INSIDE MACINTOSH

Mac OS 8
Dialog Manager Reference

Updated for Appearance 1.0.2

November 10, 1998
Technical Publications

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

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

Figures, Tables, and Listings

The Dialog Manager provides a simplified human interface that you can use to readily implement dialog boxes and alert boxes. Your program typically uses dialog boxes and alert boxes to present information to and solicit information from the user.

Portions of the Dialog Manager application programming interface are new, changed, or not recommended with Appearance Manager 1.0. See the following sections for descriptions of the changes to the Dialog Manager:

Dialog Manager Functions

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

Creating Alert Boxes

The following Dialog Manager functions for creating alert boxes are new, changed, or not recommended with Appearance Manager 1.0:

- `StandardAlert` (page 8) displays a standard alert box. New with Appearance Manager 1.0.

- `Alert` (page 9) displays an alert box and/or plays an alert sound. Changed with Appearance Manager 1.0.
- `StopAlert` (page 11) displays an alert box with a stop icon and/or plays an alert sound. Changed with Appearance Manager 1.0.
- `NoteAlert` (page 12) displays an alert box with a note icon and/or plays an alert sound. Changed with Appearance Manager 1.0.
- `CautionAlert` (page 14) displays an alert box with a caution icon and/or plays an alert sound. Changed with Appearance Manager 1.0.

StandardAlert

Displays a standard alert box.

```
pascal OSErr StandardAlert (
                    AlertType inAlertType,
                    StringPtr inError,
                    StringPtr inExplanation,
                    AlertStdAlertParamPtr inAlertParam,
                    SInt16 *outItemHit);
```

`inAlertType` A constant indicating the type of alert box you wish to create; see "Alert Type Constants" (page 45).

`inError` A pointer to a Pascal string containing the primary error text you wish to display.

`inExplanation` A pointer to a Pascal string containing the secondary text you wish to display; secondary text is displayed in the small system font. Pass `nil` to indicate no secondary text.

`inAlertParam` A pointer to the standard alert structure; see `AlertStdAlertParamRec` (page 34). Pass `nil` to specify that you do not wish to your alert box to incorporate any of the features that the standard alert structure provides.

`outItemHit` A pointer to an signed 16-bit integer value. On return, the value indicates the alert button pressed; see "Alert Button Constants" (page 47).

function result A result code; see "Dialog Manager Result Codes" (page 51).

DISCUSSION

The `StandardAlert` function displays an alert box based on the values you pass it. You can pass the error text you wish displayed in the `error` and `explanation` parameters, and customize the alert button text by filling in the appropriate fields of the standard alert structure passed in the `inAlertParam` parameter.

`StandardAlert` automatically resizes the height of a dialog box to fit all static text. It ignores alert stages and therefore provides no corresponding alert sounds.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Alert

Displays an alert box and / or plays an alert sound.

```
pascal short Alert  (short alertID,
                     ModalFilterUPP modalFilter);
```

`alertID` The resource ID of an alert resource and extended alert resource. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See `'alrx'` (page 39) for a description of the extended alert resource.

`modalFilter` A universal procedure pointer for a filter function that responds to events not handled by the `ModalDialog` (page 30) function. If you set this parameter to `nil`, the Dialog Manager uses the standard event filter function.

function result If no alert box is to be drawn at the current alert stage or the `'ALRT'` resource is not found, `Alert` returns –1; otherwise, it creates and displays the alert box and returns the item number of the control selected by the user; see “Alert Button Constants” (page 47).

DISCUSSION

The `Alert` function displays an alert box or, if appropriate for the alert stage, plays an alert sound instead of or in addition to displaying the alert box. The

`Alert` function creates the alert defined in the specified alert resource and its corresponding extended alert resource. The function calls the current alert sound function and passes it the sound number specified in the alert resource for the current alert stage. If no alert box is to be drawn at this stage, `Alert` returns –1; otherwise, it uses the `NewDialog` function to create and display the alert box. The default system window colors are used unless your application provides an alert color table resource with the same resource ID as the alert resource. The `Alert` function uses the `ModalDialog` (page 30) function to get and handle most events for you.

The `Alert` function does not display a default icon in the upper-left corner of the alert box; you can leave this area blank, or you can specify your own icon in the alert's item list resource, which in turn is specified in the alert resource.

The `Alert` function continues calling `ModalDialog` until the user selects an enabled control (typically a button), at which time the `Alert` function removes the alert box from the screen and returns the item number of the selected control. Your application then responds as appropriate when the user clicks this item.

IMPORTANT

Your application should never draw its own default rings. Prior to Mac OS 8, the `Alert` function would only redraw the default button ring once and never redraw it on an update event. However, when Appearance is available, default rings do redraw when you call `Alert`.

SPECIAL CONSIDERATIONS

If you need to display an alert box while your application is running in the background or is otherwise invisible to the user, call `AEInteractWithUser`; see *Inside Macintosh: Interapplication Communication*.

The Dialog Manager uses the system alert sound as the error sound unless you change it by calling the `ErrorSound` function .

VERSION NOTES

Changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

SEE ALSO

The function `NoteAlert` (page 12).

The function `CautionAlert` (page 14).

The function `StopAlert` (page 11).

StopAlert

Displays an alert box with a stop icon and / or plays an alert sound.

```
pascal short StopAlert (short alertID, ModalFilterUPP modalFilter);
```

`alertID` The resource ID of an alert resource and extended alert resource. The resource ID of both types of resources must be identical. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See `'alrx'` (page 39) for a description of the extended alert resource.

`modalFilter` A universal procedure pointer for a filter function that responds to events not handled by the `ModalDialog` (page 30) function. If you set this parameter to `nil`, the Dialog Manager uses the standard event filter function.

function result If no stop alert box is to be drawn at the current alert stage, `StopAlert` returns –1; otherwise, it creates and displays the alert box and returns the item number of the control selected by the user; see “Alert Button Constants” (page 47).

DISCUSSION

The `StopAlert` function displays an alert box with a stop icon in its upper-left corner or, if appropriate for the alert stage, plays an alert sound instead of or in addition to displaying the alert box.

The `StopAlert` function is the same as the `Alert` function (page 9) except that, before drawing the items in the alert box, `StopAlert` draws the stop icon in the upper-left corner. The stop icon has resource ID 0, which you can also specify with the constant `stopIcon` . By default, the Dialog Manager uses the standard stop icon from the System file. You can change this icon by providing your own `'ICON'` resource with resource ID 0.

Use a stop alert to inform the user that a problem or situation is so serious that the action cannot be completed. Stop alerts typically have only a single button (OK), because all the user can do is acknowledge that the action cannot be completed.

IMPORTANT

Your application should never draw its own default rings or alert icons. Prior to Mac OS 8, the `StopAlert` function would only redraw the alert icon and default button ring once and never redraw them on an update event. However, when Appearance is available, alert icons and default rings do redraw when you call `StopAlert`.

VERSION NOTES

Changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

SEE ALSO

The function `NoteAlert` (page 12).

The function `CautionAlert` (page 14).

NoteAlert

Displays an alert box with a note icon and/or plays an alert sound.

```
pascal short NoteAlert (
                    short alertID,
                    ModalFilterUPP modalFilter);
```

`alertID` The resource ID of an alert resource and extended alert resource. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See `'alrx'` (page 39) for a description of the extended alert resource.

`modalFilter` A universal procedure pointer for a filter function that responds to events not handled by the `ModalDialog` (page 30) function. If you set this parameter to `nil`, the Dialog Manager uses the standard event filter function.

function result If no alert box is to be drawn at the current alert stage, `NoteAlert` returns –1; otherwise, it creates and displays the alert box and returns the item number of the control selected by the user; see "Alert Button Constants" (page 47).

DISCUSSION

The `NoteAlert` function displays an alert box with a note icon in its upper-left corner or, if appropriate for the alert stage, plays an alert sound instead of or in addition to displaying the alert box.

The `NoteAlert` function is the same as the `Alert` (page 9) function except that, before drawing the items in the alert box, `NoteAlert` draws the note icon in the upper-left corner. The note icon has resource ID 1, which you can also specify with the constant `noteIcon`. By default, the Dialog Manager uses the standard note icon from the System file. You can change this icon by providing your own `'ICON'` resource with resource ID 1.

Use a note alert to inform users of a minor mistake that won't have any disastrous consequences if left as is. Usually this type of alert simply offers information, and the user responds by clicking an OK button. Occasionally, a note alert may ask a simple question and provide a choice of responses.

IMPORTANT

Your application should never draw its own default rings or alert icons. Prior to Mac OS 8, the `NoteAlert` function would only redraw the alert icon and default button ring once and never redraw them on an update event. However, when Appearance is available, alert icons and default rings do redraw when you call `NoteAlert`.

VERSION NOTES

Changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

SEE ALSO

The function `CautionAlert` (page 14).

The function `StopAlert` (page 11).

CautionAlert

Displays an alert box with a caution icon and/or plays an alert sound.

```
pascal short CautionAlert (
                    short alertID,
                    ModalFilterUPP modalFilter);
```

`alertID` The resource ID of an alert resource and extended alert resource. If the alert resource is missing, the Dialog Manager returns to your application without creating the requested alert. See `'alrx'` (page 39) for a description of the extended alert resource.

`modalFilter` A universal procedure pointer for a filter function that responds to events not handled by the `ModalDialog` function (page 30). If you set this parameter to `nil`, the Dialog Manager uses the standard event filter function.

function result If no alert box is to be drawn at the current alert stage, `CautionAlert` returns –1; otherwise, it uses `NewDialog` to create and display the alert box and returns the item hit; see "Alert Button Constants" (page 47).

DISCUSSION

Displays an alert box with a caution icon in its upper-left corner or, if appropriate for the alert stage, to play an alert sound instead of or in addition to displaying the alert box.

The `CautionAlert` function is the same as the `Alert` (page 9) function except that, before drawing the items in the alert box, `CautionAlert` draws the caution icon in the upper-left corner. The caution icon has resource ID 2, which you can also specify with the constant `kCautionIcon`. By default, the Dialog Manager uses the standard caution icon from the System file. You can change this icon by providing your own `'ICON'` resource with resource ID 2.

Use a caution alert to alert the user of an operation that may have undesirable results if it's allowed to continue. Give the user the choice of continuing the action (by clicking an OK button) or stopping it (by clicking a Cancel button).

IMPORTANT

Your application should never draw its own default rings or alert icons. Prior to Mac OS 8, the `CautionAlert` function would only redraw the alert icon and default button ring once and never redraw them on an update event. However, when Appearance is available, alert icons and default rings do redraw when you call `CautionAlert`.

VERSION NOTES

Changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

SEE ALSO

The function `NoteAlert` (page 12).

The function `StopAlert` (page 11).

Creating Dialog Boxes

The following Dialog Manager functions for creating dialog boxes are new, changed, or not recommended with Appearance Manager 1.0:

- `GetNewDialog (page 16)` creates a dialog box from a resource-based description. Changed with Appearance Manager 1.0.
- `NewFeaturesDialog` (page 17) creates a dialog box from information passed in memory. New with Appearance Manager 1.0.

GetNewDialog

Creates a dialog box from a resource-based description.

```
pascal DialogPtr GetNewDialog (
                    short dialogID,
                    void *dStorage,
                    WindowPtr behind);
```

`dialogID` The resource ID of a dialog resource and an extended dialog resource. The resource IDs for both resources must be identical. If the dialog resource is missing, the Dialog Manager returns to your application without creating the requested dialog box. See `'DLOG'` (page 35) and `'dlgx'` (page 38) for a description of the dialog resource and the extended dialog resource, respectively.

`dStorage` A pointer to the memory for the dialog structure. If you set this parameter to `nil`, the Dialog Manager automatically allocates a nonrelocatable block in your application heap.

`behind` A pointer to the window behind which the dialog box is to be placed on the desktop. Set this parameter to the window pointer `(WindowPtr)-1L` to bring the dialog box in front of all other windows.

function result Returns a pointer to a dialog box. If none was created, returns `nil`.

DISCUSSION

The `GetNewDialog` function creates a dialog structure from information in a dialog resource and an extended dialog resource (if it exists) and returns a pointer to the dialog structure. You can use this pointer with Window Manager or QuickDraw functions to manipulate the dialog box. If the dialog resource specifies that the dialog box should be visible, the dialog box is displayed. If the dialog resource specifies that the dialog box should initially be invisible, use the Window Manager function `ShowWindow` to display the dialog box.

The dialog resource contains a resource ID that specifies both the dialog box's item list (`'DITL'`) resource and its dialog font table (`'dftb'`) resource. After calling the Resource Manager to read these resources into memory (if they are not already in memory), `GetNewDialog` makes a copy of the `'DITL'` resource and uses that copy; thus you may have several dialog boxes with identical items.

If you supply a dialog color table (`'dctb'`) resource with the same resource ID as the dialog resource, `GetNewDialog` uses `NewColorDialog` and returns a pointer to a color graphics port. If no dialog color table resource is present, `GetNewDialog` uses `NewDialog` to return a pointer to a black-and-white graphics port, although system software draws the window frame using the system's default colors. However, if the Appearance Manager is available and the `kDialogFlagsUseThemeBackground` feature bit of the extended dialog resource is set, then the `'dctb'` resource is ignored and a color graphics port is created.

SPECIAL CONSIDERATIONS

The `GetNewDialog` function doesn't release the memory occupied by the resources. Therefore, your application should mark all resources used for a dialog box as purgeable or you should release the resources yourself.

If either the dialog resource or the item list resource can't be read, the function result is `nil`; your application should test to ensure that `nil` is not returned before performing any more operations with the dialog box or its items.

As with all other windows, dialogs are created with an update region equal to their port rectangle. However, if the dialog's `'DLOG'` resource specifies that the dialog be made visible upon creation, the Dialog Manager draws the controls immediately and calls `ValidRgn` for each of their bounding rectangles. Other items are not drawn until the first update event for the dialog box is serviced.

If you need to display an alert box while your application is running in the background or is otherwise invisible to the user, call `AEInteractWithUser`; see *Inside Macintosh: Interapplication Communication*.

VERSION NOTES

Changed with Appearance Manager 1.0 to support the extended dialog (`'dlgx'`) resource and the dialog font table (`'dftb'`) resource.

NewFeaturesDialog

Creates a dialog box from information passed in memory.

```
pascal DialogPtr NewFeaturesDialog (
                    void *inStorage,
                    const Rect *inBoundsRect,
```

```
                    ConstStr255Param inTitle,
                    Boolean inIsVisible,
                    SInt16 inProcID,
                    WindowPtr inBehind,
                    Boolean inGoAwayFlag,
                    SInt32 inRefCon,
                    Handle inItemListHandle,
                    UInt32 inFlags);
```

`inStorage` A pointer to the memory for the dialog box. If you set this parameter to `nil`, the Dialog Manager automatically allocates a nonrelocatable block in your application heap.

`inBoundsRect` A pointer to a rectangle, given in global coordinates, that determines the size and position of the dialog box; these coordinates specify the upper-left and lower-right corners of the dialog box.

`inTitle` A pointer to a text string used for the title of a modeless or movable modal dialog box. You can specify an empty string (not `nil`) for a title bar that contains no text.

`inIsVisible` A flag that specifies whether the dialog box should be drawn on the screen immediately. If you set this parameter to `false`, the dialog box is not drawn until your application uses the Window Manager function `ShowWindow` to display it.

`inProcID` The window definition ID for the type of dialog box, specified with constants defined by the Window Manager. Use the `kWindowModalDialogProc` constant to specify modal dialog boxes, the `kWindowDocumentProc` constant to specify modeless dialog boxes, and the `kWindowMovableModalDialogProc` constant to specify movable modal dialog boxes.

`inBehind` A pointer to the window behind which the dialog box is to be placed on the desktop. Set this parameter to the window pointer `(WindowPtr)-1L` to bring the dialog box in front of all other windows.

`inGoAwayFlag` A Boolean value. If `true`, specifies that an active modeless dialog box has a close box in its title bar.

`inRefCon` A value that the Dialog Manager uses to set the `refCon` field of the dialog box's window structure. Your application may store any value here for any purpose. For example, your application

can store a number that represents a dialog box type, or it can store a handle to a structure that maintains state information about the dialog box. You can use the Window Manager function `SetWRefCon` at any time to change this value in the dialog structure for a dialog box, and you can use the `GetWRefCon` function to determine its current value.

`inItemListHandle`
A handle to an item list resource for the dialog box. You can get the handle by calling the Resource Manager function `GetResource` to read the item list resource into memory.

`inFlags` An unsigned 32-bit mask specifying the dialog box's Appearance-compliant feature flags; see "Dialog Feature Flag Constants" (page 46). To establish an embedding hierarchy in a dialog box, pass `kDialogFlagsUseControlHierarchy` in the `inFlags` parameter.

function result A pointer to the newly created dialog box. If `NewFeaturesDialog` doesn't create a new dialog box, it returns `nil`.

DISCUSSION

The `NewFeaturesDialog` function creates a dialog box without using `'DLOG'` or `'dlgx'` resources. Although the `inItemListHandle` parameter specifies an item list (`'DITL'`) resource for the dialog box, the corresponding dialog font table (`'dftb'`) resource is not automatically accessed. You must explicitly set the dialog box's control font style(s) individually.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Manipulating Items in Dialog Boxes and Alert Boxes

The following Dialog Manager functions for manipulating items in dialog boxes and alert boxes are new, changed, or not recommended with Appearance Manager 1.0:

- `GetDialogItemAsControl` (page 20) obtains the control handle for a dialog item in an embedding hierarchy. New with Appearance Manager 1.0.

- `GetDialogItem` (page 21) obtains a handle to a dialog item. Changed with Appearance Manager 1.0.
- `SetDialogItem` (page 22) sets or changes information for a dialog item. Changed with Appearance Manager 1.0.
- `GetDialogKeyboardFocusItem` (page 23). Not recommended with Appearance Manager 1.0 and later.
- `FindDialogItem` (page 24) obtains the control handle for a dialog item in an embedding hierarchy. New with Appearance Manager 1.0.
- `MoveDialogItem` (page 25) obtains the control handle for a dialog item in an embedding hierarchy. New with Appearance Manager 1.0.
- `SizeDialogItem` (page 25) obtains the control handle for a dialog item in an embedding hierarchy. New with Appearance Manager 1.0.
- `AutoSizeDialog` (page 26) obtains the control handle for a dialog item in an embedding hierarchy. New with Appearance Manager 1.0.
- `AppendDialogItemList` (page 27) adds items to an existing dialog box while your program is running. New with Appearance Manager 1.0.

GetDialogItemAsControl

Obtains the control handle for a dialog item in an embedding hierarchy.

```
pascal OSErr GetDialogItemAsControl (
                    DialogPtr inDialog,
                    SInt16 inItemNo,
                    ControlHandle *outControl);
```

`inDialog` A pointer to the dialog box to examine.

`inItemNo` The position of an item in the dialog box's item list.

`outControl` A pointer to a control handle that, on return, refers to the embedded control.

function result A result code; see "Dialog Manager Result Codes" (page 51). The Control Manager result code `errItemNotControl` indicates that the specified dialog item is not a control.

DISCUSSION

When an embedding hierarchy is established, `GetDialogItemAsControl` produces a handle to the embedded controls (except Help items). It should be used instead of `GetDialogItem` (page 21) when an embedding hierarchy is established.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

GetDialogItem

Obtains a handle to a dialog item.

```
pascal void GetDialogItem (
                    DialogPtr theDialog,
                    short itemNo,
                    short *itemType,
                    Handle *item,
                    Rect *box);
```

`theDialog` A pointer to the dialog box to examine.

`itemNo` The position of the item in the dialog box's item list resource; use `FindDialogItem` (page 24) to determine this value.

`itemType` A pointer to a short value. On return, the value identifies the item type of the dialog item requested in the `itemNo` parameter. See *Inside Macintosh: Macintosh Toolbox Essentials* for a discussion of dialog item types.

`item` A pointer to an item handle. On return the handle refers to the item specified in the `itemNo` parameter or, for application-defined draw functions, a pointer (coerced to a handle) to the draw function.

`box` A pointer to a rectangle. On return, the rectangle specifies the display rectangle (described in coordinates local to the dialog box), for the item specified in the `itemNo` parameter.

DISCUSSION

The `GetDialogItem` function produces the item type, a handle to the item (or, for application-defined draw functions, the function pointer), and the display rectangle for a specified item in an item list resource. When a control hierarchy is present in the dialog box, `GetDialogItem` gets the appropriate information (for example, a text handle) from the controls. If you wish to get a control handle for a dialog item in an embedding hierarchy, see `GetDialogItemAsControl` (page 20).

You should call `GetDialogItem` before calling functions such as `SetDialogItemText` (page 29) that need a handle to a dialog item.

SEE ALSO

`SetDialogItem` (page 22).

VERSION NOTES

Changed with Appearance Manager 1.0 to support retrieving item information from controls.

SetDialogItem

Sets or changes information for a dialog item.

```
pascal void SetDialogItem (
                    DialogPtr theDialog,
                    short itemNo,
                    short itemType,
                    Handle item,
                    const Rect *box);
```

`theDialog` A pointer to the dialog box containing the dialog item.

`itemNo` The position of the item in the dialog box's item list resource; use `FindDialogItem` (page 24) to determine this value.

`itemType` A short value. Pass an item type constant identifying the dialog item specified in the `itemNo` parameter. See *Inside Macintosh: Macintosh Toolbox Essentials* for a discussion of dialog item types. When an embedding hierarchy is established, only the `kItemDisableBit` item type constant is honored.

item	Either a handle to the dialog item specified in the itemNo parameter or, for a custom dialog item, a pointer (coerced to a handle) to an application-defined item drawing function. When an embedding hierarchy is established, the item parameter is ignored unless you pass a universal procedure pointer to an application-defined item draw function.
box	A pointer to the display rectangle (in local coordinates) for the item specified in the itemNo parameter. If you set the control rectangle on an item when an embedding hierarchy is present, SetDialogItem will move and resize the item appropriately for you, on return.

DISCUSSION

The SetDialogItem function sets the item specified by the itemNo parameter for the specified dialog box. If an embedding hierarchy exists, however, you cannot change the type or handle of an item, although application-defined item drawing functions can still be set.

SEE ALSO

GetDialogItem (page 21).

VERSION NOTES

Changed with Appearance Manager 1.0 to work with embedding hierarchies.

GetDialogKeyboardFocusItem

When the Appearance Manager is available and an embedding hierarchy is established, you should call the Control Manager function GetKeyboardFocus instead of GetDialogKeyboardFocusItem to return the item number of the item in a dialog box that has keyboard focus.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

FindDialogItem

Determines the item number of an item at a particular location in a dialog box.

```
pascal short FindDialogItem (
                    DialogPtr theDialog,
                    Point thePt);
```

`theDialog` A pointer to a dialog structure.

`thePt` The point (in local coordinates) where the mouse-down event occurred.

function result When an embedding hierarchy is established, the `FindDialogItem` function returns the deepest control selected by the user corresponding to the point specified in the `thePt` parameter. When an embedding hierarchy does not exist, `FindDialogItem` performs a linear search of the item list resource and returns a number corresponding to the hit item's position in the item list resource. For example, it returns 0 for the first item in the item list, 1 for the second, and 2 for the third. If the mouse is not over a dialog item, `FindDialogItem` returns –1.

DISCUSSION

The function `FindDialogItem` is useful for changing the cursor when the user moves the cursor over a particular item.

To get the proper item number before calling the `GetDialogItem` (page 21) function or the `SetDialogItem` (page 22) function, add 1 to the result of `FindDialogItem`, as shown here:

```
theItem = FindDialogItem(theDialog, thePoint) + 1;
```

Note that `FindDialogItem` returns the item number of disabled items as well as enabled items.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

MoveDialogItem

Moves a dialog item to a specified location in a window.

```
pascal OSErr MoveDialogItem (
                    DialogPtr inDialog,
                    SInt16 inItemNo,
                    SInt16 inHoriz,
                    SInt16 inVert);
```

`inDialog`	A pointer to the dialog box containing the item to move.
`inItemNo`	The position of the item in the dialog box's item list resource; use `FindDialogItem` (page 24) to determine this value.
`inHoriz`	The new horizontal coordinate for the dialog item.
`inVert`	The new vertical coordinate for the dialog item.
function result	A result code; see "Dialog Manager Result Codes" (page 51).

DISCUSSION

The `MoveDialogItem` function moves a dialog item to a specified location in a window. `MoveDialogItem` ensures that if the item is a control, the control rectangle and the dialog item rectangle (maintained by the Dialog Manager) are always the same.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SizeDialogItem

Sizes a dialog item.

```
pascal OSErr SizeDialogItem (
                    DialogPtr inDialog,
                    SInt16 inItemNo,
                    SInt16 inWidth,
                    SInt16 inHeight);
```

`inDialog`	A pointer to the dialog box containing the item to be resized.

`inItemNo`	The position of the item in the dialog box's item list resource; use `FindDialogItem` (page 24) to determine this value.
`inWidth`	The new width (in pixels) of the dialog item's control rectangle.
`inHeight`	The new height (in pixels) of the dialog item's control rectangle.
function result	A result code; see "Dialog Manager Result Codes" (page 51).

DISCUSSION

The `SizeDialogItem` function resizes a dialog item to a specified size. If the dialog item is a control, the control rectangle and the dialog item rectangle (maintained by the Dialog Manager) are always the same.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

AutoSizeDialog

Automatically resizes static text fields and their dialog boxes to accommodate changed static text.

```
pascal OSErr AutoSizeDialog (DialogPtr inDialog);
```

`inDialog`	A pointer to a dialog box.
function result	A result code; see "Dialog Manager Result Codes" (page 51).

DISCUSSION

The `AutoSizeDialog` function is useful in situations such as localization, where the size of a static text field (and the dialog box that contains it) may need to be altered to accommodate a change in the size of the static text.

For each static text item `AutoSizeDialog` finds in the item list resource, it adjusts the static text field and the bottom of the dialog box window to accommodate the text. Any items below a static text field are moved down. If the dialog box is visible when this function is called, it is hidden, resized, and then shown. If the dialog box has enough room to show the text as is, no resizing is done.

Note that the `AutoSizeDialog` function does not process update events for your dialog box, so your program must call the `DrawDialog` function or the `DialogSelect` function to process the update event generated from showing the window.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

AppendDialogItemList

Adds items to an existing dialog box while your program is running.

```
pascal OSErr AppendDialogItemList (
                    DialogPtr dialog,
                    SInt16 ditlID,
                    DITLMethod method);
```

`dialog` A pointer to the dialog box to which the items in the item list resource specified in the `ditlID` parameter are to be appended.

`ditlID` The resource ID of the item list resource whose items are to be appended to the dialog box specified in the `dialog` parameter.

`method` The manner in which the new items are to be displayed in the dialog box.
If you use the `overlayDITL` constant, `AppendDialogItemList` superimposes the appended items over the dialog box by interpreting the coordinates of the display rectangles for the appended items (as specified in their item list resource) as local coordinates within the dialog box.
If you use the `appendDITLRight` constant, `AppendDialogItemList` appends the items to the right of the dialog box by positioning the display rectangles of the appended items relative to the upper-right coordinate of the dialog box. The `AppendDialogItemList` function automatically expands the dialog box to accommodate the new dialog items.
If you use the `appendDITLBottom` constant, `AppendDialogItemList` appends the items to the bottom of the dialog box by positioning the display rectangles of the appended items relative to the lower-left coordinate of the dialog box. The

AppendDialogItemList function automatically expands the dialog box to accommodate the new dialog items.
You can append a list of items relative to an existing item by passing a negative number. The absolute value of this number is interpreted as the item in the dialog box relative to which the new items are to be positioned. For example, if you pass -2, the display rectangles of the appended items are offset relative to the upper-left corner of item number 2 in the dialog box.

function result A result code; see "Dialog Manager Result Codes" (page 51).

DISCUSSION

To be Appearance-compliant, your program should use the AppendDialogItemList function rather than the AppendDITL function. Unlike AppendDITL, the AppendDialogItemList function takes a 'DITL' resource ID instead of a handle as the parameter describing the dialog item list to be appended, and it properly appends entries from a dialog font table ('dftb') resource, if there is a 'dftb' resource with the same resource ID as the 'DITL' resource.

The AppendDialogItemList function adds the items in the item list resource specified in the parameter ditlID to the items of a dialog box. This is especially useful if several dialog boxes share a single item list resource, because you can use AppendDialogItemList to add items that are appropriate for individual dialog boxes. Your application can use the Resource Manager function GetResource to get a handle to the item list resource whose items you wish to add.

You typically create an invisible dialog box, call the AppendDialogItemList function, then make the dialog box visible by using the Window Manager function ShowWindow.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Handling Text in Alert and Dialog Boxes

The following Dialog Manager function for handling text in dialog boxes and alert boxes is changed with Appearance Manager 1.0:

- SetDialogItemText (page 29) sets the text string for static text and editable text fields. Changed with Appearance Manager 1.0.

SetDialogItemText

Sets the text string for static text and editable text fields.

```
pascal void SetDialogItemText (
                    Handle item,
                    ConstStr255Param text);
```

item — A handle to an editable text field or static text field. When embedding is on, you should pass in the control handle produced by a call to the function GetDialogItemAsControl (page 20). If embedding is not on, pass in the handle produced by the GetDialogItem (page 21) function.

text — A pointer to a string containing the text to display in the field.

DISCUSSION

The SetDialogItemText function sets and redraws text strings for static text and editable text fields. SetDialogItemText is useful for supplying a default text string—such as a document name—for an editable text field while your application is running.

VERSION NOTES

Changed with Appearance Manager 1.0 to support embedding hierarchies.

Handling Events in Dialog Boxes

The following Dialog Manager function for handling events in dialog boxes is changed with Appearance Manager 1.0:

- ModalDialog (page 30) handles events while your application displays a modal or movable modal dialog box. Changed with Appearance Manager 1.0.

ModalDialog

Handles events while your application displays a modal or movable modal dialog box.

```
pascal void ModalDialog (
                    ModalFilterUPP modalFilter,
                    short *itemHit);
```

`modalFilter` A universal procedure pointer for an event filter function. For modal dialog boxes, you can specify `nil` if you want to use the standard event-handling function. For movable modal dialog boxes, you should specify your own event filter function.

`itemHit` A pointer to a short integer. After receiving an event involving an enabled item, `ModalDialog` produces a number representing the position of the selected item in the active dialog box's item list resource.

DISCUSSION

Call the `ModalDialog` function immediately after displaying a modal or movable modal dialog box. Your application should continue calling `ModalDialog` until the user dismisses your dialog.

For modal dialogs, the `ModalDialog` function repeatedly handles events until an event involving an enabled dialog box item—such as a click in a radio button, for example—occurs. If the event is a mouse-down event outside the content region of the dialog box, `ModalDialog` plays the system alert sound and gets the next event.

For movable modal dialogs, if the `kDialogFlagsHandleMovableModal` feature bit in the extended dialog resource is set, the `ModalDialog` function will handle all standard movable modal user interactions, such as dragging a dialog box by its title bar and allowing the user to switch into another application. However, a difference between the `ModalDialog` function's behavior with movable modal and modal dialogs is that, with movable modal dialogs, your event filter function receives all events. If you want the Dialog Manager to assist you in handling events in movable modal dialog boxes, call `GetStdFilterProc` and `StdFilterProc`.

For events inside the dialog box, `ModalDialog` passes the event to the event filter function pointed to in the `modalFilter` parameter before handling the event. When the event filter returns `false`, `ModalDialog` handles the event. If the event

filter function handles the event, returning `true`, `ModalDialog` performs no more event handling.

If you set the `modalFilter` parameter to `nil`, the standard event filter function is executed. The standard event filter function checks whether

- the user has pressed the Enter or Return key and, if so, returns the item number of the default button
- the user has pressed the Escape key or Command-period and, if so, returns the item number of the Cancel button
- the cursor is over an editable text box, and optionally changes the cursor to an I-beam whenever this is the case

If you set the `modalFilter` parameter to point to your own event filter function, that function can use the standard filter function to accomplish the above tasks. (To do so, you can call `GetStdFilterProc`, and dispatch the event to the standard filter function yourself, or you can call `StdFilterProc` , which obtains a `ModalFilterUPP` for the standard filter function and then dispatches the function.) Additionally, your own event filter function should also

- handle update events, so that background processes can receive processor time, and return `false`
- return `false` for all events that your event filter function doesn't handle

You can also use your event filter function to test for and respond to keyboard equivalents and more complex events—for instance, the user dragging the cursor within an application-defined item. You can use your same event filter function in most or all of your alert and modal dialog boxes.

If the event filter function does not handle the event (returning `false`), `ModalDialog` handles the event as follows:

- In response to an activate or update event for the dialog box, `ModalDialog` activates or updates its window.
- If the user presses the mouse button while the cursor is in an editable text item, `ModalDialog` responds to the mouse activity as appropriate—that is, either by displaying an insertion point or by selecting text. If a key-down event occurs and there's an editable text item, `ModalDialog` uses TextEdit to handle text entry and editing automatically. If the editable text item is enabled, `ModalDialog` produces its item number after it receives either the mouse-down or key-down event. Normally, editable text items are disabled,

and you use the `GetDialogItemText` function to read the information in the items only after the user clicks the OK button.

- If the user presses the mouse button while the cursor is in a control, `ModalDialog` calls the Control Manager function `TrackControl`. If the user releases the mouse button while the cursor is in an enabled control, `ModalDialog` produces the control's item number. Your application should respond appropriately—for example, by performing a command after the user clicks the OK button.
- If the user presses the mouse button while the cursor is in any other enabled item in the dialog box, `ModalDialog` produces the item's number, and your application should respond appropriately. Generally, only controls should be enabled. If your application creates a control more complex than a button, radio button, or checkbox, your application must handle events inside that item with your event filter function.
- If the user presses the mouse button while the cursor is in a disabled item or in no item, or if any other event occurs, `ModalDialog` does nothing.

SPECIAL CONSIDERATIONS

The `ModalDialog` function traps all events. This prevents your event loop from receiving activate events for your windows. Thus, if one of your application's windows is active when you use `GetNewDialog` to create a modal dialog box, you must explicitly deactivate that window before displaying the modal dialog box.

When `ModalDialog` calls the Control Manager function `TrackControl`, it does not allow you to specify the action function necessary for anything more complex than a button, radio button, or checkbox. If you need a more complex control, you can create your own control, a picture, or an application-defined item that draws a control-like object in your dialog box. You must then provide an event filter function that appropriately handles events in that item.

VERSION NOTES

Changed with Appearance Manager 1.0 to handle events for movable modal dialogs.

Defining Your Own Dialog Item Function

When the Appearance Manager is available and an embedding hierarchy is established in a dialog box, you should provide the Control Manager user pane drawing function `MyUserPaneDrawProc` instead of the user item drawing function `MyUserItemProc` to draw an application-defined control (a dialog item becomes a control in a dialog box with an embedding hierarchy).

You can provide other user pane application-defined functions to hit test, track, perform idle processing, handle keyboard, activate, and deactivate event processing, handle keyboard focus, and set the background color or pattern in a user pane control. For examples of how to write these functions, see *Mac OS 8 Control Manager Reference*.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

Dialog Manager Data Types

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

- `AlertStdAlertParamRec` (page 34)
- `'DLOG'` (page 35)
- `'dlgx'` (page 38)
- `'alrx'` (page 39)
- `'dftb'` (page 40)
- `'dctb'` (page 44)
- `'actb'` (page 44)
- `'ictb'` (page 44)

AlertStdAlertParamRec

A standard alert structure of type `AlertStdAlertParamRec` can be used when you call the function `StandardAlert` (page 8) to customize the alert box. The `AlertStdAlertParamRec` type is available with Appearance Manager 1.0 and later.

```
struct AlertStdAlertParamRec {
    Boolean             movable;
    Boolean             helpButton;
    ModalFilterUPP      filterProc;
    StringPtr           defaultText;
    StringPtr           cancelText;
    StringPtr           otherText;
    SInt16              defaultButton;
    SInt16              cancelButton;
    UInt16              position;
};
typedef struct AlertStdAlertParamRec AlertStdAlertParamRec;
typedef AlertStdAlertParamRec *AlertStdAlertParamPtr;
```

Field descriptions

`movable` A Boolean value indicating whether or not the alert box is movable.

`helpButton` A Boolean value indicating whether or not the alert includes a Help button.

`filterProc` If the value in the `movable` field is `true` (alert is movable), then specify in this parameter a universal procedure pointer to an application-defined filter function that responds to events not handled by `ModalDialog` (page 30). If you do, all events will get routed to your application-defined event filter function for handling, even when your alert box window is in the background. If you set this parameter to `nil`, the Dialog Manager uses the standard event filter function.

`defaultText` Text for button in OK position; see “Alert Default Text Constants” (page 48). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1`. To indicate that no button should be displayed, pass `nil`.

`cancelText`	Text for button in Cancel position; see "Alert Default Text Constants" (page 48). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1`. To indicate that no button should be displayed, pass `nil`.
`otherText`	Text for button in leftmost position; see "Alert Default Text Constants" (page 48). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1`. To indicate that no button should be displayed, pass `nil`.
`defaultButton`	Specifies which button acts as the default button; see "Alert Button Constants" (page 47).
`cancelButton`	Specifies which button acts as the Cancel button (can be 0); see "Alert Button Constants" (page 47).
`position`	The alert box position, as defined by a window positioning constant; see *Macintosh Toolbox Essentials* for a discussion of these constants. In this structure, the constant `kWindowDefaultPosition` is equivalent to the constant `kWindowAlertPositionParentWindowScreen`.

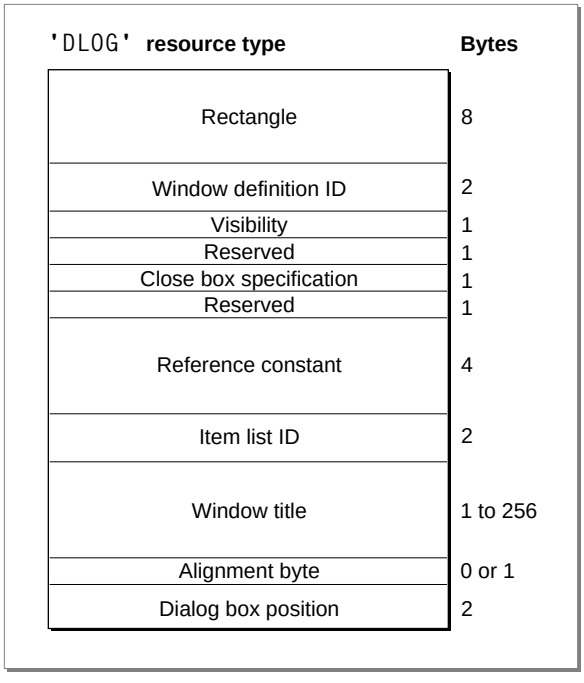
'DLOG'

A dialog resource is a resource of type `'DLOG'`. All dialog resources can be marked purgeable, and they must have resource ID numbers greater than 127. With Appearance Manager 1.0, the `'DLOG'` resource has been changed to support the dialog font table (`'dftb'`) resource; see `'dftb'` (page 40) for a description of the dialog font table resource.

You can use an extended dialog resource with the same resource ID as your dialog resource to provide additional features for your dialog box, including movable modal behavior, Appearance-compliant backgrounds and controls, and embedding hierarchies. See `'dlgx'` (page 38) for a description of the extended dialog resource. Use the `GetNewDialog (page 16)` function to create the dialog box defined in the dialog resource and extended dialog resource.

Figure 1-1 shows the format of a compiled dialog resource.

Figure 1-1 Structure of a compiled dialog (`'DLOG'`) resource



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

- Rectangle. This determines the dialog box's dimensions and, possibly, its position. (The last element in the dialog resource usually specifies a position for the dialog box.)
- Window definition ID. See *Mac OS 8 Window Manager Reference* for descriptions of the ID constants that can be used in this field to specify Appearance-compliant modal, movable modal, or modeless dialog boxes.
- Visibility. If this is set to a value of 1 (as specified by the `visible` constant in the Rez input file), the Dialog Manager displays this dialog box as soon as you call the `GetNewDialog` function (page 16). If this is set to a value of 0 (as specified by the `invisible` constant in the Rez input file), the Dialog Manager does not display this dialog box until you call the Window Manager function `ShowWindow`.

- Close box specification. This specifies whether to draw a close box. Normally, this is set to a value of 1 (as specified by the `goAway` constant in the Rez input file) only for a modeless dialog box to specify a close box in its title bar. Otherwise, this is set to a value of 0 (as specified by the `noGoAway` constant in the Rez input file).
- Reference constant. This contains any value that an application stores here. For example, an application can store a number here that represents a dialog box type, in order to distinguish between a number of similar dialog boxes. As this information is stored in a window structure within the dialog structure, you can use the Window Manager function `SetWRefCon` at any time to change this value in the dialog structure for a dialog box, and you can use the Window Manager function `GetWRefCon` to determine its current value.
- Item list resource ID. This ID specifies both the item list resource and the dialog font table resource that will be used with this dialog box. See `'dftb'` (page 40) for a description of the dialog font table resource.
- Window title. A string displayed in the dialog box's title bar only when the dialog box is modeless or movable modal.
- Alignment byte. This is an extra byte added if necessary to make the previous Pascal string end on a word boundary.
- Dialog box position. A constant specifying the position of the dialog box on the screen; see the discussion of window positioning constants in *Macintosh Toolbox Essentials* (page 4-126). If your application positions dialog boxes on its own, you shouldn't use these constants, because they may cause conflicts with the Dialog Manager.
  - If 0x0000 appears here (as specified by the `kWindowDefaultPosition` constant in the Rez input file), the Dialog Manager positions this dialog box according to the global coordinates specified in the rectangle element of this resource.
  - If 0xB00A appears here (as specified by the `kWindowAlertPositionParentWindow` constant in the Rez input file), the Dialog Manager positions the dialog box over the frontmost window so that the window's title bar appears.
  - If 0x300A appears here (as specified by the `kWindowAlertPositionMainScreen` constant in the Rez input file), the Dialog Manager centers the dialog box near the top of the main screen.

- If 0x700A appears here (as specified in the Rez input file by the `kWindowAlertPositionParentWindowScreen` constant), the Dialog Manager positions the dialog box on the screen where the user is currently working.

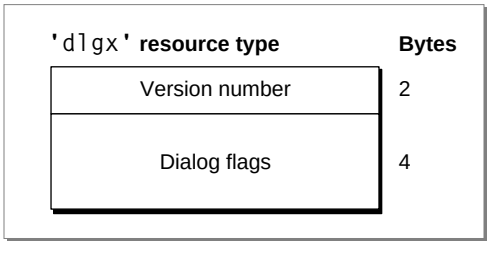
'dlgx'

Use an extended dialog (`'dlgx'`) resource with the same resource ID as your dialog resource to provide additional features for your dialog box, including movable modal behavior, theme-compliant backgrounds and controls, and embedding hierarchies. The extended dialog resource is available with Appearance 1.0 and later.

All extended dialog resources can be marked purgeable, and they must have the same resource ID and be located in the same file as their corresponding dialog resource. Use the function `GetNewDialog (page 16)` to create the dialog box defined in the dialog resource and extended dialog resource.

Figure 1-2 shows the format of a compiled extended dialog resource.

Figure 1-2 Structure of a compiled extended dialog `('dlgx')` resource



The compiled version of an extended dialog resource contains the following elements:

- Version number. An integer specifying the version of the format of the resource.
- Dialog flags. Constants that specify the dialog box's Appearance features; see "Dialog Feature Flag Constants" (page 46).

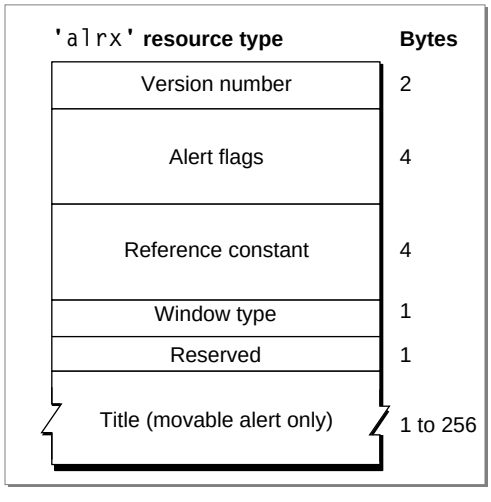
'alrx'

You can use an extended alert (`'alrx'`) resource with the same resource ID as your alert resource to provide additional features for your alert box, including movable modal behavior, Appearance-compliant backgrounds and controls, and embedding hierarchies. The resource also gives you the option of creating a title for movable alert boxes. The extended alert resource is available with Appearance 1.0 and later.

Note
Alert titles are only available with Appearance version 1.0.1 and later.

All extended alert resources can be marked purgeable, and they must have the same resource ID and resource file as their corresponding alert resource. Figure 1-3 shows the structure of a compiled extended alert resource.

Figure 1-3 Structure of a compiled extended alert `('alrx')` resource



The compiled version of an extended alert resource contains the following elements:

- Version number. An integer specifying the version of the format of the resource.

- Alert flags. Constants specifying the alert box's Appearance features; see "Alert Feature Flag Constants" (page 46).
- Reference constant. This contains any value that an application wishes to store here. For example, an application can store a number here that represents an alert box type, in order to distinguish between a number of similar alert boxes. As this information is stored in a window structure within the dialog structure, you can use `GetWRefCon` to determine this value.
- Window type. If this Boolean is set to 1 (`true`), the Dialog Manager specifies an Appearance-compliant window definition ID constant directly when drawing the alert box window.
- Reserved. Set to 0.
- Window title. A string representing the title of a movable alert box.

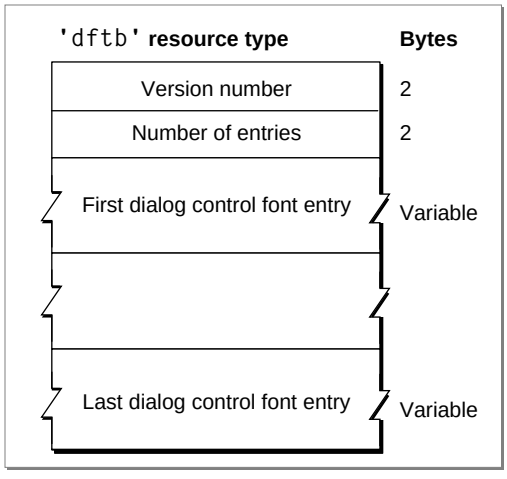
'dftb'

Your application can specify the initial font settings for all controls in a dialog box or alert box by creating a dialog font table resource of type `'dftb'` with the same resource ID as the item list resource (`'DITL'`). The control font style information in the dialog font table resource is automatically read in (along with the `'DITL'`) by the Dialog Manager. When the `'dftb'` resource is read in, the control font styles are set, and the resource is marked purgeable.

When an embedding hierarchy is established in a dialog box, the dialog font table resource should be used instead of the item color table (`'ictb'`) resource, since edit and static text dialog items become controls in an embedding hierarchy. The dialog font table resource is available with Appearance 1.0 and later.

Figure 1-4 shows the format of a compiled dialog font table resource.

Figure 1-4 Structure of a compiled dialog font table (`'dftb'`) resource

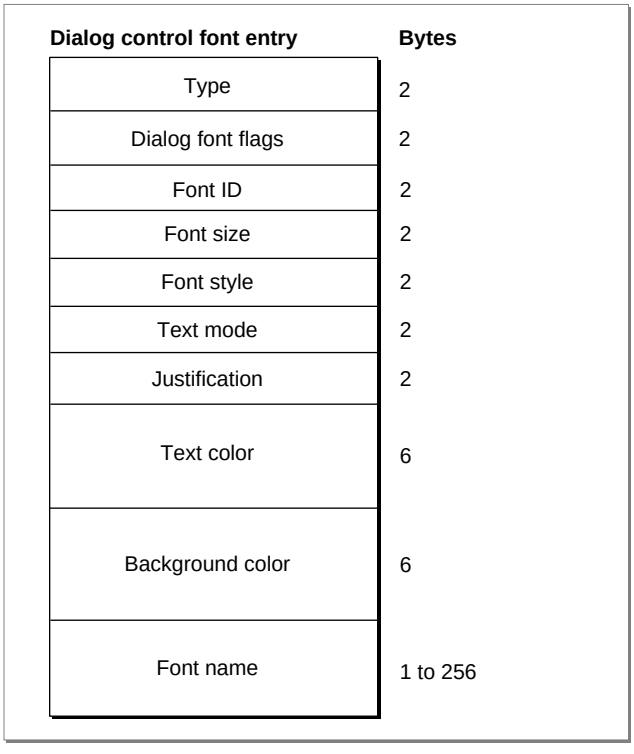


A compiled version of a `'dftb'` resource contains the following elements:

- Version number. An integer specifying the version of the format of the resource.
- Number of entries. An integer that specifies the number of entries in the resource. Each entry is a dialog control font structure.
- Dialog control font entries. A series of dialog control font structures, each of which consist of type, dialog font flags, the font ID, font size, font style, text mode, justification, text color, background color, and font name.

Figure 1-5 shows the format of a compiled dialog control font entry in a `'dftb'` resource.

Figure 1-5 Structure of dialog control font entry in a `'dftb'` resource



Each entry in a `'dftb'` resource corresponds to a dialog item and contains the following elements:

- Type. An integer that specifies whether there is font information for the dialog or alert item in the `'DITL'`. If you specify a value of 0, there is no font information for the item in the corresponding `'DITL'`, and no data follows. If you specify a value of 1, there is font information for the item, and the rest of the structure is read. You can cause entries to be skipped by setting this value to 0.
- Dialog font flags. You can use one or more of these flag constants to specify which other fields in the dialog font table should be used; see “Dialog Font Flag Constants” (page 49).

- Font ID. If the `kDialogFontUseFontMask` bit is set to 1, then this element will contain an integer indicating the ID of the font family to use. See *Mac OS 8 Control Manager Reference* for information about the metafont constants that you can specify in this field. If this bit is set to 0, then the system default font is used.
- Font size. If a constant representing the system font, small system font, or small emphasized system font is specified in the Font ID field, this field is ignored. If the `kDialogFontUseSizeMask` bit is set, this field should contain an integer representing the point size of the text. If the `kDialogFontAddSizeMask` bit is set, this value will contain the size to add to the current point size of the text.
- Style. If the `kDialogFontUseFaceMask` bit is set, then this element should contain an integer specifying the text style to describe which styles to apply to the text. You can use one or more of the following `style` data type mask constants to specify font style:

Bit value	Style
0x00	Normal
0x01	Bold
0x02	Italic
0x04	Underline
0x08	Outline
0x10	Shadow
0x20	Condense
0x40	Extend

- Text mode. If the `kDialogFontUseModeMask` bit is set, then this element should contain an integer specifying how characters are drawn. See *Inside Macintosh: Imaging With QuickDraw* for a discussion of source transfer modes.
- Justification. If the `kDialogFontUseJustMask` bit is set, then this element should contain an integer specifying text justification (left, right, centered, or system-justified).
- Text color. If the `kDialogUseFontForeColorMask` bit is set, then this element should contain a color to use when drawing the text.
- Background color. If the `kDialogFontUseBackColorMask` bit is set, then this element should contain a color to use when drawing the background behind the text. In certain text modes, background color is ignored.

- Font name. If the `kDialogFontUseFontNameMask` bit is set, then this element should contain a Pascal string representing the font name to be used. This overrides the font ID.

'dctb'

The dialog color table resource is not recommended with Appearance Manager 1.0 and later. When the Appearance Manager is available and the `kDialogFlagsUseThemeBackground` feature bit of the extended dialog resource (page 35) is set, the entire dialog color table resource (`'dctb'`) is ignored. If the Appearance Manager is available, but the above is not true, the `wContent` field of the `'dctb'` resource is used, but all other fields are still ignored.

'actb'

The alert color table resource is not recommended with Appearance Manager 1.0 and later. When the Appearance Manager is available and the `kAlertFlagsUseThemeBackground` feature bit of the extended alert resource is set, the entire alert color table resource (`'actb'`) is ignored. If the Appearance Manager is available, but the `kAlertFlagsUseThemeBackground` bit is not set, the `wContent` field of the `'actb'` resource is used, but all other fields are still ignored.

'ictb'

The item color table resource is not recommended with Appearance Manager 1.0 and later. When the Appearance Manager is available and an embedding hierarchy is established in the dialog box, any item color table (`'ictb'`) resource information is ignored. The dialog font table (`'dftb'`) resource should be used instead of the item color table resource to specify the font settings for all dialog items in an embedding hierarchy.

If an embedding hierarchy is not established, the item color table resource can be used to set the font information for any editable text and static text dialog items, but the dialog font table resource will still be used for any controls in the dialog box.

Dialog Manager Constants

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

Alert Type Constants

You can pass constants of type `AlertType` in the `inAlertType` parameter of `StandardAlert` (page 8) to specify the type of alert box you wish to create. Alert type constants are available with Appearance Manager 1.0 and later.

```
enum {
    kAlertStopAlert     = 0,
    kAlertNoteAlert     = 1,
    kAlertCautionAlert  = 2,
    kAlertPlainAlert    = 3
};
typedef SInt16 AlertType;
```

Constant descriptions

`kAlertStopAlert` Stop alert box.

`kAlertNoteAlert` Note alert box.

`kAlertCautionAlert` Caution alert box.

`kAlertPlainAlert` Alert box with no icon.

Dialog Feature Flag Constants

You can set the following bits in the dialog flags field of the extended dialog resource (page 35) or pass them in the `inFlags` parameter of `NewFeaturesDialog` (page 17) to specify the dialog box's Appearance-compliant features. Dialog feature flag constants are available with Appearance Manager 1.0 and later.

```
enum {
    kDialogFlagsUseThemeBackground      = (1 << 0),
    kDialogFlagsUseControlHierarchy     = (1 << 1),
    kDialogFlagsHandleMovableModal      = (1 << 2),
    kDialogFlagsUseThemeControls        = (1 << 3)
};
```

Constant descriptions

`kDialogFlagsUseThemeBackground`
: If this bit (bit 0) is set, the Dialog Manager sets the dialog box's background color or pattern.

`kDialogFlagsUseControlHierarchy`
: If this bit (bit 1) is set, the Dialog Manager creates a root control in the dialog box and establishes an embedding hierarchy. Any dialog items become controls once the embedding hierarchy is established.

`kDialogFlagsHandleMovableModal`
: If this bit (bit 2) is set, and the dialog box is a movable modal (specify the `kWindowMovableModalDialogProc` window definition ID), the Dialog Manager handles movable modal behavior such as dragging a dialog box by its title bar or switching out of the application by clicking in another one.

`kDialogFlagsUseThemeControls`
: If this bit (bit 3) is set, the Dialog Manager creates Appearance-compliant controls in the dialog box directly. Otherwise, push buttons, checkboxes, and radio buttons will be displayed in their pre-Appearance forms when systemwide Appearance is off.

Alert Feature Flag Constants

You can set the following bits in the alert flags field of the extended alert resource (page 39) to specify the alert box's Appearance-compliant features.

Alert feature flag constants are available with Appearance Manager 1.0 and later.

```
enum {
    kAlertFlagsUseThemeBackground           = (1 << 0),
    kAlertFlagsUseControlHierarchy          = (1 << 1),
    kAlertFlagsAlertIsMovable               = (1 << 2),
    kAlertFlagsUseThemeControls             = (1 << 3)
};
```

`kAlertFlagsUseThemeBackground`
If this bit (bit 0) is set, the Dialog Manager sets the alert box's background color or pattern.

`kAlertFlagsUseControlHierarchy`
If this bit (bit 1) is set, the Dialog Manager creates a root control in the alert box and establishes an embedding hierarchy. Any alert items become controls once the embedding hierarchy is established.

`kAlertFlagsAlertIsMovable`
If this bit (bit 2) is set, the alert box is movable modal. The Dialog Manager handles movable modal behavior such as dragging the alert box by its title bar or switching out of the application by clicking in another one.

`kAlertFlagsUseThemeControls`
If this bit (bit 3) is set, the Dialog Manager creates Appearance-compliant controls in your alert box. Otherwise, push buttons, checkboxes, and radio buttons will be displayed in their pre-Appearance forms when systemwide Appearance is off.

Alert Button Constants

You can use these constants in the `defaultButton` and `cancelButton` fields in the standard alert structure (page 34) to specify which buttons act as the default and Cancel buttons in the standard alert structure. These constants are also returned in the `itemHit` parameter of `StandardAlert` (page 8). Alert button constants are available with Appearance Manager 1.0 and later.

```
enum {
    kAlertStdAlertOKButton      = 1,
    kAlertStdAlertCancelButton  = 2,
    kAlertStdAlertOtherButton   = 3,
    kAlertStdAlertHelpButton    = 4
};
```

Constant descriptions

`kAlertStdAlertOKButton`
The OK button. The default text for this button is “OK”.

`kAlertStdAlertCancelButton`
The Cancel button (optional). The default text for this button is “Cancel”.

`kAlertStdAlertOtherButton`
A third button (optional). The default text for this button is “Don’t Save”.

`kAlertStdAlertHelpButton`
The Help button (optional).

Alert Default Text Constants

You can use these constants in the `defaultText`, `cancelText`, and `otherText` fields of the standard alert structure (page 34) to specify the default text for the OK, Cancel, and Don’t Save buttons. Alert default text constants are available with Appearance Manager 1.0 and later.

```
enum {
    kAlertDefaultOKText     = -1,
    kAlertDefaultCancelText = -1,
    kAlertDefaultOtherText  = -1
};
```

Constant descriptions

`kAlertDefaultOKText`
The default text for the default (right) button is “OK” on an English system. The text will vary depending upon the localization of the user’s system. Use this constant in the `defaultText` field of the standard alert structure (page 34).

`kAlertDefaultCancelText`

The default text for the Cancel (middle) button is “Cancel” on an English system. The text will vary depending upon the localization of your system. Use this constant in the `cancelText` field of the standard alert structure (page 34).

`kAlertDefaultOtherText`

The default text for the third (leftmost) button is “Don’t Save” for an English system. The text will vary depending upon the localization of the user’s system. Use this constant in the `otherText` field of the standard alert structure.

Dialog Font Flag Constants

You can set the following bits in the dialog font table resource (page 40) to specify fields in the dialog font table that should be used. Dialog font flag constants are available with Appearance Manager 1.0 and later.

```
enum {
    kDialogFontNoFontStyle      = 0,
    kDialogFontUseFontMask      = 0x0001,
    kDialogFontUseFaceMask      = 0x0002,
    kDialogFontUseSizeMask      = 0x0004,
    kDialogFontUseForeColorMask = 0x0008,
    kDialogFontUseBackColorMask = 0x0010,
    kDialogFontUseModeMask      = 0x0020,
    kDialogFontUseJustMask      = 0x0040,
    kDialogFontUseAllMask       = 0x00FF,
    kDialogFontAddFontSizeMask  = 0x0100,
    kDialogFontUseFontNameMask  = 0x0200
};
```

Constant descriptions

`kDialogFontNoFontStyle`

If the `kDialogFontNoFontStyle` constant is used, no font style information is applied.

`kDialogFontUseFontMask`

If the `kDialogFontUseFontMask` flag (bit 0) is set, the font ID specified in the Font ID field of the dialog font table is applied.

`kDialogFontUseFaceMask`

If the `kDialogFontUseFaceMask` flag (bit 1) is set, the font style specified in the Style field of the dialog font table is applied.

`kDialogFontUseSizeMask`

If the `kDialogFontUseSizeMask` flag (bit 2) is set, the font size specified in the Font Size field of the dialog font table is applied.

`kDialogFontUseForeColorMask`

If the `kDialogFontUseForeColorMask` flag (bit 3) is set, the text color specified in the Text Color field of the dialog font table is applied. This flag only applies to static text controls.

`kDialogFontUseBackColorMask`

If the `kDialogFontUseBackColorMask` flag (bit 4) is set, the background color specified in the Background Color field of the dialog font table is applied. This flag only applies to static text controls.

`kDialogFontUseModeMask`

If the `kDialogFontUseModeMask` flag (bit 5) is set, the text mode specified in the Text Mode field of the dialog font table is applied.

`kDialogFontUseJustMask`

If the `kDialogFontUseJustMask` flag (bit 6) is set, the text justification specified in the Justification field of the dialog font table is applied.

`kDialogFontUseAllMask`

If the `kDialogFontUseAllMask` constant is used, all flags in this mask will be set except `kDialogFontAddFontSizeMask` and `kDialogFontUseFontNameMask`.

`kDialogFontAddFontSizeMask`

If the `kDialogFontAddFontSizeMask` flag (bit 8) is set, the Dialog Manager will add a specified font size to the existing font size indicated in the Font Size field of the dialog font table resource.

`kDialogFontUseFontNameMask`

If the `kDialogFontUseFontNameMask` flag (bit 9) is set, the Dialog Manager will use the string in the Font Name field for the font name instead of a font ID.

Dialog Manager Result Codes

The most common result codes that can be returned by Dialog 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 Manager not set up

Version History

This document has had the following releases:

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

Version	Notes
Nov. 10, 1998	Removed "Dialog Manager Reference" chapter from *Mac OS 8 Toolbox Reference*, making *Mac OS 8 Dialog Manager Reference* available as an independent document. The following corrections were also made at this time: `AppendDialogItemList` (page 27). A description of this function has been added. `SizeDialogItem` (page 25). Corrected the parameter order for this function. `AutoSizeDialog` (page 26). Amended the description of this function to note that it does not process update events.
Jan. 15, 1998	The following corrections were made: Noted Appearance 1.0.2 where applicable.
Dec. 10, 1997	The following corrections were made: `GetNewDialog (page 16)`. Specified the creation of a color graphics port under Appearance when the `kDialogFlagsUseThemeBackground` feature bit of the `'dlgx'` resource is set.
Dec. 2, 1997	PDF formatting improved.
Nov. 3, 1997	First document release.

APPENDIX

Version History

Index

M

N

P

R

S

U

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, Judith Rosado, and Jun Suzuki

ILLUSTRATORS
David Arrigoni and Karin Stroud

PRODUCTION EDITOR
Glen Frank

PROJECT MANAGER
Tony Francis

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