function result Returns the part of the user pane actually focused. `kControlFocusNoPart` is returned if the user pane has lost the focus or cannot be focused.

DISCUSSION

Your `MyUserPaneFocusProc` function is called in response to a change in keyboard focus. It should respond by changing keyboard focus based on the part code passed in the `action` parameter.

11/18/98 © Apple Computer, Inc.

This function will only get called if you've set the `kControlSupportsFocus` feature bit on creation of the user pane control. Once you have created the function `MyUserPaneFocusProc`, pass `kControlUserPaneFocusProcTag` in the `tagName` parameter of `SetControlData` (page 49).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

MyUserPaneBackgroundProc

Sets the background color or pattern for user panes that support embedding.

The Control Manager declares the type for an application-defined user pane background color function as follows:

```
typedef pascal (*ControlUserPaneBackgroundProcPtr)(
                    ControlHandle control,
                    ControlBackgroundPtr info);
```

The Control Manager defines the data type `ControlUserPaneBackgroundUPP` to identify the universal procedure pointer for this application-defined function:

```
typedef UniversalProcPtr ControlUserPaneBackgroundUPP;
```

You typically use the `NewControlUserPaneBackgroundProc` macro like this:

```
ControlUserPaneBackgroundUPP myControlUserPaneBackgroundUPP;
myControlUserPaneBackgroundUPP = NewControlUserPaneBackgroundProc
(MyUsePaneBackground);
```

You typically use the `CallControlUserPaneBackgroundProc` macro like this:

```
CallControlUserPaneBackgroundProc(myControlUserPaneBackgroundUPP,
control, info);
```


11/18/98 © Apple Computer, Inc.

Here's how to declare the function `MyUserPaneBackgroundProc`:

```
pascal void MyUserPaneBackgroundProc (
                    ControlHandle control
                    ControlBackgroundPtr info);
```

`control`	A handle to the control for which the background color or pattern is to be set.
`info`	A pointer to information such as the depth and type of the drawing device.

DISCUSSION

Your `MyUserPaneBackgroundProc` function should set the user pane background color or pattern to whatever is appropriate given the bit depth and device type passed in. Your `MyUserPaneBackgroundProc` function is called to set up the background color. This ensures that when an embedded control calls `EraseRgn` or `EraseRect`, the background is erased to the correct color or pattern.

This function will only get called if there is a control embedded in the user pane and if you've set the `kControlHasSpecialBackground` and `kControlSupportsEmbedding` feature bits on creation of the user pane control. Once you have created the function `MyUserPaneBackgroundProc`, pass `kControlUserPaneBackgroundProcTag` in the `tagName` parameter of `SetControlData` (page 49).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Control Manager Data Types

The following Control Manager data types are new, changed, or not recommended with Appearance Manager 1.0:

- `ControlFontStyleRec` (page 95)
- `ControlButtonContentInfo` (page 97)
- `ControlEditTextSelectionRec` (page 98)

11/18/98 © Apple Computer, Inc.

- ControlTabInfoRec (page 99)
- AuxCtlRec (page 99)
- PopupPrivateData (page 99)
- CtlCTab (page 100)
- 'CNTL' (page 100)
- 'cctb' (page 102)
- 'ldes' (page 102)
- 'tab#' (page 104)

ControlFontStyleRec

You can use the ControlFontStyleRec type to specify a control's font. You pass a pointer to the control font style structure in the inStyle parameter of SetControlFontStyle (page 53) to specify a control's font. If none of the flags in the flags field of the structure are set, the control uses the system font unless the control variant kControlUsesOwningWindowsFontVariant has been specified, in which case the control uses the window font. The ControlFontStyleRec type is available with Appearance Manager 1.0 and later.

Note that if you wish to specify the font for controls in a dialog box, you should use a dialog font table resource, which is automatically read in by the Dialog Manager.

```
struct ControlFontStyleRec {
SInt16      flags;
SInt16      font;
SInt16      size;
SInt16      style;
SInt16      mode;
SInt16      just;
RGBColor    foreColor;
RGBColor    backColor;
};
typedef struct ControlFontStyleRec ControlFontStyleRec;
typedef ControlFontStyleRec *ControlFontStylePtr;
```


11/18/98 © Apple Computer, Inc.

Field descriptions

`flags` A signed 16-bit integer specifying which fields of the structure should be applied to the control; see "Control Font Style Flag Constants" (page 126). If none of the flags in the `flags` field of the structure are set, the control uses the system font unless the control variant `kControlUsesOwningWindowsFontVariant` has been specified, in which case the control uses the window font.

`font` If the `kControlUseFontMask` bit is set, then this field contains a value specifying the ID of the font family to use. If this bit is not set, then the system default font is used. A meta font constant can be specified instead; see "Meta Font Constants" (page 138).

`size` If the `kControlUseSizeMask` bit is set, then this field contains a value specifying the point size of the text. If the `kControlAddSizeMask` bit is set, this value will represent the size to add to the current point size of the text. A meta font constant can be specified instead; see "Meta Font Constants" (page 138).

`style` If the `kControlUseFaceMask` bit is set, then this field contains a value specifying which styles to apply to the text. If all bits are clear, the plain font style is used. The bit numbers and the styles they represent are

Bit value	Style
0	Bold
1	Italic
2	Underline
3	Outline
4	Shadow
5	Condensed
6	Extended

`mode` If the `kControlUseModeMask` bit is set, then this field contains a value specifying how characters are drawn in the bit image. See *Inside Macintosh: Imaging With QuickDraw* for a discussion of transfer modes.

`just` If the `kControlUseJustMask` bit is set, then this field contains a value specifying text justification. Possible values are `teFlushDefault` (0), `teCenter` (1), `teFlushRight` (-1), and `teFlushLeft` (-2).

11/18/98 © Apple Computer, Inc.

foreColor	If the kControlUseForeColorMask bit is set, then this field contains an RGB color value to use when drawing the text.
backColor	If the kControlUseBackColorMask bit is set, then this field contains an RGB color value to use when drawing the background behind the text. In certain text modes, background color is ignored.

ControlButtonContentInfo

You can use the ControlButtonContentInfo structure to specify the content for a bevel button or image well. Values of type ControlButtonContentInfo are set via SetControlData (page 49) and obtained from GetControlData (page 50), in conjunction with the kControlBevelButtonContentTag and kControlImageWellContentTag constants; see "Control Data Tag Constants" (page 118). The ControlButtonContentInfo type is available with Appearance Manager 1.0 and later.

```
struct ControlButtonContentInfo {
    ControlContentType contentType;
    union {
        SInt16      resID;
        CIconHandle cIconHandle;
        Handle      iconSuite;
        Handle      iconRef;
        PicHandle   picture;
    } u;
};
typedef struct ControlButtonContentInfo ControlButtonContentInfo;
typedef ControlButtonContentInfo *ControlButtonContentInfoPtr;
```

Field descriptions

contentType	Specifies the bevel button or image well content type and whether the content is text-only, resource-based, or handle-based; see "Bevel Button and Image Well Content Type Constants" (page 130). The value specified in the contentType field determines which of the other fields in the structure are used.
resID	If the content type specified in the contentType field is kControlContentIconSuiteRes, kControlContentCIconRes, or

11/18/98 © Apple Computer, Inc.

	kControlContentPictRes, this field contains the resource ID of a picture, color icon, or icon suite resource.
cIconHandle	If the content type specified in the contentType field is kControlContentCIconHandle, this field contains a handle to a color icon.
iconSuite	If the content type specified in the contentType field is kControlContentIconSuiteHandle, this field contains a handle to an icon suite.
iconRef	If the content type specified in the contentType field is kControlContentIconRef, this field contains an IconRef value. IconRef values are supported under Mac OS 8.5 and later.
picture	If the content type specified in the contentType field is kControlContentPictHandle, this field contains a handle to a picture.

ControlEditTextSelectionRec

You can use the ControlEditTextSelectionRec type to specify a selection range in an editable text control. You pass a pointer to the editable text selection structure to GetControlData (page 50) and SetControlData (page 49) to access and set the current selection range in an editable text control. The ControlEditTextSelectionRec type is available with Appearance Manager 1.0 and later.

```
struct ControlEditTextSelectionRec {
    SInt16  selStart;
    SInt16  selEnd;
};
typedef struct ControlEditTextSelectionRec ControlEditTextSelectionRec;
typedef ControlEditTextSelectionRec *ControlEditTextSelectionPtr;
```

Field descriptions

selStart	A signed 16-bit integer indicating the beginning of the editable text selection.
selEnd	A signed 16-bit integer indicating the end of the editable text selection.

11/18/98 © Apple Computer, Inc.

ControlTabInfoRec

You can use the `ControlTabInfoRec` type to specify the icon and title for a tab control. If you are not creating a tab control with a `'tab#'` resource, you can call `SetControlMaximum` to set the number of tabs in a tab control. Then use the functions `SetControlData` (page 49) and `GetControlData` (page 50) with the `ControlTabInfoRec` structure to access information for an individual tab in a tab control. The `ControlTabInfoRec` type is available with Appearance Manager 1.0.1 and later.

```
struct ControlTabInfoRec {
    SInt16  version;
    SInt16  iconSuiteID;
    Str255  name;
};
```

Field descriptions

`version`	A signed 16-bit integer indicating the version of the tab information structure. The only currently available version value is 0.
`iconSuiteID`	A signed 16-bit integer indicating the ID of an icon suite to use for the tab label. If the specified ID is not found, no icon is displayed for the tab label. Pass 0 for no icon.
`name`	A string specifying the title to be used for the tab label.

AuxCtlRec

The auxiliary control structure is not recommend with the Appearance Manager. When the Appearance Manager is available and you are using standard controls, most of the fields of the auxiliary control structure are ignored except the `acCTable` and `acFlags` fields. If you are creating your own control definition function, the entire auxiliary control structure can be used.

PopupPrivateData

The pop-up menu private structure is not recommend with the Appearance Manager. When the Appearance Manager is available, you should not access the pop-up menu private data structure. Instead, you should pass the value `kControlBevelButtonMenuHandleTag` in the `tagName` parameter of `GetControlData`

11/18/98 © Apple Computer, Inc.

(page 50) to get the menu handle of a bevel button, and the menu handle and the menu ID of the menu associated with a pop-up menu.

CtlCTab

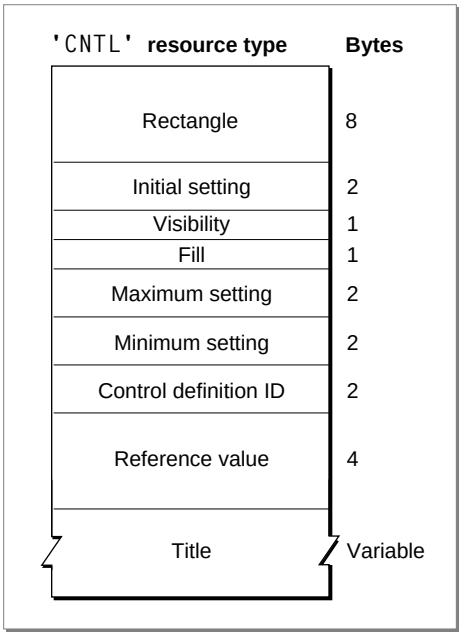
The control color table structure is not recommend with the Appearance Manager. When the Appearance Manager is available and you are using standard controls, the control color table structure is ignored and the colors are determined by the current theme. If you are creating your own control definition function, you can use the control color table structure to draw a control using colors other than the system default.

'CNTL'

The control resource is changed with the Appearance Manager to support the additional standard controls that are available with the Appearance Manager. You can use a control (`'CNTL'`) resource to define a standard control. All control resources must have resource ID numbers greater than 127. Use `GetNewControl` (page 12) to create a control defined in a control resource. The Control Manager uses the information you specify to create a control structure in memory. Figure 1-1 shows the structure of this resource.

11/18/98 © Apple Computer, Inc.

Figure 1-1 Structure of a compiled control ('CNTL') resource



The compiled version of a control resource contains the following elements:

- The rectangle, specified in coordinates local to the window, that encloses the control and thus determines its size and location.
- The initial setting for the control; see "Settings Values for Standard Controls" (page 113).
- The visibility of the control. If this element contains the value `true`, `GetNewControl` draws the control immediately, without using the application's standard updating mechanism for windows. If this element contains the value `false`, the application must use `ShowControl` (page 27) when it's prepared to display the control.
- Fill. Set to 0.
- The maximum setting for the control; see "Settings Values for Standard Controls" (page 113).

11/18/98 © Apple Computer, Inc.

- The minimum setting for the control; see “Settings Values for Standard Controls” (page 113).
- The control definition ID, which the Control Manager uses to determine the control definition function for this control; see “Control Definition IDs” (page 106).
- The control’s reference value, which is set and used only by the application—except when the application adds the `kControlPopupUseAddResMenuVariant` variation code to the `kControlPopupButtonProc` control definition ID.
- For controls that need a title, the string for that title; for controls that don’t use titles, an empty string.

Note
The titles of all Appearance-compliant standard system controls appear in the system font. You should generally use the system font or small system font in your controls; see *Mac OS 8 Human Interface Guidelines* for more details.

'cctb'

The control color table resource is not recommend with the Appearance Manager. When the Appearance Manager is available and you are using standard controls, the control color table (`'cctb'`) resource is ignored and the colors are determined by the current theme. If you are creating your own control definition function, you can still use the control color table structure to draw a control using colors other than the system default.

'ldes'

You can use a list box description resource to specify information in a list box. A list box description resource is a resource of type `'ldes'`. All list box description resources must have resource ID numbers greater than 127. The Control Manager uses the information you specify to provide additional information to the corresponding list box control. The list box description resource is available with Appearance Manager 1.0 and later.

Figure 1-2 shows the structure of this resource.

11/18/98 © Apple Computer, Inc.

Figure 1-2 Structure of a compiled list box description (`'ldes'`) resource

`'ldes'` resource type	Bytes
Version number	2
Number of rows	2
Number of columns	2
Cell height	2
Cell width	2
Has vertical scroll	1
Reserved	1
Has horizontal scroll	1
Reserved	1
List definition resource ID	2
Has size box	1
Reserved	1

You define a list box description resource by specifying these elements:

- Version number. An integer specifying the version of the resource format.
- Number of rows. An integer specifying the number of rows in the list box.
- Number of columns. An integer specifying the number of columns in the list box.
- Cell height. An integer specifying the height of a list item. If 0 is specified, the list item height is automatically calculated.
- Cell width. An integer specifying the width of a list item. If 0 is specified, the list item width is automatically calculated.
- Has vertical scroll bar. A Boolean value that indicates whether the list box should contain a vertical scroll bar. If `true`, the list box contains a vertical scroll bar; if `false`, no vertical scroll bar.
- Reserved. Set to 0.
- Has horizontal scroll bar. A Boolean value that indicates whether the list should contain a horizontal scroll bar. Specify `true` if your list requires a horizontal scroll bar; specify `false` otherwise.

11/18/98 © Apple Computer, Inc.

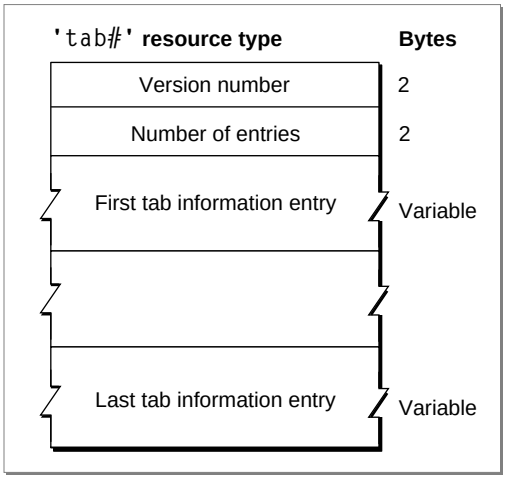
- Reserved. Set to 0.
- Resource ID. This is the resource ID of the list definition procedure to use for the list. To use the default list definition procedure, which supports the display of unstyled text, specify a resource ID of 0.
- Has size box. A Boolean value that indicates whether the List Manager should leave room for a size box. If `true`, a size box will be drawn; if `false`, a size box will not be drawn.
- Reserved. Set to 0.

'tab#'

You can use a tab information resource to specify the icon suite ID and name of each tab in a tab control. A tab information resource is a resource of type `'tab#'`. All tab information resources must have resource ID numbers greater than 127. The Control Manager uses the information you specify to provide additional information to the corresponding tab control. The tab information resource is available with Appearance Manager 1.0 and later.

Figure 1-3 shows the structure of this resource.

Figure 1-3 Structure of a compiled tab information (`'tab#'`) resource



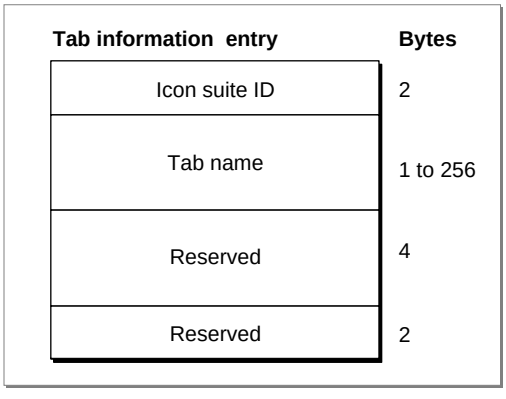
11/18/98 © Apple Computer, Inc.

A compiled version of a tab information resource contains the following elements:

- Version number. An integer specifying the version of the resource.
- An integer that specifies the number of entries in the resource (that is, the number of tab information structures).
- A series of tab information structures, each of which consists of a 2-byte icon suite identifier and a variable-length string indicating the tab name.

Figure 1-4 shows the format of a compiled entry in a `'tab#'` resource. A tab information entry specifies the icon suite ID and the name of a tab control.

Figure 1-4 Structure of a tab information entry



Each entry in a `'tab#'` resource contains the following:

- Icon suite ID. A value of 0 indicates no icon.
- Tab name. The title of the tab control.
- Reserved. Set to 0.
- Reserved. Set to 0.

11/18/98 © Apple Computer, Inc.

Control Manager Constants

The following Control Manager constants are new, changed, or not recommended with Appearance Manager 1.0:

Control Definition IDs

When creating a control, your application supplies a control definition ID to one of the Control Manager control-creation functions or to the control resource; see `'CNTL'` (page 100). The control definition ID indicates the type of control to create. A **control definition ID** is an integer that contains the resource ID of a

11/18/98 © Apple Computer, Inc.

control definition function in its upper 12 bits and a variation code in its lower 4 bits. A control definition ID is derived as follows:

control definition ID = 16 * (`'CDEF'` resource ID) + variation code

A **control definition function** determines how a control generally looks and behaves. Control definition functions are stored as resources of type `'CDEF'`. Various Control Manager functions call a control definition function whenever they need to perform some control-dependent action, such as drawing the control on the screen. For more information on how to create a control definition function, see “Defining Your Own Control Definition Function” (page 56).

A control definition function, in turn, can use a **variation code** to describe variations of the same basic control. For example, all pop-up arrows share the same basic control definition function, which is stored in a resource of type `'CDEF'` and has a resource ID of 12. The standard pop-up arrow is large and points to the right; it has a control definition ID of 192. A variation of this is a large, left-pointing arrow, which has a control definition ID of 193. Still another variation, in which the arrow points up, has a control definition ID of 194.

Your application can use the constants listed in Table 1-1 in place of control definition IDs. Most of these constants, and their associated IDs, are new with the Appearance Manager and are not supported unless the Appearance Manager is available. A control definition ID that is new is identified with an asterisk (*) in its description in Table 1-1. For illustrations of these new controls, see “Control Guidelines” in *Mac OS 8 Human Interface Guidelines*.

If your application contains code that uses the older, pre-Appearance control definition IDs or their constants, your application can use the Appearance Manager to map the old IDs to those for the new, updated controls introduced by the Appearance Manager. In particular, the control definition IDs for

11/18/98 © Apple Computer, Inc.

pre-Appearance checkboxes, buttons, scroll bars, radio buttons, and pop-up menus will be automatically mapped to Appearance-compliant equivalents.

Table 1-1 Control definition IDs and resource IDs for standard controls

Constant (and Value) for Control Definition ID	Description	Resource ID
`pushButProc` (0)	Pre-Appearance push button.	0
`pushButProc + kControlUsesOwningWindowsFontVariant` (8)	Pre-Appearance push button with its text in the window font.	0
`kControlPushButtonProc` (368)	Appearance-compliant push button.*	23
`kControlPushButLeftIconProc` (374)	Appearance-compliant push button with a color icon to the left of the control title.* (This direction is reversed when the system justification is right to left). The `contrlMax` field of the control structure for this control contains the resource ID of the `'cicn'` resource drawn in the pushbutton.	23
`kControlPushButRightIconProc` (375)	Appearance-compliant push button with a color icon to right of control title.* (This direction is reversed when the system justification is right to left). The `contrlMax` field of the control structure for this control contains the resource ID of the `'cicn'` resource drawn in the pushbutton.	23
`checkBoxProc` (1)	Pre-Appearance checkbox.	0
`checkBoxProc + kControlUsesOwningWindowsFontVariant` (8)	Pre-Appearance checkbox with a control title in the window font.	0
`kControlCheckBoxProc` (369)	Appearance-compliant checkbox.*	23
`radioButProc` (2)	Pre-Appearance radio button.	0
`radioButProc + kControlUsesOwningWindowsFontVariant` (8)	Pre-Appearance radio button with a title in the window font.	0
`kControlRadioButtonProc` (370)	Appearance-compliant radio button.*	23
`scrollBarProc` (16)	Pre-Appearance scroll bar.	1

11/18/98 © Apple Computer, Inc.

Table 1-1 Control definition IDs and resource IDs for standard controls (continued)

Constant (and Value) for Control Definition ID	Description	Resource ID
kControlScrollBarProc (384)	Appearance-compliant scroll bar.*	24
kControlScrollBarLiveProc (386)	Appearance-compliant scroll bar with live feedback.*	24
kControlBevelButtonSmallBevelProc (32)	Bevel button with a small bevel.*	2
kControlBevelButtonNormalBevelProc (33)	Bevel button with a normal bevel.*	2
kControlBevelButtonLargeBevelProc (34)	Bevel button with a large bevel.*	2
kControlBevelButtonSmallBevelProc + kControlBevelButtonMenuOnRight (4)	Small bevel button with a pop-up menu.*	2
kControlSliderProc (48)	Slider.* Your application calls the function SetControlAction (page 48) to set the last value for the control.	3
kControlSliderProc + kControlSliderLiveFeedback (1)	Slider with live feedback.* The value of the control is updated automatically by the Control Manager before your action function is called. If no application-defined action function is supplied, the slider draws an outline of the indicator as the user moves it.	3
kControlSliderProc + kControlSliderHasTickMarks (2)	Slider with tick marks.* The control rectangle must be large enough to include the tick marks.	3
kControlSliderProc + kControlSliderReverseDirection (4)	Slider with a directional indicator.* The indicator is positioned perpendicularly to the slider; that is, if the slider is horizontal, the indicator points up, and if the slider is vertical, the indicator points left.	3
kControlSliderProc + kControlSliderNonDirectional (8)	Slider with a rectangular, non-directional indicator.* This variant overrides the kSliderReverseDirection and kSliderHasTickMarks variants.	3
kControlTriangleProc (64)	Disclosure triangle.*	4
kControlTriangleLeftFacingProc (65)	Left-facing disclosure triangle.*	4

11/18/98 © Apple Computer, Inc.

Table 1-1 Control definition IDs and resource IDs for standard controls (continued)

Constant (and Value) for Control Definition ID	Description	Resource ID
kControlTriangleAutoToggleProc (66)	Auto-tracking disclosure triangle.*	4
kControlTriangleLeftFacingAutoToggleProc (67)	Left-facing, auto-tracking disclosure triangle.*	4
kControlProgressBarProc (80)	Progress indicator.* To make the control determinate or indeterminate, set the kControlProgressBarIndeterminateTag constant; see "Control Data Tag Constants" (page 118). Progress indicators are only horizontal in orientation; vertical progress indicators are not currently supported.	5
kControlLittleArrowsProc (96)	Little arrows.*	6
kControlChasingArrowsProc (11)	Asynchronous arrows.*	7
kControlTabLargeProc (128)	Normal tab control.*	8
kControlTabSmallProc (129)	Small tab control.*	
kControlSeparatorLineProc (144)	Separator line.	9
kControlGroupBoxTextTitleProc (160)	Primary group box with text title.*	10
kControlGroupBoxCheckBoxProc (161)	Primary group box with checkbox title.*	10
kControlGroupBoxPopupButtonProc (162)	Primary group box with pop-up button title.*	10
kControlGroupBoxSecondaryTextTitleProc (164)	Secondary group box with text title.*	10
kControlGroupBoxSecondaryCheckBoxProc (165)	Secondary group box with checkbox title.*	10
kControlGroupBoxSecondaryPopupButtonProc (166)	Secondary group box with pop-up button title.*	10

11/18/98 © Apple Computer, Inc.

Table 1-1 Control definition IDs and resource IDs for standard controls (continued)

Constant (and Value) for Control Definition ID	Description	Resource ID
`kControlImageWellProc` (176)	Image well.* This control behaves as a palette-type object: it can be selected by clicking, and clicking on another object should change the keyboard focus. If the keyboard focus is removed, your application should then set the value to 0 to remove the checked border.	11
`kControlImageWellAutoTrackProc` (177)	Image well with autotracking.* This variant sets the value itself so the control remains highlighted.	11
`kControlPopupArrowEastProc` (192)	Large, right-facing pop-up arrow.*	12
`kControlPopupArrowWestProc` (193)	Large, left-facing pop-up arrow.*	12
`kControlPopupArrowNorthProc` (194)	Large, up-facing pop-up arrow.*	12
`kControlPopupArrowSouthProc` (195)	Large, down-facing pop-up arrow.*	12
`kControlPopupArrowSmallEastProc` (196)	Small, right-facing pop-up arrow.*	12
`kControlPopupArrowSmallWestProc` (197)	Small, left-facing pop-up arrow.*	12
`kControlPopupArrowSmallNorthProc` (198)	Small, up-facing pop-up arrow.*	12
`kControlPopupArrowSmallSouthProc` (199)	Small, down-facing pop-up arrow.*	12
`kControlPlacardProc` (224)	Placard.*	14
`kControlClockTimeProc` (240)	Clock control displaying hour / minutes.*	15
`kControlClockTimeSecondsProc` (241)	Clock control displaying hours / minutes / seconds.*	15
`kControlClockDateProc` (242)	Clock control displaying date / month / year.*	15
`kControlClockMonthYearProc` (243)	Clock control displaying month / year.*	15
`kControlUserPaneProc` (256)	User pane.*	16
`kControlEditTextProc` (272)	Editable text field for windows.* This control maintains its own text handle (`TEHandle`).	17

11/18/98 © Apple Computer, Inc.

Table 1-1 Control definition IDs and resource IDs for standard controls (continued)

Constant (and Value) for Control Definition ID	Description	Resource ID
`kControlEditTextPasswordProc` (274)	Editable text field for passwords.* This control is supported by the Script Manager. Password text can be accessed via the `kEditTextPasswordTag` constant; see “Control Data Tag Constants” (page 118).	17
`kControlStaticTextProc` (288)	Static text field.*	18
`kControlPictureProc` (304)	Picture control.*	19
`kControlPictureNoTrackProc` (305)	Non-tracking picture.* Immediately returns `kControlPicturePart` as the part code hit without tracking.	19
`kControlIconProc` (320)	Icon control.*	20
`kControlIconNoTrackProc` (321)	Non-tracking icon.*	20
`kControlIconSuiteProc` (322)	Icon suite.*	20
`kControlIconSuiteNoTrackProc` (323)	Non-tracking icon suite.*	20
`kControlWindowHeaderProc` (336)	Window header.*	21
`kControlWindowListViewHeaderProc` (337)	Window list view header.*	21
`kControlListBoxProc` (352)	List box.*	21
`kControlListBoxAutoSizeProc` (353)	Autosizing list box.*	21
`popupMenuProc` (1008)	Pre-Appearance standard pop-up menu.	63
`popupMenuProc + popupFixedWidth` (1009)	Pre-Appearance, fixed-width pop-up menu.	63
`popupMenuProc + popupVariableWidth` (1010)	Pre-Appearance, variable-width pop-up menu.	63
`popupMenuProc + popupUseAddResMenu` (1012)	Pre-Appearance pop-up menu with a value of type `ResType` in the `contrlRfCon` field of the control structure. The Menu Manager adds resources of this type to the menu.	63

11/18/98 © Apple Computer, Inc.

Table 1-1 Control definition IDs and resource IDs for standard controls (continued)

Constant (and Value) for Control Definition ID	Description	Resource ID
`popupMenuProc + popupUseWFont` (1016)	Pre-Appearance pop-up menu with a control title in the window font.	63
`kControlPopupButtonProc` (400)	Appearance-compliant standard pop-up menu.*	25
`kControlPopupButtonProc + kControlPopupFixedWidthVariant` (1)	Appearance-compliant fixed-width pop-up menu.*	25
`kControlPopupButtonProc + kControlPopupVariableWidthVariant` (2)	Appearance-compliant variable-width pop-up menu.*	25
`kControlPopupButtonProc + kControlPopupUseAddResMenuVariant` (4)	Appearance-compliant pop-up menu with a value of type `ResType` in the `contrlRfCon` field of the control structure.* The Menu Manager adds resources of this type to the menu.	25
`kControlPopupButtonProc + kControlPopupUseWFontVariant` (8)	Appearance-compliant pop-up menu with control title in window font.*	25
`kControlRadioGroupProc` (416)	Radio group.* Embedder control for controls that have set the feature bit `kControlHasRadioBehavior`.	26

* This control definition is new with the Appearance Manager and is not supported unless the Appearance Manager is available.

Settings Values for Standard Controls

This section lists the initial, minimum, and maximum settings for all standard controls. You can use these values in the control resource when creating a new control from a resource or with the function `NewControl` (page 13). Note that some controls specify other information besides their range in their minimum and maximum settings. For example, bevel buttons use the high byte of their minimum value to indicate their behavior.

Control Values

Push button (pre-Appearance)

Initial: 0

11/18/98 © Apple Computer, Inc.

Minimum: 0
Maximum: 1

Push button (Appearance-compliant)

Initial: 0
Minimum: 0
Maximum: 1

Checkbox (pre-Appearance)

Initial: `kControlCheckboxUncheckedValue`
Minimum: `kControlCheckboxUncheckedValue`
Maximum: `kControlCheckboxCheckedValue`

Checkbox (Appearance-compliant)

Initial: `kControlCheckboxUncheckedValue`
Minimum: `kControlCheckboxUncheckedValue`
Maximum: `kControlCheckboxCheckedValue` or `kControlCheckboxMixedValue`

Radio button (pre-Appearance)

Initial: `kControlRadioButtonUncheckedValue`
Minimum: `kControlRadioButtonUncheckedValue`
Maximum: `kControlRadioButtonCheckedValue`

Radio button (Appearance-compliant)

Initial: `kControlRadioButtonUncheckedValue`
Minimum: `kControlRadioButtonUncheckedValue`
Maximum: `kControlRadioButtonCheckedValue` or `kControlRadioButtonMixedValue`

Scroll bar (pre-Appearance and Appearance-compliant versions)

Initial: Appropriate value between –32768 and 32768.
Minimum: –32768 to 32768
Maximum: –32768 to 32768; when the maximum setting is equal to the minimum setting, the scroll bar is inactive.

Bevel button

Initial: If you wish to attach a resource-based menu, the menu's resource ID. If you wish to attach a non-resource-based menu, you must pass in a non-zero initial value, then call the `SetControlData` function with the `kControlBevelButtonMenuHandleTag` control data tag constant and the return value from a call to the `NewMenu` function. If no menu is to be attached, 0.
Minimum: High byte specifies behavior; see "Bevel Button Behavior Constants" (page 128) and "Bevel Button Menu Constants" (page 129). Low byte specifies content type; see

11/18/98 © Apple Computer, Inc.

	"Bevel Button and Image Well Content Type Constants" (page 130). **Maximum:** Resource ID of bevel button's content if resource-based; see "Bevel Button and Image Well Content Type Constants" (page 130).
Slider	**Initial:** Appropriate value between –32768 and 32768; for tick mark variant, the number of ticks. The control definition function resets this value to the minimum setting once the slider is created. **Minimum:** –32768 to 32768 **Maximum:** –32768 to 32768; when the maximum setting is equal to the minimum setting, the slider is inactive.
Disclosure triangle	**Initial:** 0 (collapsed) or 1 (expanded) **Minimum:** 0 (collapsed) **Maximum:** 1 (expanded)
Progress indicator	**Initial:** Appropriate value between –32768 and 32768. **Minimum:** –32768 to 32768 **Maximum:** –32768 to 32768
Little arrows	**Initial:** Appropriate value between –32768 and 32768. **Minimum:** –32768 to 32768 **Maximum:** –32768 to 32768
Asynchronous arrows	**Initial:** Reserved. Set to 0. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Tab control	**Initial:** Resource ID of the `'tab#'` resource you are using to hold tab information. The control definition function resets this value to the minimum setting once the tab control is created. Under Appearance 1.0.1 and later, a value of 0 indicates not to read a `'tab#'` resource; see `ControlTabInfoRec` (page 99). **Minimum:** Ignored. The control definition function resets this value to 1 once the tab control is created. **Maximum:** Under Appearance 1.0, the maximum value is ignored. Under Appearance 1.0.1, the maximum value specifies the number of tabs in the tab control.
Separator line	**Initial:** Reserved. Set to 0. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.

11/18/98 © Apple Computer, Inc.

Primary group box and secondary group box	**Initial:** Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same value as the checkbox or pop-up button. **Minimum:** Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same minimum setting as the checkbox or pop-up button. **Maximum:** Ignored if group box has text title. If the group box has a checkbox or pop-up button title, same maximum setting as the checkbox or pop-up button.
Image well	**Initial:** Resource ID of the image well's content, if the content type specified in the minimum value is resource-based. The control definition function resets this value to 0 once the image well is created. **Minimum:** Low byte specifies content type; see "Bevel Button and Image Well Content Type Constants" (page 130). The control definition function resets this value to 0 once the image well is created. **Maximum:** Ignored. The control definition function resets this value to 2 once the image well is created.
Pop-up arrow	**Initial:** Reserved. Set to 0. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Placard	**Initial:** Reserved. Set to 0. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Clock	**Initial:** One or more of the clock value flags; see "Clock Value Flag Constants" (page 135). The control definition function resets this value to 0 once the clock is created. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
User pane	**Initial:** One or more of the control feature constants; see "Specifying Which Appearance-Compliant Messages Are Supported" (page 68). The control definition function resets this value to 0 once the user pane is created. **Minimum:** Ignored. The control definition function resets this value to a setting between –32768 to 32768 once the user pane is created. **Maximum:** Ignored. The control definition function resets

11/18/98 © Apple Computer, Inc.

	this value to a setting between –32768 to 32768 once the user pane is created.
Editable text field	**Initial:** Reserved. Set to 0. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Static text field	**Initial:** Reserved. Set to 0. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Picture	**Initial:** Resource ID of the `'pict'` resource you wish to display. The control definition function resets this value to 0 once the picture control is created. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Icon	**Initial:** Resource ID of the `'cicn'`, `'ICON'`, or icon suite resource you wish to display. For icon suite variant, it only looks for an icon suite. If not, it looks for a `'cicn'` or `'ICON'` resource. The control definition function resets this value to 0 once the icon control is created. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Window header	**Initial:** Reserved. Set to 0. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
List box	**Initial:** Resource ID of the `'ldes'` resource you are using to hold list box information. The control definition function resets this value to 0 once the list box is created. An initial value of 0 indicates not to read an `'ldes'` resource under Appearance 1.0.1 and later. **Minimum:** Reserved. Set to 0. **Maximum:** Reserved. Set to 0.
Pop-up menu (pre-Appearance and Appearance-compliant versions)	
	Initial: One or more of the pop-up menu title constants. **Minimum:** Resource ID of the `'MENU'` resource. **Maximum:** Width (in pixels) of the pop-up menu title.
Radio group	**Initial:** Set to 0 on creation. The control definition function resets this value to the index of the currently selected embedded radio button control once the radio group is created. If currently selected control does not support radio behavior, value will be set to 0 and the control will be

11/18/98 © Apple Computer, Inc.

deselected. To deselect all controls, set to 0.
Minimum: Set to 0.
Maximum: Set to 0 on creation. The control definition function resets this value to the number of embedded controls as controls are added.

Control Data Tag Constants

You can use the control data tag constants to set or obtain data that is associated with a control. The control data tag constants are passed in the `inTagName` parameters of `SetControlData` (page 49) and `GetControlData` (page 50) to specify the piece of data in a control that you wish to set or get. You can also pass these constants in the `inTagName` parameter of `GetControlDataSize` (page 52) if you wish to determine the size of variable-length control data (e.g., text in an editable text control). These constants can also be used by custom control definition functions that return the feature bit `kControlSupportsDataAccess` in response to a `kControlMsgGetFeatures` message. The control data tag constants are available with Appearance Manager 1.0 and later.

The data that your application sets or obtains can be of various types, dependent upon the control. Therefore, the descriptions of the control data tag constants list the data types for the information that you can set in the `inData` parameter to the `SetControlData` function and that you can get in the `inBuffer` parameter to the `GetControlData` function.

```
enum {
    kControlPushButtonDefaultTag            = ('dflt'),
    kControlBevelButtonContentTag           = ('cont'),
    kControlBevelButtonTransformTag         = ('tran'),
    kControlBevelButtonTextAlignTag         = ('tali'),
    kControlBevelButtonTextOffsetTag        = ('toff'),
    kControlBevelButtonGraphicAlignTag      = ('gali'),
    kControlBevelButtonGraphicOffsetTag     = ('goff'),
    kControlBevelButtonTextPlaceTag         = ('tplc'),
    kControlBevelButtonMenuValueTag         = ('mval'),
    kControlBevelButtonMenuHandleTag        = ('mhnd'),
    kControlBevelButtonCenterPopupGlyphTag  = ('pglc'),
    kControlTriangleLastValueTag            = ('last'),
    kControlProgressBarIndeterminateTag     = ('inde'),
    kControlTabContentRectTag               = ('rect'),
    kControlTabEnabledFlagTag               = ('enab'),
```


11/18/98 © Apple Computer, Inc.

```
    kControlTabInfoTag                      = ('tabi'),
    kControlGroupBoxMenuHandleTag           = ('mhan'),
    kControlImageWellContentTag             = ('cont'),
    kControlImageWellTransformTag           = ('tran'),
    kControlClockLongDateTag                = ('date'),
    kControlUserItemDrawProcTag             = ('uidp'),
    kControlUserPaneDrawProcTag             = ('draw'),
    kControlUserPaneHitTestProcTag          = ('hitt'),
    kControlUserPaneTrackingProcTag         = ('trak'),
    kControlUserPaneIdleProcTag             = ('idle'),
    kControlUserPaneKeyDownProcTag          = ('keyd'),
    kControlUserPaneActivateProcTag         = ('acti'),
    kControlUserPaneFocusProcTag            = ('foci'),
    kControlUserPaneBackgroundProcTag       = ('back'),
    kControlEditTextTextTag                 = ('text'),
    kControlEditTextTEHandleTag             = ('than'),
    kControlEditTextSelectionTag            = ('sele'),
    kControlEditTextPasswordTag             = ('pass'),
    kControlStaticTextTextTag               = ('text'),
    kControlStaticTextTextHeightTag         = ('thei'),
    kControlIconTransformTag                = ('trfm'),
    kControlIconAlignmentTag                = ('algn'),
    kControlListBoxListHandleTag            = ('lhan'),
    kControlFontStyleTag                    = ('font'),
    kControlKeyFilterTag                    = ('fltr'),
    kControlBevelButtonLastMenuTag          = ('lmnu'),
    kControlBevelButtonMenuDelayTag         = ('mdly'),
    kControlPopupButtonMenuHandleTag        = ('mhan'),
    kControlPopupButtonMenuIDTag            = ('mnid'),
    kControlListBoxDoubleClickTag           = ('dblc'),
    kControlListBoxLDEFTag                  = ('ldef')
};
```

Constant descriptions

`kControlPushButtonDefaultTag`

Tells Appearance-compliant button whether to draw a default ring, or returns whether the Appearance Manager draws a default ring for the button.

Data type returned or set: `Boolean`

11/18/98 © Apple Computer, Inc.

`kControlBevelButtonContentTag`

Gets or sets a bevel button's content type for drawing; see "Bevel Button and Image Well Content Type Constants" (page 130).

Data type returned or set: `ControlButtonContentInfo` structure

`kControlBevelButtonTransformTag`

Gets or sets a transform that is added to the standard transform of a bevel button; see "Icon Utilities" in *More Macintosh Toolbox*.

Data type returned or set: `IconTransformType`

`kControlBevelButtonTextAlignTag`

Gets or sets the alignment of text in a bevel button; see "Bevel Button Text Alignment Constants" (page 133).

Data type returned or set: `ControlButtonTextAlignment`

`kControlBevelButtonTextOffsetTag`

Gets or sets the number of pixels that text is offset in a bevel button from the button's left or right edge; this is used with left, right, or system justification, but it is ignored when the text is center aligned.

Data type returned or set: `SInt16`

`kControlBevelButtonGraphicAlignTag`

Gets or sets the alignment of graphics in a bevel button in relation to any text the button may contain; see "Bevel Button Graphic Alignment Constants" (page 132).

Data type returned or set: `ControlButtonGraphicAlignment`

`kControlBevelButtonGraphicOffsetTag`

Gets or sets the horizontal and vertical amounts that a graphic element contained in a bevel button is offset from the button's edges; this value is ignored when the graphic is specified to be center aligned on the button. Note that offset values should not be used for bevel buttons with content of type `kControlContentIconRef`, because `IconRef` based icons may change with a theme switch; see "Bevel Button and Image Well Content Type Constants" (page 130).

Data type returned or set: `point`

`kControlBevelButtonTextPlaceTag`

Gets or sets the placement of a bevel button's text; see

11/18/98 © Apple Computer, Inc.

"Bevel Button Text Placement Constants" (page 134).
Data type returned or set: `ControlButtonTextPlacement`

`kControlBevelButtonMenuValueTag`
Gets the menu value for a bevel button with an attached menu; see "Bevel Button Menu Constants" (page 129).
Data type returned: `SInt16`

`kControlBevelButtonMenuHandleTag`
Gets or sets the menu handle for a bevel button with an attached menu. To set a non-resource-based menu for a bevel button, you must pass in a non-zero value in the `initialValue` parameter of the `NewControl` function, then call the `SetControlData` function with the `kControlBevelButtonMenuHandleTag` constant and the return value from a call to the `NewMenu` function.
Data type returned: `MenuHandle`

`kControlBevelButtonCenterPopUpGlyphTag`
Gets or sets the position of the pop-up arrow in a bevel button when a pop-up menu is attached.
Data type returned or set: `Boolean`; if `true`, glyph is vertically centered on the right; if `false`, glyph is on the bottom right.

`kControlTriangleLastValueTag`
Gets or sets the last value of a disclosure triangle. Used primarily for setting up a disclosure triangle properly when using the auto-toggle variant.
Data type returned or set: `SInt16`

`kControlProgressBarIndeterminateTag`
Gets or sets whether a progress indicator is determinate or indeterminate.
Data type returned or set: `Boolean`; if `true`, switches to an indeterminate progress indicator; if `false`, switches to an determinate progress indicator.

`kControlTabContentRectTag`
Gets the content rectangle of a tab control.
Data type returned: `Rect`

`kControlTabEnabledFlagTag`
Enables or disables a single tab in a tab control.
Data type returned or set: `Boolean`; if `true`, enabled; if `false`, disabled.

11/18/98 © Apple Computer, Inc.

`kControlTabInfoTag`

Gets or sets information for a tab in a tab control; see `ControlTabInfoRec` (page 99). Available with Appearance 1.0.1 and later.
Data type returned or set: `ControlTabInfoRec`.

`kControlGroupBoxMenuHandleTag`

Gets the menu handle of a group box.
Data type returned: `MenuHandle`

`kControlImageWellContentTag`

Gets or sets the content for an image well; see `ControlButtonContentInfo` (page 97).
Data type returned or set: `ControlButtonContentInfo` structure

`kControlImageWellTransformTag`

Gets or sets a transform that is added to the standard transform of an image well; see "Icon Utilities" in *More Macintosh Toolbox*.
Data type returned or set: `IconTransformType`

`kControlClockLongDateTag`

Gets or sets the clock control's time or date.
Data type returned or set: `LongDateRec` structure

`kControlUserItemDrawProcTag`

Gets or sets an application-defined item drawing function. If an embedding hierarchy is established, a user pane drawing function should be used instead of an item drawing function.
Data type returned or set: `UserItemUPP`

`kControlUserPaneDrawProcTag`

Gets or sets a user pane drawing function; see `MyUserPaneDrawProc` (page 84). Indicates that the Control Manager needs to draw a control.
Data type returned or set: `ControlUserPaneDrawingUPP`

`kControlUserPaneHitTestProcTag`

Gets or sets a user pane hit-testing function. Indicates that the Control Manager needs to determine if a control part was hit; see `MyUserPaneHitTestProc` (page 85).
Data type returned or set: `ControlUserPaneHitTestUPP`

`kControlUserPaneTrackingProcTag`

Gets or sets a user pane tracking function, which will be

11/18/98 © Apple Computer, Inc.

called when a control definition function returns the `kControlHandlesTracking` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane handles its own tracking; see `MyUserPaneTrackingProc` (page 86).
Data type returned or set: `ControlUserPaneTrackingUPP`

`kControlUserPaneIdleProcTag`
Gets or sets a user pane idle function, which will be called when a control definition function returns the `kControlWantsIdle` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane performs idle processing; see `MyUserPaneIdleProc` (page 88).
Data type returned or set: `ControlUserPaneIdleUPP`

`kControlUserPaneKeyDownProcTag`
Gets or sets a user pane key down function, which will be called when a control definition function returns the `kControlSupportsFocus` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane performs keyboard event processing; see `MyUserPaneKeyDownProc` (page 88).
Data type returned or set: `ControlUserPaneKeyDownUPP`

`kControlUserPaneActivateProcTag`
Gets or sets a user pane activate function, which will be called when a control definition function returns the `kControlWantsActivate` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane wants to be informed of activate and deactivate events; see `MyUserPaneActivateProc` (page 90).
Data type returned or set: `ControlUserPaneActivateUPP`

`kControlUserPaneFocusProcTag`
Gets or sets a user pane keyboard focus function, which will be called when a control definition function returns the `kControlSupportsFocus` feature bit in response to a `kControlMsgGetFeatures` message. Indicates that a user pane handles keyboard focus; see `MyUserPaneFocusProc` (page 91).
Data type returned or set: `ControlUserPaneFocusUPP`

`kControlUserPaneBackgroundProcTag`
Gets or sets a user pane background function, which will be called when a control definition function returns the `kControlHasSpecialBackground` and

11/18/98 © Apple Computer, Inc.

kControlSupportsEmbedding feature bits in response to a kControlMsgGetFeatures message. Indicates that a user pane can set its background color or pattern; see MyUserPaneBackgroundProc (page 93).
Data type returned or set: ControlUserPaneBackgroundUPP

kControlEditTextTextTag
Gets or sets text in an editable text control.
Data type returned or set: character buffer

kControlEditTextTEHandleTag
Gets a handle to a text edit structure.
Data type returned: TEHandle

kControlEditTextSelectionTag
Gets or sets the selection in an editable text control.
Data type returned or set: ControlEditTextSelectionRec structure

kControlEditTextPasswordTag
Gets clear password text from an editable text control, that is, the text of the actual password typed, not the bullet text.
Data type returned: character buffer

kControlStaticTextTextTag
Gets or sets text in a static text control.
Data type returned or set: character buffer

kControlStaticTextTextHeightTag
Gets the height of text in a static text control.
Data type returned or set: SInt16

kControlIconTransformTag
Gets or sets a transform that is added to the standard transform of an icon; see "Icon Utilities" in *More Macintosh Toolbox*.
Data type returned or set: IconTransformType

kControlIconAlignmentTag
Gets or sets an icon's position (centered, left, right); see "Icon Utilities" in *More Macintosh Toolbox*.
Data type returned or set: IconAlignmentType

kControlListBoxListHandleTag
Gets a handle to a list box.
Data type returned: ListHandle

11/18/98 © Apple Computer, Inc.

`kControlFontStyleTag`

Gets or sets the font style for controls that support text (includes list box, tab, clock, static and editable text).
Data type returned or set: `kControlFontStyleTag`

`kControlKeyFilterTag`

Gets or sets the key filter function for controls that handle filtered input (includes editable text and list box).
Data type returned or set: `ControlKeyFilterUPP`

`kControlBevelButtonLastMenuTag`

Gets the menu ID of the last menu selected in the submenu or main menu. Available with Appearance 1.0.1 and later.
Data type returned: `SInt16`

`kControlBevelButtonMenuDelayTag`

Gets or sets the delay (in number of ticks) before the menu is displayed. Available with Appearance 1.0.1 and later.
Data type returned or set: `SInt32`

`kControlPopupButtonMenuHandleTag`

Gets or sets the menu handle for a pop-up menu. Available with Appearance 1.0.1 and later.
Data type returned or set: `MenuHandle`

`kControlPopupButtonMenuIDTag`

Gets or sets the menu ID for a pop-up menu. Available with Appearance 1.0.1 and later.
Data type returned or set: `SInt16`

`kControlListBoxDoubleClickTag`

Checks to see whether the most recent click in a list box was a double click. Available with Appearance 1.0.1 and later.
Data type returned: `Boolean`; if `true`, the last click was a double click; if `false`, not.

`kControlListBoxLDEFTag`

Sets the `'LDEF'` resource to be used to draw a list box's contents; this is useful for creating a list box without an `'ldes'` resource. Available with Appearance 1.0.1 and later.
Data type set: `SInt16`

11/18/98 © Apple Computer, Inc.

Control Font Style Flag Constants

You can pass one or more control font style flag constants in the `flags` field of the control font style structure to specify the field(s) of the structure that should be applied to the control; see `ControlFontStyleRec` (page 95). If none of the flags are set, the control uses the system font unless a control variant specifies use of a window font. The control font style flag constants are available with Appearance Manager 1.0 and later.

```
enum {
    kControlUseFontMask         = 0x0001,
    kControlUseFaceMask         = 0x0002,
    kControlUseSizeMask         = 0x0004,
    kControlUseForeColorMask    = 0x0008,
    kControlUseBackColorMask    = 0x0010,
    kControlUseModeMask         = 0x0020,
    kControlUseJustMask         = 0x0040,
    kControlUseAllMask          = 0x00FF,
    kControlAddFontSizeMask     = 0x0100
};
```

Constant descriptions

`kControlUseFontMask`

If the `kControlUseFontMask` flag is set (bit 0), the `font` field of the control font style structure is applied to the control.

`kControlUseFaceMask`

If the `kControlUseFaceMask` flag is set (bit 1), the `style` field of the control font style structure is applied to the control. This flag is ignored if you specify a meta font value; see “Meta Font Constants” (page 138).

`kControlUseSizeMask`

If the `kControlUseSizeMask` flag is set (bit 2), the `size` field of the control font style structure is applied to the control. This flag is ignored if you specify a meta font value; see “Meta Font Constants” (page 138).

`kControlUseForeColorMask`

If the `kControlUseForeColorMask` flag is set (bit 3), the `foreColor` field of the control font style structure is applied to the control. This flag only applies to static text controls.

11/18/98 © Apple Computer, Inc.

`kControlUseBackColorMask`
: If the `kControlUseBackColorMask` flag is set (bit 4), the `backColor` field of the control font style structure is applied to the control. This flag only applies to static text controls.

`kControlUseModeMask`
: If the `kControlUseModeMask` flag is set (bit 5), the text mode specified in the `mode` field of the control font style structure is applied to the control.

`kControlUseJustMask`
: If the `kControlUseJustMask` flag is set (bit 6), the `just` field of the control font style structure is applied to the control.

`kControlUseAllMask`
: If `kControlUseAllMask` is used, all flags in this mask will be set except `kControlUseAddFontSizeMask`.

`kControlUseAddFontSizeMask`
: If the `kControlUseAddFontSizeMask` flag is set (bit 8), the Dialog Manager will add a specified font size to the `size` field of the control font style structure. This flag is ignored if you specify a meta font value; see "Meta Font Constants" (page 138).

Checkbox Value Constants

The checkbox value constants specify the value of a standard checkbox control and are passed in the `newValue` parameter of `SetControlValue` and are returned by `GetControlValue`. The checkbox value constants are changed with Appearance Manager 1.0 to support mixed-value checkboxes.

```
enum {
    kControlCheckboxUncheckedValue  = 0,
    kControlCheckboxCheckedValue    = 1,
    kControlCheckboxMixedValue      = 2
};
```

Constant descriptions

`kControlCheckboxUncheckedValue`
: The checkbox is unchecked.

`kControlCheckboxCheckedValue`
: The checkbox is checked.

11/18/98 © Apple Computer, Inc.

`kControlCheckboxMixedValue`
: Mixed value. Indicates that a setting is on for some elements in a selection and off for others. This state only applies to standard Appearance-compliant checkboxes.

Radio Button Value Constants

These constants specify the value of a standard radio button control and are passed in the `newValue` parameter of `SetControlValue` and are returned by `GetControlValue`. The radio button value constants are changed with Appearance Manager 1.0 to support mixed-value radio buttons.

```
enum {
    kControlRadioButtonUncheckedValue   = 0,
    kControlRadioButtonCheckedValue     = 1,
    kControlRadioButtonMixedValue       = 2
};
```

Constant descriptions

`kControlRadioButtonUncheckedValue`
: The radio button is unselected.

`kControlRadioButtonCheckedValue`
: The radio button is selected.

`kControlRadioButtonMixedValue`
: Mixed value. Indicates that a setting is on for some elements in a selection and off for others. This state only applies to standard Appearance-compliant radio buttons.

Bevel Button Behavior Constants

You can pass the bevel button behavior constants in the high byte of the `minimumValue` parameter of `NewControl` (page 13) to create a bevel button with a specific behavior. The bevel button behavior constants are available with Appearance Manager 1.0 and later.

```
enum {
    kControlBehaviorPushbutton      = 0,
    kControlBehaviorToggles         = 0x0100,
```


11/18/98 © Apple Computer, Inc.

```
    kControlBehaviorSticky          = 0x0200,
    kControlBehaviorOffsetContents  = 0x8000
};
```

Constant descriptions

`kControlBehaviorPushbutton`
: Push button (momentary) behavior. The bevel button pops up after being clicked.

`kControlBehaviorToggles`
: Toggle behavior. The bevel button toggles state automatically when clicked.

`kControlBehaviorSticky`
: Sticky behavior. Once clicked, the bevel button stays down until your application sets the control's value to 0. This behavior is useful in tool palettes and radio groups.

`kControlBehaviorOffsetContents`
: Bevel button contents are offset (one pixel down and to the right) when button is pressed.

Bevel Button Menu Constants

You can pass one or more bevel button menu constants in the high byte of the `minimumValue` parameter of `NewControl` (page 13) to create a bevel button with a menu of a certain behavior. Bevel buttons with menus have two values: the value of the button and the value of the menu. You can specify the direction of the pop-up menu arrow (down or right) by using the `kControlBevelButtonMenuOnRight` bevel button variant. The bevel button menu constants are available with Appearance Manager 1.0 and later.

```
enum{
    kControlBehaviorCommandMenu     = 0x2000,
    kControlBehaviorMultiValueMenu  = 0x4000
};
```

Constant descriptions

`kControlBehaviorCommandMenu`
: If this bit is set, the menu contains commands, not choices, and should not be marked with a checkmark. If this bit is set, it overrides the `kControlBehaviorMultiValueMenu` bit.

11/18/98 © Apple Computer, Inc.

This constant is only available with Appearance 1.0.1 and later.

`kControlBehaviorMultiValueMenu`

If this bit is set, the menus are multi-valued. The bevel button does not maintain the menu value as it normally would (requiring that only one item is selected at a time). This allows the user to toggle entries in a menu and have multiple items checked. In this mode, the menu value accessed with the `kControlMenuLastValueTag` will return the value of the last menu item selected.

Bevel Button and Image Well Content Type Constants

You can use constants of type `ControlContentType` in the `contentType` field of the `ControlButtonContentInfo` (page 97) structure to display various kinds of bevel button and image well content, including text, icons, and pictures. The `ControlContentType` constants are available with Appearance Manager 1.0 and later, except as noted.

The resource IDs for icon suite, color icon, and picture resources are passed in the `maximumValue` parameter of `NewControl` (page 13) or in a control resource; see `'CNTL'` (page 100). The content type is passed in the low byte of the `minimumValue` parameter of `NewControl`.

Note
Resource-based content is owned by the control, while handle-based content is owned by you. The control definition function will not dispose of handle-based content. If you replace handle-based content with resource-based content on the fly, you must dispose of the handle properly to avoid a memory leak.

```
enum {
    kControlContentTextOnly         = 0,
    kControlContentIconSuiteRes     = 1,
    kControlContentCIconRes         = 2,
    kControlContentPictRes          = 3,
    kControlContentIconSuiteHandle  = 129,
    kControlContentCIconHandle      = 130,
    kControlContentPictHandle       = 131,
```


11/18/98 © Apple Computer, Inc.

```
    kControlContentIconRef          = 132
};
typedef SInt16 ControlContentType;
```

Constant descriptions

`kControlContentTextOnly`
: Content type is text. This constant is passed in the `contentType` field of the `ControlButtonContentInfo` structure if the content is text only. The variation code `kControlUsesOwningWindowsFontVariant` applies when text content is used.

`kControlContentIconSuiteRes`
: Content type uses an icon suite resource ID. The resource ID of the icon suite resource you wish to display should be in the `resID` field of the `ControlButtonContentInfo` structure.

`kControlContentCIconRes`
: Content type is a color icon resource ID. The resource ID of the color icon resource you wish to display should be in the `resID` field of the `ControlButtonContentInfo` structure.

`kControlContentPictRes`
: Content type is a picture resource ID. The resource ID of the picture resource you wish to display should be in the `resID` field of the `ControlButtonContentInfo` structure.

`kControlContentIconSuiteHandle`
: Content type is an icon suite handle. The handle of the icon suite you wish to display should be in the `iconSuite` field of the `ControlButtonContentInfo` structure.

`kControlContentCIconHandle`
: Content type uses a color icon handle. The handle of the color icon you wish to display should be in the `cIconHandle` field of the `ControlButtonContentInfo` structure.

`kControlContentPictHandle`
: Content type uses a picture handle. The handle of the picture you wish to display should be in the `picture` field of the `ControlButtonContentInfo` structure.

`kControlContentIconRef`
: Content type is `IconRef`. An `IconRef` value for the icon you wish to display should be provided in the `iconRef` field of

11/18/98 © Apple Computer, Inc.

the `ControlButtonContentInfo` structure. Note that the `kControlBevelButtonGraphicOffsetTag` control data tag constant should not be used with `IconRef` based bevel button content, because `IconRef` based icons may change with a theme switch; see "Control Data Tag Constants" (page 118). Supported with Mac OS 8.5 and later.

Bevel Button Graphic Alignment Constants

You can use the `ControlButtonGraphicAlignment` constants to specify the alignment of icons and pictures in bevel buttons. These constants are passed in the `inData` parameter of `SetControlData` (page 49) and returned by `GetControlData` (page 50). The `ControlButtonGraphicAlignment` constants are available with Appearance Manager 1.0 and later.

```
enum {
    kControlBevelButtonAlignSysDirection    = -1,
    kControlBevelButtonAlignCenter          = 0,
    kControlBevelButtonAlignLeft            = 1,
    kControlBevelButtonAlignRight           = 2,
    kControlBevelButtonAlignTop             = 3,
    kControlBevelButtonAlignBottom          = 4,
    kControlBevelButtonAlignTopLeft         = 5,
    kControlBevelButtonAlignBottomLeft      = 6,
    kControlBevelButtonAlignTopRight        = 7,
    kControlBevelButtonAlignBottomRight     = 8
};
typedef SInt16 ControlButtonGraphicAlignment;
```

Constant descriptions

`kControlBevelButtonAlignSysDirection`
: Bevel button graphic is aligned according to the system default script direction (only left or right).

`kControlBevelButtonAlignCenter`
: Bevel button graphic is aligned center.

`kControlBevelButtonAlignLeft`
: Bevel button graphic is aligned left.

`kControlBevelButtonAlignRight`
: Bevel button graphic is aligned right.

11/18/98 © Apple Computer, Inc.

`kControlBevelButtonAlignTop`
: Bevel button graphic is aligned top.

`kControlBevelButtonAlignBottom`
: Bevel button graphic is aligned bottom.

`kControlBevelButtonAlignTopLeft`
: Bevel button graphic is aligned top left.

`kControlBevelButtonAlignBottomLeft`
: Bevel button graphic is aligned bottom left.

`kControlBevelButtonAlignTopRight`
: Bevel button graphic is aligned top right.

`kControlBevelButtonAlignBottomRight`
: Bevel button graphic is aligned bottom right.

Bevel Button Text Alignment Constants

You can use the `ControlButtonTextAlignment` constants to specify the alignment of text in a bevel button. These constants are passed in the `inData` parameter of `SetControlData` (page 49) and returned by `GetControlData` (page 50). The `ControlButtonTextAlignment` constants are available with Appearance Manager 1.0 and later.

```
enum {
    kControlBevelButtonAlignTextSysDirection     = teFlushDefault,
    kControlBevelButtonAlignTextCenter           = teCenter,
    kControlBevelButtonAlignTextFlushRight       = teFlushRight,
    kControlBevelButtonAlignTextFlushLeft        = teFlushLeft
};
typedef SInt16 ControlButtonTextAlignment;
```

Constant descriptions

`kControlBevelButtonAlignTextSysDirection`
: Bevel button text is aligned according to the current script direction (left or right).

`kControlBevelButtonAlignTextCenter`
: Bevel button text is aligned center.

`kControlBevelButtonAlignTextFlushRight`
: Bevel button text is aligned flush right.

11/18/98 © Apple Computer, Inc.

`kControlBevelButtonAlignTextFlushLeft`
Bevel button text is aligned flush left.

Bevel Button Text Placement Constants

You can use the `ControlButtonTextPlacement` constants to specify the placement of text in a bevel button, in relation to an icon or picture. These constants are passed in the `inData` parameter of `SetControlData` (page 49) and returned by `GetControlData` (page 50). They can be used in conjunction with bevel button text and graphic alignment constants to create, for example, a button where the graphic and text are left justified with the text below the graphic. The `ControlButtonTextPlacement` constants are available with Appearance Manager 1.0 and later.

```
enum {
    kControlBevelButtonPlaceSysDirection        = -1,
    kControlBevelButtonPlaceNormally            = 0,
    kControlBevelButtonPlaceToRightOfGraphic    = 1,
    kControlBevelButtonPlaceToLeftOfGraphic     = 2,
    kControlBevelButtonPlaceBelowGraphic        = 3,
    kControlBevelButtonPlaceAboveGraphic        = 4
};
typedef SInt16 ControlButtonTextPlacement;
```

Constant descriptions

`kControlBevelButtonPlaceSysDirection`
Bevel button text is placed according to the system default script direction.

`kControlBevelButtonPlaceNormally`
Bevel button text is centered.

`kControlBevelButtonPlaceToRightOfGraphic`
Bevel button text is placed to the right of the graphic.

`kControlBevelButtonPlaceToLeftOfGraphic`
Bevel button text is placed to the left of the graphic.

`kControlBevelButtonPlaceBelowGraphic`
Bevel button text is placed below the graphic.

`kControlBevelButtonPlaceAboveGraphic`
Bevel button text is placed above the graphic.

11/18/98 © Apple Computer, Inc.

Clock Value Flag Constants

You can use the clock value flag constants to specify behaviors for a clock control. You can pass one or more of these mask constants into the control ('CNTL') resource or in the `initialValue` parameter of `NewControl` (page 13). Note that the standard clock control is editable and supports keyboard focus. Also, the little arrows that allow manipulation of the date and time are part of the control, not a separate embedded little arrows control. The clock value flag constants are available with Appearance Manager 1.0 and later.

```
enum {
    kControlClockNoFlags        = 0,
    kControlClockIsDisplayOnly  = 1,
    kControlClockIsLive         = 2
};
```

Constant descriptions

`kControlClockNoFlags`
Indicates that clock is editable but does not display the current “live” time.

`kControlClockIsDisplayOnly`
When only this bit is set, the clock is not editable. When this bit and the `kControlClockIsLive` bit is set, the clock automatically updates on idle (clock will have the current time).

`kControlClockIsLive`
When only this bit is set, the clock automatically updates on idle and any changes to the clock affect the system clock. When this bit and the `kControlClockIsDisplayOnly` bit is set, the clock automatically updates on idle (clock will have the current time), but is not editable.

Control Part Code Constants

Constants of type `ControlPartCode` identify specific parts of controls for functions such as `SetControlData` (page 49), `GetControlData` (page 50), and `FindControlUnderMouse` (page 35). The `ControlPartCode` constants are changed with the Appearance Manager to support new control part codes.

11/18/98 © Apple Computer, Inc.

Part codes are meaningful only within the scope of a single control definition function. For example, the standard tab control uses part codes 1...*N*, where *N* is the number of tabs, even though those numbers do collide with part codes defined for use with other control definition functions. Therefore, when you wish to specify part codes for the tab control for use with the function `SetControlData`, for example, you should use a part code corresponding to a 1-based index of the tab whose data you wish to set. In other words, the first tab is part code 1, the second tab is part code 2, and so on.

Note that if you wish to create part codes for a custom control definition function, you may assign values anywhere within the ranges 1–128 and 130–253. Note also that the function `FindControl` does not typically return the `kControlDisabledPart` or `kControlInactivePart` part codes and never returns them with standard controls.

```
enum {
    kControlNoPart                  = 0,
    kControlLabelPart               = 1,
    kControlMenuPart                = 2,
    kControlTrianglePart            = 4,
    kControlEditTextPart            = 5,
    kControlPicturePart             = 6,
    kControlIconPart                = 7,
    kControlClockPart               = 8,
    kControlButtonPart              = 10,
    kControlCheckBoxPart            = 11,
    kControlRadioButtonPart         = 12,
    kControlUpButtonPart            = 20,
    kControlDownButtonPart          = 21,
    kControlPageUpPart              = 22,
    kControlPageDownPart            = 23,
    kControlListBoxPart             = 24,
    kControlListBoxDoubleClickPart  = 25,
    kControlImageWellPart           = 26,
    kControlRadioGroupPart          = 27,
    kControlIndicatorPart           = 129,
    kControlDisabledPart            = 254,
    kControlInactivePart            = 255
};
typedef SInt16 ControlPartCode;
```

Constant descriptions

11/18/98 © Apple Computer, Inc.

`kControlNoPart` Identifies no specific control part. This value unhighlights any highlighted part of the control when passed to the `HiliteControl` function. For events in bevel buttons with an attached menu, this part code indicates that either the mouse was released outside the bevel button and menu or that the button was disabled.

`kControlLabelPart` Identifies the label of a pop-up menu control.

`kControlMenuPart` Identifies the menu of a pop-up menu control. For bevel buttons with a menu attached, this part code specifies a menu item of the bevel button.

`kControlTrianglePart`
Identifies a disclosure triangle control.

`kControlEditTextPart`
Identifies an editable text control. Available with Appearance Manager 1.0 and later.

`kControlPicturePart`
Identifies a picture control. Available with Appearance Manager 1.0 and later.

`kControlIconPart` Identifies an icon control. Available with Appearance Manager 1.0 and later.

`kControlClockPart` Identifies a clock control. Available with Appearance Manager 1.0 and later.

`kControlButtonPart`
Identifies either a push button or bevel button control. For bevel buttons with a menu attached, this part code specifies the button but not the attached menu.

`kControlCheckBoxPart`
Identifies a checkbox control.

`kControlRadioButtonPart`
Identifies a radio button control.

`kControlUpButtonPart`
Identifies the up button of a scroll bar control (the arrow at the top or the left).

`kControlDownButtonPart`
Identifies the down button of a scroll bar control (the arrow at the right or the bottom).

`kControlPageUpPart`
Identifies the page-up part of a scroll bar control.

11/18/98 © Apple Computer, Inc.

`kControlPageDownPart`
: Identifies the page-down part of a scroll bar control.

`kControlListBoxPart`
: Identifies a list box control. Available with Appearance Manager 1.0 and later.

`kControlListBoxDoubleClickPart`
: Identifies a double-click in a list box control. Available with Appearance Manager 1.0 and later.

`kControlImageWellPart`
: Identifies an image well control. Available with Appearance Manager 1.0 and later.

`kControlRadioGroupPart`
: Identifies a radio group control. Available with Appearance Manager 1.0.2 and later.

`kControlIndicatorPart`
: Identifies the scroll box of a scroll bar control.

`kControlDisabledPart`
: Used with `HiliteControl` to disable the control.

`kControlInactivePart`
: Used with `HiliteControl` to make the control inactive.

Part Identifier Constants

The part identifier constants are not recommended with the Appearance Manager. When the Appearance Manager is available and you are using standard controls, part identifier constants are ignored and the colors are determined by the current theme. If you are creating your own control definition function, you can still use these constants in the `partIdentifier` field of a control color table structure to draw a control using colors other than the system default and to identify the part of a control that a color affects.

Meta Font Constants

You can use the meta font constants in the `font` field of the structure `ControlFontStyleRec` (page 95) and the Font ID field of a dialog font table resource to specify the style, size, and font family of the control font. You should use these meta font constants whenever possible because the system font can change, depending upon the current theme. If none of these constants are

11/18/98 © Apple Computer, Inc.

specified, the control uses the system font unless directed to use a window font by a control variant. The meta font constants are available with Appearance Manager 1.0 and later.

```
enum {
    kControlFontBigSystemFont       = -1,
    kControlFontSmallSystemFont     = -2,
    kControlFontSmallBoldSystemFont = -3
};
```

Constant descriptions

`kControlFontBigSystemFont`
Use the system font.

`kControlFontSmallSystemFont`
Use the small system font.

`kControlFontSmallBoldSystemFont`
Use the small emphasized system font (emphasis applied correctly for locale).

Control Variant Constants

You can use the control variant constants with any of the standard control resource IDs to specify additional features of a control. The control variant constants are changed with Appearance Manager 1.0 to support the additional control types available with the Appearance Manager.

```
typedef SInt16 ControlVariant;
enum {
    kControlNoVariant   = 0,
    kControlUsesOwningWindowsFontVariant = 1 << 3
};
```

Constant descriptions

`kControlNoVariant` Specifies no change to the standard control resource.

`kControlUsesOwningWindowsFontVariant`
Specifies that the control use the window font for any control text.

11/18/98 © Apple Computer, Inc.

Result Codes

The most common result codes returned by Control Manager functions are listed below.

noErr	0	No error
paramErr	–50	Error in parameter list
memFullErr	–108	Not enough memory
resNotFound	–192	Unable to read resource
hmHelpManagerNotInited	–855	Help menu not set up
errMessageNotSupported	–30580	Message not supported
errDataNotSupported	–30581	Data not supported
errControlDoesntSupportFocus	–30582	Control does not support focus
errWindowDoesntSupportFocus	–30583	Window does not support focus
errUnknownControl	–30584	Specified control not found
errCouldntSetFocus	–30585	Could not set focus
errNoRootControl	–30586	No embedding hierarchy established
errRootAlreadyExists	–30587	Root control already exists
errInvalidPartCode	–30588	Invalid part code
errControlsAlreadyExist	–30589	Control already exists
errControlIsNotEmbedder	–30590	Control is not an embedder
errDataSizeMismatch	–30591	Data size mismatch
errControlHiddenOrDisabled	–30592	Control hidden or disabled
errWindowRegionCodeInvalid	–30593	Window region code invalid
errCantEmbedIntoSelf	–30594	Can’t embed control in self
errCantEmbedRoot	–30595	Can’t embed root control
errItemNotControl	–30596	Dialog item not a control

11/18/98 © Apple Computer, Inc.

Version History

This document has had the following releases:

Table A-1 *Mac OS 8 Control Manager Reference* Revision History

Version	Notes
Nov. 18, 1998	Removed "Control Manager Reference" chapter from the *Mac OS 8 Toolbox Reference* document. *Inside Macintosh: Control Manager Reference* is now available as an independent document. The following corrections were made: `MyControlKeyFilterProc` (page 81). Corrected description of `NewControlKeyFilterProc` macro—changed `NewControlKeyFilterUPP` to `ControlKeyFilterUPP`. `MyUserPaneBackgroundProc` (page 93). Corrected the function discussion in various ways, including noting the requirements for it to be called. `ControlFontStyleRec` (page 95). Noted that the bit mask relevant to the `style` field is `kControlUseFaceMask`, not `kControlUseStyleMask`. Specified the actual values that can be used in the `just` field. `ControlButtonContentInfo` (page 97). Added description of the `iconRef` field. `ControlTabInfoRec` (page 99). Noted that no icon is displayed for the tab label if the specified resource ID is not found. Control Definition Function Resource. Recategorized from "changed with the Appearance Manager" to "unchanged" and, therefore, removed from this delta document. "Settings Values for Standard Controls" (page 113). Noted that the control definition function is responsible for resetting values for some controls after the controls are created. Discussed tab control maximum value behavior under Appearance Manager 1.0.1. Clarified mechanism for attaching a non-resource-based menu to a bevel button.

11/18/98 © Apple Computer, Inc.

Table A-1 *Mac OS 8 Control Manager Reference* Revision History

Version	Notes
Nov. 18, 1998	"Control Data Tag Constants" (page 118). Discussed use of the `kControlBevelButtonMenuHandleTag` constant with the `SetControlData` function to attach a non-resource-based menu to a bevel button. Noted that the `kControlBevelButtonGraphicOffsetTag` constant should not be used to set offsets for bevel buttons with `IconRef` based content. "Bevel Button Menu Constants" (page 129). Noted that these values must be passed in the `minimumValue` parameter of the `NewControl` function, not the `initialValue` parameter. "Bevel Button and Image Well Content Type Constants" (page 130). Added description for `kControlContentIconRef` constant. "Control Part Code Constants" (page 135). Added discussion of part code scope. Noted allowable ranges for application-defined part codes for custom control definition functions. Also noted that the `kControlRadioGroupPart` constant is available with Appearance Manager 1.0.2 and later, not Appearance Manager 1.0.1 and later. "Control Variant Constants" (page 139). Added this section to document the `ControlVariant` type.
Jan. 15, 1998	The following corrections were made: Noted Appearance 1.0.2 where applicable. `HiliteControl`. Recategorized from "not recommended with the Appearance Manager" to "unchanged" and, therefore, removed from this delta document.
Dec. 2, 1997	PDF formatting improved.
Nov. 3, 1997	First document release.

11/18/98 © Apple Computer, Inc.

Index

11/18/98 © Apple Computer, Inc.

D

E

F

G

11/18/98 © Apple Computer, Inc.

H

I

K

11/18/98 © Apple Computer, Inc.

11/18/98 © Apple Computer, Inc.

11/18/98 © Apple Computer, Inc.

L

M

N

P

11/18/98 © Apple Computer, Inc.

R

S

T

U

V

W

11/18/98 © Apple Computer, Inc.

This Apple manual was written, edited, and composed on a desktop publishing system using Apple Macintosh computers and FrameMaker software. Line art was created using Adobe™ Illustrator and Adobe Photoshop.

Text type is Palatino® and display type is Helvetica®. Bullets are ITC Zapf Dingbats®. Some elements, such as program listings, are set in Adobe Letter Gothic.

WRITERS
Lisa Karpinski, Donna S. Lee, and Judith Rosado

ILLUSTRATORS
David Arrigoni and Karin Stroud

PRODUCTION EDITOR
Glen Frank

PROJECT MANAGER
Tony Francis

Acknowledgments to Matt Ackeret, Pete Gontier, Guy Fullerton, Chris Thomas, and Ed Voas.

11/18/98 © Apple Computer, Inc.