Dialog Manager Reference

Carbon > User Experience

2007-10-31

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

Simultaneously published in the United States and Canada.

Contents

Dialog Manager Reference

Framework: Carbon/Carbon.h

Declared in Dialogs.h

Overview

Your application can use the Dialog Manager to alert users to unusual situations and to solicit information from users. For example, in some situations your application might not be able to carry out a command normally, and in other situations the user must specify multiple parameters before your application can execute a command. For circumstances like these, the Macintosh user interface includes these two features:

- alerts–including alert sounds and alert boxes–which warn the user whenever an unusual or potentially undesirable situation occurs within your application
- dialog boxes, which allow the user to provide additional information or to modify settings before your application carries out a command

Virtually all applications need to implement alerts and dialog boxes. To avoid needless development effort, use the Dialog Manager to implement alerts and to create most dialog boxes. It is possible, however–and sometimes desirable–to bypass the Dialog Manager and instead use Window Manager, Control Manager, QuickDraw, and Event Manager routines to create or respond to events in complex dialog boxes.

Carbon supports the majority of the Dialog Manager. However, your application must access Dialog Manager data structures only through the supplied accessor functions. Furthermore, your application must use the functions provided for creating and disposing of Dialog Manager data structures.

Functions by Task

Creating Alert Boxes

`StandardAlert` (page 65)
Displays a standard alert box.

`Alert` (page 12)
Displays an alert box and/or plays an alert sound.

`StopAlert` (page 67)
Displays an alert box with a stop icon and/or plays an alert sound.

`NoteAlert` (page 52)
Displays an alert box with a note icon and/or plays an alert sound.

`CautionAlert` (page 17)
Displays an alert box with a caution icon and/or plays an alert sound.

`GetAlertStage` (page 28)
Determines the stage of the last occurrence of an alert.

`ResetAlertStage` (page 54)
Resets the current alert stage to the first alert stage.

Creating and Disposing of Dialog Boxes

`GetNewDialog` (page 36)
Creates a dialog box from a resource-based description.

`NewFeaturesDialog` (page 49)
Creates a dialog box from information passed in memory.

`NewDialog` (page 48)
Creates a dialog box from information passed in memory.

`NewColorDialog` (page 46)
Creates a dialog box from information passed in memory.

`CloseDialog` (page 18)
Dismisses a dialog box without disposing of the dialog structure.

`DisposeDialog` (page 25)
Dismisses a dialog box for which the Dialog Manager supplies memory and disposes of the dialog structure.

Displaying Dialog Boxes and Items

`DrawDialog` (page 27)
Draws the entire contents of a specified dialog box.

`HideDialogItem` (page 39)
Makes an item in a dialog box invisible.

`ShowDialogItem` (page 63)
Redisplays an item that has been hidden by HideDialogItem.

Filtering Dialog Box Events

`GetModalDialogEventMask` (page 35)
Obtains the events to be received by the `ModalDialog` function.

`SetModalDialogEventMask` (page 62)
Specifies the events to be received by the `ModalDialog` function.

Handling Events in Dialog Boxes

`ModalDialog` (page 43)
Handles events while your application displays a modal or movable modal dialog box.

`IsDialogEvent` (page 41)
Determines whether a modeless dialog box or a movable modal dialog box is active when an event occurs.

`DialogSelect` (page 23)
Handles most of the events inside the dialog box after you have determined that an event related to an active modeless dialog box or an active movable modal dialog box has occurred.

`UpdateDialog` (page 68)
Redraws the update region of a specified dialog box.

Handling Text in Alert and Dialog Boxes

`ParamText` (page 53)
Replaces the text strings in the static text items of your alert or dialog boxes while your application is running.

`SetDialogItemText` (page 60)
Sets the text string for static text and editable text fields.

`GetDialogItemText` (page 32)
Obtains the text string contained in an edit text or a static text item.

`SelectDialogItemText` (page 55)
Selects and highlights text contained in an edit text item.

`DialogCut` (page 22)
Handles the Cut editing command when a dialog box containing an edit text item is active.

`DialogCopy` (page 22)
Handles the Copy editing command when a dialog box containing an edit text item is active.

`DialogPaste` (page 23)
Handles the Paste editing command when a dialog box containing an edit text item is active.

`DialogDelete` (page 23)
Handles the Delete editing command when a dialog box containing an edit text item is active.

Initializing the Dialog Manager

`SetDialogFont` (page 58)
Sets the font used in static and edit text items.

Manipulating Items in Dialog Boxes and Alert Boxes

`SetDialogCancelItem` (page 56)
Sets the cancel item for a dialog box.

`GetDialogCancelItem` (page 29)
Returns the item number of the cancel item previously set with `SetDialogCancelItem`.

`SetDialogDefaultItem` (page 57)
Sets the default item for a dialog box and draws an appropriate border around the default item.

`GetDialogDefaultItem` (page 29)
Returns the item number of the default item currently set for the standard filter function.

`GetDialogItemAsControl` (page 31)
Obtains the control handle for a dialog item in an embedding hierarchy.

`GetDialogItem` (page 30)
Obtains a handle to a dialog item.

`SetDialogItem` (page 59)
Sets or changes information for a dialog item.

`GetDialogKeyboardFocusItem` (page 32)
Returns the item number of the editable text item in a dialog box that has keyboard focus.

`SetDialogTracksCursor` (page 61)
Determines whether the Dialog Manager tracks the cursor's movements and changes the cursor to an I-beam whenever it is over an edit dialog box.

`FindDialogItem` (page 27)
Determines the item number of an item at a particular location in a dialog box.

`MoveDialogItem` (page 45)
Moves a dialog item to a specified location in a window.

`SizeDialogItem` (page 64)
Sizes a dialog item.

`AutoSizeDialog` (page 16)
Automatically resizes static text fields and their dialog boxes to accommodate changed static text.

`AppendDialogItemList` (page 13)
Adds items to an existing dialog box while your program is running.

`AppendDITL` (page 14)
Adds items to an existing dialog box while your application is running.

`ShortenDITL` (page 63)
Removes items from an existing dialog box while your application is running.

`CountDITL` (page 19)
Determines the number of items in a dialog box.

Simulating User Responses in Dialog Boxes

`GetDialogTimeout` (page 34)
Obtains the original countdown duration, the time remaining, and the item selection to be simulated for a specified modal dialog box.

`SetDialogTimeout` (page 60)
Simulates an item selection in a modal dialog box after a specified amount of time elapses.

Using the Standard Filter Function

`StdFilterProc` (page 66)
Handles standard event filtering for a dialog box.

`GetStdFilterProc` (page 38)
Returns a pointer to the standard filter function.

Miscellaneous

`RunStandardAlert` (page 55)
Shows and runs a standard alert using a modal dialog loop.

`SetPortDialogPort` (page 62)

Functions

Alert

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

```
DialogItemIndex Alert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

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'` 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 43) function. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

Return Value

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 82). See the description of the `DialogItemIndex` data type.

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 43) 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.

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`.

See also the functions `NoteAlert` (page 52) , `CautionAlert` (page 17) , and `StopAlert` (page 67).

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`

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

Version Notes

This function was changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

AppendDialogItemList

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

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

Parameters

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.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

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
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

AppendDITL

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

```
void AppendDITL (
   DialogRef theDialog,
   Handle theHandle,
   DITLMethod method
);
```

Parameters

theDialog

A pointer to a dialog structure. This is the dialog structure to which you will add the item list resource specified in the parameter theHandle.

theHandle

A handle to the item list resource whose items you want to append to the dialog box. To avoid item number conflicts, `AppendDITL` assigns new numbers to the items you are adding. For example, if you have a dialog with item numbers 1-5, and you use `AppendDITL` to add a `'DITL'` resource containing item numbers 1-3, those become item numbers 6-8 in the dialog.

method

The manner in which you want the new items to be displayed in the existing dialog box. You can pass a negative value to offset the appended items from a particular item in the existing dialog box. You can also pass one of the values defined by the `DITLMethod` constant. See "Dialog Item List Display Constants" (page 91) for possible values.

Discussion

The `AppendDITL` function adds the items specified in the `theHandle` parameter to the items of a dialog box (handle-based). This function is especially useful if several dialog boxes share a single item list resource, because you can use `AppendDITL` 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.

In the parameter `method`, you specify how to append the new items, as follows:

- If you use the `overlayDITL` constant, `AppendDITL` superimposes the appended items over the dialog box. That is, `AppendDITL` interprets 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, `AppendDITL` 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 `AppendDITL` function automatically expands the dialog box to accommodate the new dialog items.
- If you use the `appendDITLBottom` constant, `AppendDITL` 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 `AppendDITL` function automatically expands the dialog box to accommodate the new dialog items.
- You can also append a list of items relative to an existing item by passing a negative number in the parameter `method`. 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.

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

Special Considerations

The `AppendDITL` function modifies the contents of the dialog box (for instance, by enlarging it). To use an unmodified version of the dialog box at a later time, your application should use the Resource Manager function `ReleaseResource` to release the memory occupied by the appended item list resource. Otherwise, if your application calls `AppendDITL` to add items to that dialog box again, the dialog box remains modified by your previous call—for example, it will still be longer at the bottom if you previously used the `appendDITLBottom` constant.

Before calling `AppendDITL`, you should make sure that it is available by using the `Gestalt` function with the `gestaltDITLExtAttr` selector. Test the bit indicated by the `gestaltDITLExtPresent` constant in the `response` parameter. If the bit is set, then `AppendDITL` is available.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

AutoSizeDialog

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

```
OSErr AutoSizeDialog (
   DialogRef inDialog
);
```

Parameters

`inDialog`

A pointer to a dialog box.

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

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

This function is available with Appearance Manager 1.0 and later.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Dialogs.h`

CautionAlert

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

```
DialogItemIndex CautionAlert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

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'` 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 43) function. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

Return Value
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 82). See the description of the `DialogItemIndex` data type.

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 12) 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).

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`.

See also the functions `NoteAlert` (page 52) and `StopAlert` (page 67).

Special Considerations

Version Notes
This function was changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Dialogs.h`

CloseDialog

Dismisses a dialog box without disposing of the dialog structure.

```
void CloseDialog (
    DialogRef theDialog
);
```

Parameters

theDialog
A pointer to a dialog structure.

Return Value

Discussion
The `CloseDialog` function removes a dialog box from the screen and deletes it from the window list. The `CloseDialog` function releases the memory occupied by

- the data structures associated with the dialog box (such as its structure, content, and update regions)
- all the items in the dialog box (except for pictures and icons, which might be shared by other resources) and any data structures associated with them

Generally, you should provide memory for the dialog structure of modeless dialog boxes when you create them. (You can let the Dialog Manager provide memory for modal and movable modal dialog boxes.) You should then use `CloseDialog` to close a modeless dialog box when the user clicks the close box or chooses Close from the File menu.

Because `CloseDialog` does not dispose of the dialog resource or the item list resource, it is important to make these resources purgeable. Unlike `GetNewDialog` (page 36) , `NewColorDialog` (page 46) does not use a copy of the item list resource. Thus, if you use `NewColorDialog` to create a dialog box, you may want to use `CloseDialog` to keep the item list resource in memory even if you didn't supply a pointer to the memory.

Carbon Porting Notes

The `CloseDialog` function is not supported because developers do not allocate their own memory for dialog boxes in Carbon. Use the `DisposeDialog` function to dismiss a dialog box instead.

Declared In
`Dialogs.h`

CloseStandardSheet

```
OSStatus CloseStandardSheet (
   DialogRef inSheet,
   UInt32 inResultCommand
);
```

Parameters

inSheet

inResultCommand

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

CountDITL

Determines the number of items in a dialog box.

```
DialogItemIndex CountDITL (
   DialogRef theDialog
);
```

Parameters

theDialog

A pointer to a dialog structure.

Return Value

The number of current items in a dialog box. See the description of the `DialogItemIndex` data type.

Discussion

You typically use `CountDITL` in conjunction with `ShortenDITL` (page 63) to remove items from a dialog box.

Special Considerations

Before calling `CountDITL`, you should make sure that it is available by using the `Gestalt` function with the `gestaltDITLExtAttr` selector. Test the bit indicated by the `gestaltDITLExtPresent` constant in the `response` parameter. If the bit is set, then `CountDITL` is available.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

CreateStandardAlert

Creates an alert containing standard elements and using standard formatting rules.

```
OSStatus CreateStandardAlert (
   AlertType alertType,
   CFStringRef error,
   CFStringRef explanation,
   const AlertStdCFStringAlertParamRec *param,
   DialogRef *outAlert
);
```

Parameters

alertType

The type of alert to create. For a list of possible values, see "Alert Type Constants" (page 85).

error

The error string to display.

explanation

The explanation string to display. May be `NULL` or empty to display no explanation.

param

The parameter block describing how to create the alert. May be `NULL`.

outAlert

A pointer to a variable that, on return, refers to the new alert.

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Discussion

This function should be used in conjunction with `RunStandardAlert` (page 55). After `CreateStandardAlert` returns, the alert is still invisible. `RunStandardAlert` shows the alert and runs a modal dialog loop to process events in the alert.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Dialogs.h`

CreateStandardSheet

Creates an alert containing standard elements and using standard formatting rules, and prepares it to be displayed as a sheet.

```
OSStatus CreateStandardSheet (
   AlertType alertType,
   CFStringRef error,
   CFStringRef explanation,
   const AlertStdCFStringAlertParamRec *param,
   EventTargetRef notifyTarget,
   DialogRef *outSheet
);
```

Parameters

alertType

The type of alert to create. For a list of possible values, see "Alert Type Constants" (page 85).

error

The error string to display.

explanation

The explanation string to display. May be `NULL` or empty to display no explanation.

param

The parameter block describing how to create the alert. May be `NULL`.

notifyTarget

The event target to be notified when the user dismisses the sheet. The caller should install an event handler on this target for the `kEventProcessCommand` event. May be `NULL` if the caller does not need the command event to be sent to any target. For more information, see the Discussion below.

outSheet

A pointer to a variable that, on return, refers to the new alert.

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Discussion

This function should be used in conjunction with `ShowSheetWindow`. After `CreateStandardSheet` returns, the alert is still invisible. `ShowSheetWindow` will show the alert as a sheet and then return. Events in the sheet are handled asynchronously; the application should be prepared for the sheet window to be part of its window list while running its own event loop.

When a button in the sheet is pressed, the event target passed to `CreateStandardSheet` will receive a command event with one of the following commands: `kHICommandOK`, `kHICommandCancel`, or `kHICommandOther`. The system takes care of closing the sheet and releasing the alert. Therefore after using `ShowSheetWindow`, you do not need to call `HideSheetWindow` or `DisposeDialog` (page 25).

Typically, the event target you pass in the *notifyTarget* parameter is the parent window of the sheet. A standard practice is to install a command event handler on the parent window just before showing the sheet window, and to remove the handler from the parent window after the sheet has been closed.

It is also possible to install a handler on the sheet window itself, in which case you would pass `NULL` in the *notifyTarget* parameter, since the command event is automatically sent to the sheet window already. If you install a handler on the sheet itself, make sure to return `eventNotHandledErr` from your handler, because `CreateStandardSheet` installs its own handler on the sheet and that handler must be allowed to run to close the sheet window and release the alert.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Dialogs.h`

DialogCopy

Handles the Copy editing command when a dialog box containing an edit text item is active.

```
void DialogCopy (
   DialogRef theDialog
);
```

Parameters

`theDialog`

A pointer to a dialog structure.

Discussion
The `DialogCopy` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TECopy` to the selected text. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

DialogCut

Handles the Cut editing command when a dialog box containing an edit text item is active.

```
void DialogCut (
   DialogRef theDialog
);
```

Parameters

`theDialog`

On input, a pointer to a dialog structure.

Discussion
The `DialogCut` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TECut` to the selected text. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

DialogDelete

Handles the Delete editing command when a dialog box containing an edit text item is active.

```
void DialogDelete (
   DialogRef theDialog
);
```

Parameters

theDialog

A pointer to a dialog structure.

Discussion

The `DialogDelete` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TEDelete` to the selected text. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

DialogPaste

Handles the Paste editing command when a dialog box containing an edit text item is active.

```
void DialogPaste (
   DialogRef theDialog
);
```

Parameters

theDialog

On input, a pointer to a dialog structure.

Discussion

The `DialogPaste` function checks whether the dialog box has any edit text items and, if so, applies the TextEdit function `TEPaste` to the selected edit text item. Your application should test whether a dialog box is the frontmost window when handling mouse-down events in the Edit menu and then call this function when appropriate.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

DialogSelect

Handles most of the events inside the dialog box after you have determined that an event related to an active modeless dialog box or an active movable modal dialog box has occurred.

```
Boolean DialogSelect (
   const EventRecord *theEvent,
   DialogRef *theDialog,
   DialogItemIndex *itemHit
);
```

Parameters

theEvent

A pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

theDialog

A pointer to a dialog structure for the dialog box where the event occurred.

itemHit

A pointer to a short integer. `DialogSelect` returns a number corresponding to the position of an item within the item list resource of the active dialog box.

Return Value

A Boolean value. If the event is an activate or update event for a dialog box, `DialogSelect` activates or updates it and returns `false`. If the event involves an enabled item, `DialogSelect` returns a function result of `true`.

Discussion

The `DialogSelect` function handles most of the events relating to a dialog box. Through its `itemHit` parameter, it returns the item number of the item selected by the user. Through the parameter `theDialog`, it returns a pointer to the dialog structure for the dialog box where the event occurred. In all other cases, the `DialogSelect` function returns `false`. When `DialogSelect` returns `true`, do whatever is appropriate as a response to the event involving that item in that particular dialog box; when it returns `false`, do nothing.

Generally, only controls should be enabled in a dialog box; therefore your application should normally respond only when `DialogSelect` returns `true` after the user clicks an enabled control, such as the OK button.

The `DialogSelect` function first obtains a pointer to the window containing the event. For update and activate events, the event structure contains the window pointer. For other types of events, `DialogSelect` calls the Window Manager function `FrontWindow`. The Dialog Manager then makes this window the current graphics port by calling the QuickDraw function `SetPort`. Then `DialogSelect` prepares to handle the event by setting up text information if there are any edit text items in the active dialog box.

When an item is a control defined in a control resource, the rectangle added to the update region is the rectangle defined in the control resource, not the display rectangle defined in the item list resource.

The `DialogSelect` function handles the event as follows:

- In response to an activate or update event for the dialog box, `DialogSelect` activates or updates its window and returns false.
- If a key-down event or an auto-key event occurs and there's an edit text item in the dialog box, `DialogSelect` uses `TextEdit` to handle text entry and editing, and `DialogSelect` returns true for a function result. Through its `itemHit` parameter, `DialogSelect` returns the item number.
- If a key-down event or an auto-key event occurs and there's no edit text item in the dialog box, `DialogSelect` returns `false`.
- If the user presses the mouse button while the cursor is in an edit text item, `DialogSelect` responds to the mouse activity as appropriate—that is, either by displaying an insertion point or by selecting text. If the edit text item is disabled, `DialogSelect` returns `false`. If the edit text item is enabled,

`DialogSelect` returns `true` and through its `itemHit` parameter returns the item number. Normally, edit text items are disabled, and you use the `GetDialogItemText` function to read the information in the items only after the OK button is clicked.

- If the user presses the mouse button while the cursor is in a control, `DialogSelect` tracks the control. If the user releases the mouse button while the cursor is in an enabled control, `DialogSelect` returns `true` for a function result and through its `itemHit` parameter returns 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, `DialogSelect` returns `true` for a function result and through its `itemHit` parameter returns the item's number. Generally, only controls should be enabled. If your application creates a complex control—such as one that measures how far a dial is moved—your application must handle mouse events in that item before passing the event to `DialogSelect`.
- If the user presses the mouse button while the cursor is in a disabled item, or if it is in no item, or if any other event occurs, `DialogSelect` does nothing.
- If the event isn't one that `DialogSelect` specifically checks for (if it's a null event, for example), and if there's an edit text item in the dialog box, `DialogSelect` calls the TextEdit function `TEIdle` to make the insertion point blink.

Special Considerations

Because `DialogSelect` handles only mouse-down events in a dialog box and key-down events in a dialog box's edit text items, you should handle other events as appropriate before passing them to `DialogSelect`. Likewise, when `DialogSelect` calls the Control Manager function `TrackControl` , it does not allow you to specify any action function necessary for anything more complex than a button, radio button, or checkbox. If you need a more complex control (for example, one that measures how long the user holds down the mouse button or how far the user has moved an indicator), you can create your own control or a picture or an application-defined item that draws a control-like object in your dialog box. You must then test for and respond to those events yourself.

Within dialog boxes, use the functions `DialogCut` (page 22), `DialogCopy` (page 22), `DialogPaste` (page 23), and `DialogDelete` (page 23) to support Cut, Copy, Paste, and Clear commands in edit text boxes.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

DisposeDialog

Dismisses a dialog box for which the Dialog Manager supplies memory and disposes of the dialog structure.

```
void DisposeDialog (
   DialogRef theDialog
);
```

Parameters

`theDialog`

A pointer to a dialog structure.

Return Value

Discussion
The `DisposeDialog` function calls `CloseDialog` (page 18) and, in addition, releases the memory occupied by the dialog box's item list resource and the dialog structure. Call `DisposeDialog` when you're done with a dialog box if you pass `null` in the `dStorage` parameter to `GetNewDialog` (page 36) , `NewColorDialog` (page 46) , or `NewDialog` (page 48).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
ictbSample

Declared In
`Dialogs.h`

DisposeModalFilterUPP

```
void DisposeModalFilterUPP (
   ModalFilterUPP userUPP
);
```

Parameters
userUPP

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DisposeModalFilterYDUPP

```
void DisposeModalFilterYDUPP (
   ModalFilterYDUPP userUPP
);
```

Parameters
userUPP

Carbon Porting Notes
This function is supported in Carbon because several QuickTime routines require it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DisposeUserItemUPP

```
void DisposeUserItemUPP (
   UserItemUPP userUPP
);
```

Parameters

userUPP

Availability
Available in Mac OS X v10.0 and later.

Declared In
Dialogs.h

DrawDialog

Draws the entire contents of a specified dialog box.

```
void DrawDialog (
   DialogRef theDialog
);
```

Parameters

theDialog

A pointer to a dialog structure.

Return Value

Discussion
The DrawDialog function draws all dialog items, calls the Control Manager function DrawOneControl to draw all controls, and calls the TextEdit function TEUpdate to update all static and edit text items and to draw their display rectangles. The DrawDialog function also calls the application-defined items' draw functions if the items' rectangles are within the update region.

DialogSelect (page 23) , ModalDialog (page 43) , Alert (page 12) , StopAlert (page 67) , NoteAlert (page 52) , and CautionAlert (page 17) use DrawDialog automatically. If you use GetNewDialog (page 36) to create a dialog box but don't use any of these other Dialog Manager functions when handling events in the dialog box, you can use DrawDialog to redraw the contents of the dialog box when it's visible. If the dialog box is invisible, first use the Window Manager function ShowWindow and then use DrawDialog.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Dialogs.h

FindDialogItem

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

```
DialogItemIndexZeroBased FindDialogItem (
   DialogRef theDialog,
   Point thePt
);
```

Parameters

theDialog

A pointer to a dialog structure.

thePt

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

Return Value

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. See the description of the `DialogItemIndexZeroBased` data type.

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 30) function or the `SetDialogItem` (page 59) 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

This function was changed with Appearance Manager 1.0 to support embedding hierarchies.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

GetAlertStage

Determines the stage of the last occurrence of an alert.

```
SInt16 GetAlertStage (
   void
);
```

Parameters

Return Value

A number from 0 to 3 as the stage of the last occurrence of an alert.

Discussion

You can use the `GetAlertStage` function to ensure that your application deactivates the active window only if an alert box is to be displayed at that stage.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogCancelItem

Returns the item number of the cancel item previously set with `SetDialogCancelItem`.

```
SInt16 GetDialogCancelItem (
   DialogRef dialog
);
```

Parameters

dialog
On input, a pointer to the dialog structure for the dialog box whose cancel item you want to get.

Return Value
The item number of the cancel item previously set with the `SetDialogCancelItem` (page 56) function.

Discussion
If you don't explicitly call `GetDialogCanceltItem`, the standard filter function treats item 2 as the cancel item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogDefaultItem

Returns the item number of the default item currently set for the standard filter function.

```
SInt16 GetDialogDefaultItem (
   DialogRef dialog
);
```

Parameters

dialog
On input, a pointer to the dialog structure for the dialog box whose default item you want to get.

Return Value
The item number of the default item currently set for the standard filter function.

Discussion
If you don't explicitly call `GetDialogDefaultItem`, the standard filter function treats item 1 as the default item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogFromWindow

```
DialogRef GetDialogFromWindow (
   WindowRef window
);
```

Parameters

window

Return Value
See the description of the `DialogRef` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogItem

Obtains a handle to a dialog item.

```
void GetDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   DialogItemType *itemType,
   Handle *item,
   Rect *box
);
```

Parameters

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 27) 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.

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.

Return Value

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 31).

You should call `GetDialogItem` before calling functions such as `SetDialogItemText` (page 60) that need a handle to a dialog item.

See also the function `SetDialogItem` (page 59).

Version Notes

This function was changed with Appearance Manager 1.0 to support retrieving item information from controls.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

ictbSample

Declared In

`Dialogs.h`

GetDialogItemAsControl

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

```
OSErr GetDialogItemAsControl (
   DialogRef inDialog,
   DialogItemIndex inItemNo,
   ControlRef *outControl
);
```

Parameters

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.

Return Value

A result code. See "Dialog Manager Result Codes" (page 94). 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 30) when an embedding hierarchy is established.

Version Notes
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogItemText

Obtains the text string contained in an edit text or a static text item.

```
void GetDialogItemText (
   Handle item,
   Str255 text
);
```

Parameters

item
On input, a handle to an edit text or a static text item. To get this handle, call the "Alert Button Constants" (page 82) function.

text
On output, a string containing the text of the item that is specified by the `item` parameter.

Discussion
The `GetDialogItemText` function will only return the first 255 characters in an edit text item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogKeyboardFocusItem

Returns the item number of the editable text item in a dialog box that has keyboard focus.

```
SInt16 GetDialogKeyboardFocusItem (
   DialogRef dialog
);
```

Parameters

dialog
On input, a pointer to the dialog structure for the dialog box whose currently focused item you want to identify.

Return Value
The number of the editable text item in a dialog box that currently has keyboard focus.

Discussion
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.

The `GetDialogKeyboardFocusItem` function accesses the `edit` field in the dialog structure. `GetDialogKeyboardFocusItem` should only be called when there is no embedding hierarchy in the dialog box.

Version Notes
This function is not recommended with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogPort

```
CGrafPtr GetDialogPort (
   DialogRef dialog
);
```

Parameters
dialog

Return Value
See the QuickDraw Manager documentation for a description of the `CGrafPtr` data type.

Discussion

Special Considerations

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogTextEditHandle

```
TEHandle GetDialogTextEditHandle (
   DialogRef dialog
);
```

Parameters

dialog

Return Value
See the TextEdit documentation for a description of the `TEHandle` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogTimeout

Obtains the original countdown duration, the time remaining, and the item selection to be simulated for a specified modal dialog box.

```
OSStatus GetDialogTimeout (
   DialogRef inDialog,
   DialogItemIndex *outButtonToPress,
   UInt32 *outSecondsToWait,
   UInt32 *outSecondsRemaining
);
```

Parameters

inDialog

A pointer to the dialog box to be examined.

outButtonToPress

On input, a pointer to a signed 16-bit integer. On return, a value representing the number within the item list of the item that is to be selected. You may pass `NULL` for the `outButtonToPress` parameter if you do not desire this information.

outSecondsToWait

On input, a pointer to an unsigned 32-bit integer. On return, a value specifying the number of seconds that were originally set to elapse before the Dialog Manager simulates an item selection. You may pass `NULL` for the `outSecondsToWait` parameter if you do not desire this information.

outSecondsRemaining

On input, a pointer to an unsigned 32-bit integer. On return, a value specifying the number of seconds remaining before the Dialog Manager simulates an item selection. You may pass `NULL` for the `outSecondsRemaining` parameter if you do not desire this information.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

Discussion
Also see the function `SetDialogTimeout` (page 60).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetDialogWindow

```
WindowRef GetDialogWindow (
   DialogRef dialog
);
```

Parameters
dialog

Return Value
See the QuickDraw Manager documentation for a description of the `WindowRef` data type.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Dialogs.h`

GetModalDialogEventMask

Obtains the events to be received by the `ModalDialog` function.

```
OSStatus GetModalDialogEventMask (
   DialogRef inDialog,
   EventMask *outMask
);
```

Parameters
inDialog
A pointer to the dialog box for which you wish to obtain the event mask.
outMask
On input, a pointer to a unsigned 16-bit integer of type `EventMask`. On return, your application may test the bits of this value to determine the event(s) that the dialog box is currently set to receive.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

Discussion
Also see the function `SetModalDialogEventMask` (page 62).

Version Notes
This function is available with Mac OS 8.5 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetNewDialog

Creates a dialog box from a resource-based description.

```
DialogRef GetNewDialog (
   SInt16 dialogID,
   void *dStorage,
   WindowRef behind
);
```

Parameters

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`’ and ‘`dlgx`’ 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 `null`, 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.

Return Value
Returns a pointer to a dialog box. If none was created, returns `null`. See the description of the `DialogRef` data type.

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 `null`; your application should test to ensure that `null` 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`

Version Notes
This function was changed with Appearance Manager 1.0 to support the extended dialog (`'dlgx'`) resource and the dialog font table (`'dftb'`) resource.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
ictbSample

Declared In
`Dialogs.h`

GetParamText

```
void GetParamText (
   StringPtr param0,
   StringPtr param1,
   StringPtr param2,
   StringPtr param3
);
```

Parameters

param0

param1

param2

param3

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

GetStandardAlertDefaultParams

Fills out an `AlertStdCFStringAlertParamRec` with default values: - not movable - no help button - default button with title "OK" - no cancel or other buttons.

```
OSStatus GetStandardAlertDefaultParams (
   AlertStdCFStringAlertParamPtr param,
   UInt32 version
);
```

Parameters

param

The parameter block to initialize.

version

The parameter block version; pass kStdCFStringAlertVersionOne.

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code

QTCarbonShell

Declared In

`Dialogs.h`

GetStdFilterProc

Returns a pointer to the standard filter function.

```
OSErr GetStdFilterProc (
   ModalFilterUPP *theProc
);
```

Parameters

theProc

A universal procedure pointer to a filter function. On output, the Dialog Manager provides a pointer to its standard filter function.

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Discussion

The `GetStdFilterProc` function gets a pointer to the standard filter function. You must dispatch the function yourself using the CallModalFilterProc macro; see `ModalFilterProcPtr` (page 69).

You normally don't need to use `GetStdFilterProc` unless your development environment doesn't include the code required to support `StdFilterProc` (page 66).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Dialogs.h

HideDialogItem

Makes an item in a dialog box invisible.

```
void HideDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo
);
```

Parameters

theDialog

A pointer to a dialog structure.

itemNo

A number corresponding to the position of an item in the dialog box's item list resource.

Return Value

Discussion

The HideDialogItem function hides the item specified by itemNo by giving it a display rectangle that's off the screen. Specifically, if the left coordinate of the item's display rectangle is less than 8192 (hexadecimal 0x2000), HideDialogItem adds 16,384 (hexadecimal 0x4000) to both the left and right coordinates of the rectangle. If the item is already hidden (that is, if the left coordinate is greater than 8192), HideDialogItem does nothing. To redisplay an item that's been hidden by HideDialogItem, you can use the ShowDialogItem function.

Special Considerations

If your application needs to display a number of dialog boxes that are similar except for one or two items, it's generally easier to modify the common elements using the AppendDITL (page 14) and ShortenDITL (page 63) functions than to use the HideDialogItem and ShowDialogItem (page 63) functions.

If you hid an edit text item, the next visible edit text item will be highlighted.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Dialogs.h

InsertDialogItem

```
OSStatus InsertDialogItem (
   DialogRef theDialog,
   DialogItemIndex afterItem,
   DialogItemType itemType,
   Handle itemHandle,
   const Rect *box
);
```

Parameters

theDialog

afterItem

itemType

itemHandle

box

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

Dialogs.h

InvokeModalFilterUPP

```
Boolean InvokeModalFilterUPP (
   DialogRef theDialog,
   EventRecord *theEvent,
   DialogItemIndex *itemHit,
   ModalFilterUPP userUPP
);
```

Parameters

theDialog

theEvent

itemHit

userUPP

Availability

Available in Mac OS X v10.0 and later.

Declared In

Dialogs.h

InvokeModalFilterYDUPP

```
Boolean InvokeModalFilterYDUPP (
   DialogRef theDialog,
   EventRecord *theEvent,
   short *itemHit,
   void *yourDataPtr,
   ModalFilterYDUPP userUPP
);
```

Parameters

theDialog

theEvent

itemHit

yourDataPtr

userUPP

Carbon Porting Notes

This function is supported in Carbon because several QuickTime routines require it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
Dialogs.h

InvokeUserItemUPP

```
void InvokeUserItemUPP (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   UserItemUPP userUPP
);
```

Parameters

theDialog

itemNo

userUPP

Availability
Available in Mac OS X v10.0 and later.

Declared In
Dialogs.h

IsDialogEvent

Determines whether a modeless dialog box or a movable modal dialog box is active when an event occurs.

```
Boolean IsDialogEvent (
   const EventRecord *theEvent
);
```

Parameters

theEvent

A pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

Return Value

A Boolean value. If any event, including a null event, occurs when your dialog box is active, `IsDialogEvent` returns `true`; otherwise, it returns `false`.

Discussion

When `IsDialogEvent` returns `false`, pass the event to the rest of your event-handling code. When `IsDialogEvent` returns `true`, pass the event to `DialogSelect` (page 23) after testing for the events that `DialogSelect` does not handle.

A dialog structure includes a window structure. When you use the `GetNewDialog` (page 36) , `NewDialog` (page 48) , `NewFeaturesDialog` (page 49) , or `NewColorDialog` (page 46) functions to create a dialog box, the Dialog Manager sets the `windowKind` field in the window structure to `dialogKind`. To determine whether the active window is a dialog box, `IsDialogEvent` checks the `windowKind` field.

Before passing the event to `DialogSelect`, you should perform the following tests whenever `IsDialogEvent` returns `true` :

- Check whether the event is a key-down event for the Return, Enter, Esc, or Command-period keystrokes. When the user presses the Return or Enter key, your application should respond as if the user had clicked the default button; when the user presses Esc or Command-period, your application should respond as if the user had clicked the Cancel button. Use the Control Manager function `HiliteControl` to highlight the applicable button for 8 ticks.
- At this point, you may also want to check for and respond to any special events that you do not wish to pass to `DialogSelect` (page 23) or that require special processing before you pass them to `DialogSelect`. You would need to do this, for example, if the dialog box needs to respond to disk-inserted events.
- Check whether the event is an update event for a window other than the dialog box and, if it is, update your window.
- For complex items that you create, such as pictures or application-defined items that emulate complex controls, test for and respond to mouse events inside those items as appropriate. When `DialogSelect` 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 (for example, one that measures how long the user holds down the mouse button or how far the user has moved an indicator), you can create your own control or a picture or an application-defined item that draws a control-like object in your dialog box. You must then test for and respond to those events yourself.

If your application uses `IsDialogEvent` to help handle events when you display a movable modal dialog box, perform the following additional tests before passing events to `DialogSelect` :

- Test for mouse-down events in the title bar of the movable modal dialog box and respond by dragging the dialog box accordingly.

- Test for and respond to mouse-down events in the Apple menu and, if the movable modal dialog box includes edit text items, in the Edit menu. (You should disable all other menus when you display a movable modal dialog box.)
- Play the system alert sound for every other mouse-down event outside the movable modal dialog box.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

ModalDialog

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

```
void ModalDialog (
   ModalFilterUPP modalFilter,
   DialogItemIndex *itemHit
);
```

Parameters

`modalFilter`

A universal procedure pointer for an event filter function. For modal dialog boxes, you can specify `null` 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.

Return Value

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 `null`, 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
This function was changed with Appearance Manager 1.0 to handle events for movable modal dialogs.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
ictbSample

Declared In
`Dialogs.h`

MoveDialogItem

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

```
OSErr MoveDialogItem (
   DialogRef inDialog,
   DialogItemIndex inItemNo,
   SInt16 inHoriz,
   SInt16 inVert
);
```

Parameters

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 27) to determine this value.

inHoriz
The new horizontal coordinate for the dialog item.

inVert
The new vertical coordinate for the dialog item.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

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
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

NewColorDialog

Creates a dialog box from information passed in memory.

```
DialogRef NewColorDialog (
   void *dStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   SInt16 procID,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon,
   Handle items
);
```

Parameters

dStorage

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

boundsRect

On input, 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.

title

On input, a text string used for the title of a modeless or movable modal dialog box. You can specify an empty string (not `null`) for a title bar that contains no text.

visible

On input, 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.

procID

On input, 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.

behind

On input, 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.

goAwayFlag

On input, a flag to specify whether a modeless dialog box can have a close box in its title bar when the dialog box is active. If you set this parameter to `true`, the modeless dialog box has a close box in its title bar when the window is active.

refCon

On input, 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.

items

On input, 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. Use the Memory Manager function `HNoPurge` to make the handle unpurgeable while you use it or use the Operating System utility function `HandToHand` to make a copy of the handle and use the copy.

Return Value

A pointer to the new dialog box. If the function doesn't create a new dialog box, returns `null`. See the description of the `DialogRef` data type.

Discussion

The `NewColorDialog` function creates a dialog box as specified by its parameters. The first eight parameters (`dStorage` through `refCon`) are passed to the Window Manager function `NewWindow`, which creates the dialog box. You can use this pointer with Window Manager or QuickDraw functions to manipulate the dialog box.

The Dialog Manager uses the default window colors for the dialog box. By using the system's default colors, you ensure that your application's interface is consistent with that of the Finder and other applications. However, if you absolutely feel compelled to break from this consistency, you can use the Window Manager function `SetWinColor` to use your own dialog color table resource that specifies colors other than the default colors. Be aware, however, that nonstandard colors in your alert and dialog boxes may initially confuse your users.

The Window Manager creates an auxiliary window structure for the color dialog box. You can access this structure with the Window Manager function `GetAuxWin`. If the dialog box's content color isn't white, it's a good idea to call `NewColorDialog` with the `visible` flag set to `false`. After the color table and color item list resource are installed, use the Window Manager function `ShowWindow` to display the dialog box if it's the frontmost window. If the dialog box is a modeless dialog box that is not in front, use the Window Manager function `ShowHide` to display it.

The `NewColorDialog` function generates an update event for the entire window contents. Thus, with the exception of controls, items aren't drawn immediately. The Dialog Manager calls the Control Manager to draw controls, and the Control Manager draws them immediately. So that the controls won't be drawn twice, the Dialog Manager calls the Window Manager function `ValidRect` for the enclosing rectangle of each control. If you find that there is too great a lag between the drawing of controls and the drawing of other items, try making the dialog box initially invisible and then calling the Window Manager function `ShowWindow` to show it.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

NewDialog

Creates a dialog box from information passed in memory.

```
DialogRef NewDialog (
   void *dStorage,
   const Rect *boundsRect,
   ConstStr255Param title,
   Boolean visible,
   SInt16 procID,
   WindowRef behind,
   Boolean goAwayFlag,
   SRefCon refCon,
   Handle items
);
```

Parameters

dStorage

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

boundsRect

On input, 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.

title

On input, a text string used for the title of a modeless or movable modal dialog box. You can specify an empty string (not `null`) for a title bar that contains no text.

visible

On input, 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.

procID

On input, 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.

behind

On input, 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.

goAwayFlag

On input, a flag to specify whether a modeless dialog box can have a close box in its title bar when the dialog box is active. If you set this parameter to `true`, the modeless dialog box has a close box in its title bar when the window is active.

refCon

On input, 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.

items

On input, 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. Use the Memory Manager function `HNoPurge` to make the handle unpurgeable while you use it or use the Operating System utility function `HandToHand` to make a copy of the handle and use the copy.

Return Value

A pointer to the new dialog box. If the function doesn't create a new dialog box, returns `null`. See the description of the `DialogRef` data type.

Discussion

The `NewDialog` function is identical to the `NewColorDialog` function, except that `NewDialog` returns a pointer to a black-and-white graphics port. See the discussion of `NewColorDialog` (page 46) for descriptions of the parameters that you also pass to `NewDialog`.

The `NewDialog` function creates a dialog box as specified by its parameters and returns a pointer to a black-and-white graphics port for the new dialog box. The first eight parameters (`wStorage` through `refCon`) are passed to the Window Manager function `NewWindow`, which creates the dialog box.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

NewFeaturesDialog

Creates a dialog box from information passed in memory.

```
DialogRef NewFeaturesDialog (
   void *inStorage,
   const Rect *inBoundsRect,
   ConstStr255Param inTitle,
   Boolean inIsVisible,
   SInt16 inProcID,
   WindowRef inBehind,
   Boolean inGoAwayFlag,
   SRefCon inRefCon,
   Handle inItemListHandle,
   UInt32 inFlags
);
```

Parameters

inStorage

A pointer to the memory for the dialog box. If you set this parameter to `null`, 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 `null`) 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 87). To establish an embedding hierarchy in a dialog box, pass `kDialogFlagsUseControlHierarchy` in the `inFlags` parameter.

Return Value

A pointer to the newly created dialog box. If `NewFeaturesDialog` doesn't create a new dialog box, it returns `null`. See the description of the `DialogRef` data type.

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

This function is available with Appearance Manager 1.0 and later.

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

NewModalFilterUPP

```
ModalFilterUPP NewModalFilterUPP (
   ModalFilterProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ModalFilterUPP` data type.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

NewModalFilterYDUPP

```
ModalFilterYDUPP NewModalFilterYDUPP (
   ModalFilterYDProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `ModalFilterYDUPP` data type.

Carbon Porting Notes

This function is supported in Carbon because several QuickTime routines require it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

NewUserItemUPP

```
UserItemUPP NewUserItemUPP (
   UserItemProcPtr userRoutine
);
```

Parameters

userRoutine

Return Value
See the description of the `UserItemUPP` data type.

Discussion

Special Considerations

Version Notes

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

NoteAlert

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

```
DialogItemIndex NoteAlert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

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'` 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 43) function. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

Return Value
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 82). See the description of the `DialogItemIndex` data type.

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 12) 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.

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`.

See also the functions `CautionAlert` (page 17) and `StopAlert` (page 67).

Version Notes
This function was changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

ParamText

Replaces the text strings in the static text items of your alert or dialog boxes while your application is running.

```
void ParamText (
   ConstStr255Param param0,
   ConstStr255Param param1,
   ConstStr255Param param2,
   ConstStr255Param param3
);
```

Parameters

param0
A text string to substitute for the special string `^0` in the static text items of all subsequently created alert and dialog boxes.

param1
A text string to substitute for the special string `^1` in the static text items of all subsequently created alert and dialog boxes.

param2
A text string to substitute for the special string `^2` in the static text items of all subsequently created alert and dialog boxes.

param3
A text string to substitute for the special string `^3` in the static text items of all subsequently created alert and dialog boxes.

Discussion
The `ParamText` function replaces the special strings `^0` through `^3` in the static text items of all subsequently created alert and dialog boxes with the text strings you pass as parameters. Pass empty strings (not `null`) for parameters not used.

Special Considerations

If the user launches a desk accessory (such as a driver) in your application's partition and the desk accessory calls `ParamText`, it may change the text in your application's dialog box.

You should be very careful about using `ParamText` in modeless dialog boxes. If a modeless dialog box using `ParamText` is onscreen and you display another dialog box or alert box that also uses `ParamText`, both boxes will be affected by the latest call to `ParamText`.

Note that you should try to store text strings in resource files to facilitate translation into other languages; therefore, `ParamText` is best used for supplying text strings, such as document names, that the user specifies. To avoid problems with grammar and sentence structure when you localize your application, you should use `ParamText` to supply only one text string per screen message.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

RemoveDialogItems

```
OSStatus RemoveDialogItems (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   DialogItemIndex amountToRemove,
   Boolean disposeItemData
);
```

Parameters
theDialog
itemNo
amountToRemove
disposeItemData

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

ResetAlertStage

Resets the current alert stage to the first alert stage.

```
void ResetAlertStage (
   void
);
```

Parameters

Return Value

Discussion
The `ResetAlertStage` function resets every alert to a first-stage alert.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
`Dialogs.h`

RunStandardAlert

Shows and runs a standard alert using a modal dialog loop.

```
OSStatus RunStandardAlert (
   DialogRef inAlert,
   ModalFilterUPP filterProc,
   DialogItemIndex *outItemHit
);
```

Parameters

inAlert

The alert to display. On return, the alert you pass in this parameter has been released and is no longer valid. You should not call `DisposeDialog` (page 25) on this alert.

filterProc

An event filter function for handling events that do not apply to the alert. May be `NULL`.

outItemHit

On exit, contains the item index of the button that was pressed to close the alert.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

Discussion
This function displays and runs an alert created by `CreateStandardAlert` (page 20). `RunStandardAlert` handles all user interaction with the alert.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Related Sample Code
QTCarbonShell

Declared In
`Dialogs.h`

SelectDialogItemText

Selects and highlights text contained in an edit text item.

```
void SelectDialogItemText (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   SInt16 strtSel,
   SInt16 endSel
);
```

Parameters

theDialog

On input, a pointer to a dialog structure.

itemNo

On input, a number corresponding to the position of an edit text item in the dialog box's item list resource.

strtSel

On input, a number representing the position of the first character to begin selecting.

endSel

On input, a number representing one position past the last character to be selected.

Discussion

If the item in the `itemNo` parameter is an edit text item that contains text, the `SelectDialogItemText` function sets the text selection range to extend from the character position specified in the `strtSel` parameter up to but not including the character position specified in the `endSel` parameter. The selection range is highlighted unless `strtSel` equals `endSel`, in which case a blinking vertical bar is displayed to indicate an insertion point at that position. If the edit text item doesn't contain text, `SelectDialogItemText` displays the insertion point.

You can select the entire text by specifying the number 0 in the `strtSel` parameter and the number 32767 in the `endSel` parameter.

For example, if the user makes an unacceptable entry in the edit text item, your application can display an alert box reporting the problem and then use `SelectDialogItemText` to select the entire text so it can be replaced by a new entry. Without this function, the user would have to select the item before making the new entry.

Carbon Porting Notes

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

SetDialogCancelItem

Sets the cancel item for a dialog box.

```
OSErr SetDialogCancelItem (
   DialogRef theDialog,
   DialogItemIndex newItem
);
```

Parameters

theDialog

On input, a pointer to the dialog structure for the dialog box whose cancel item you want to set.

newItem

On input, the item number of the item you want to set as the cancel item; see "Alert Button Constants" (page 82).

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Discussion

If you intend to use the standard filter function, you can first use the functions `SetDialogDefaultItem` and `SetDialogCancelItem` to set the items that the standard filter function will treat as the default and cancel items. You can use `GetDialogDefaultItem` and `GetDialogCancelItem` to determine the dialog item numbers that the standard filter function will treat as the default and cancel items.

If you call the `SetDialogCancelItem` function before you call the standard filter function, the standard filter function automatically interprets Escape and Command-period keypresses to mean that the specified cancel item has been selected.

If you don't explicitly call `SetDialogCancelItem`, the standard filter function treats item 2 as the cancel item.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

SetDialogDefaultItem

Sets the default item for a dialog box and draws an appropriate border around the default item.

```
OSErr SetDialogDefaultItem (
   DialogRef theDialog,
   DialogItemIndex newItem
);
```

Parameters

theDialog

On input, a pointer to the dialog structure for the dialog box whose default item you want to set.

newItem

On input, the item number of the item you want to set as the default item.

Return Value

A result code. See "Dialog Manager Result Codes" (page 94).

Discussion
If you call the `SetDialogDefaultItem` function before you call the standard filter function, the standard filter function automatically interprets Return and Enter keypresses to mean that the specified default item has been selected.

If you don't explicitly call `SetDialogDefaultItem`, the standard filter function treats item 1 as the default item.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetDialogFont

Sets the font used in static and edit text items.

```
void SetDialogFont (
   SInt16 fontNum
);
```

Parameters

`fontNum`

A font ID number. Do not rely on font number constants. Instead, use the Font Manager function `GetFNum` to find the font number to pass in this parameter.

Discussion
For subsequently created dialog and alert boxes, `SetDialogFont` sets the font of the dialog or alert box's graphics port to the specified font. If you don't call this function, the system font is used. The `SetDialogFont` function does not affect titles of controls, which are always displayed in the system font.

Special Considerations

There are a number of caveats regarding the SetDialogFont function:

1. Most importantly, your application will be much easier to localize if you always use the system font in your alert and dialog boxes and never use `SetDialogFont`.
2. The Standard File Package does not always properly calculate the position of the standard file dialog box once this function has been called; for example, the standard file dialog box may be partially obscured by a menu bar.
3. Be aware that this function affects all static text and edit text items in all of the alert and dialog boxes you subsequently display.
4. `SetDialogFont` does not change the font for control titles.
5. You can't use `SetDialogFont` to change the font size or font style.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Dialogs.h

SetDialogItem

Sets or changes information for a dialog item.

```
void SetDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo,
   DialogItemType itemType,
   Handle item,
   const Rect *box
);
```

Parameters

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 27) to determine this value.

itemType

A short value. Pass an item type constant identifying the dialog item specified in the itemNo parameter. 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.

Return Value

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 the function GetDialogItem (page 30).

Version Notes
This function was changed with Appearance Manager 1.0 to work with embedding hierarchies.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Dialogs.h

SetDialogItemText

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

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

Parameters

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 . If embedding is not on, pass in the handle produced by the "Alert Button Constants" (page 82) 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

This function was changed with Appearance Manager 1.0 to support embedding hierarchies.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

SetDialogTimeout

Simulates an item selection in a modal dialog box after a specified amount of time elapses.

```
OSStatus SetDialogTimeout (
   DialogRef inDialog,
   DialogItemIndex inButtonToPress,
   UInt32 inSecondsToWait
);
```

Parameters

inDialog

A pointer to the dialog box for which an item selection is to be simulated.

inButtonToPress

A signed 16-bit integer. Pass a value representing the number (within the item list) of the item that is to be selected.

inSecondsToWait

An unsigned 32-bit integer. Pass a value specifying the number of seconds that are to elapse before the Dialog Manager simulates an item selection. Pass 0 to clear a preexisting timeout value and cease the countdown in progress.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

Discussion
Your application calls the `SetDialogTimeout` function each time you wish to start a countdown of the specified duration for a given modal dialog box. When the amount of time specified in the `inSecondsToWait` parameter has elapsed, the Dialog Manager simulates a click on the button specified in the `inButtonToPress` parameter. If your application calls `SetDialogTimeout` again, or if any event is received for the dialog box, the countdown is restarted.

In order to use `SetDialogTimeout` with a given modal dialog box, your application must handle events for the dialog box through the `ModalDialog` function. The Dialog Manager will not simulate an item selection for the dialog box until `ModalDialog` processes an event (including null events).

Also see the function `GetDialogTimeout` (page 34).

Version Notes
This function is available with Mac OS 8.5 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetDialogTracksCursor

Determines whether the Dialog Manager tracks the cursor's movements and changes the cursor to an I-beam whenever it is over an edit dialog box.

```
OSErr SetDialogTracksCursor (
   DialogRef theDialog,
   Boolean tracks
);
```

Parameters

`theDialog`
On input, a pointer to the dialog structure for the dialog box containing one or more edit text items for which you want the Dialog Manager to track the cursor.

`tracks`
On input, a Boolean value. A value of `true` indicates you want the Dialog Manager to track the cursor's movements and change it to an I-beam whenever the cursor is over an edit dialog box a value of `false` indicates you don't want the Dialog Manager to track the cursor in this manner.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

Discussion
You should call SetDialogTracksCursor before you call the standard filter function.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetModalDialogEventMask

Specifies the events to be received by the `ModalDialog` function.

```
OSStatus SetModalDialogEventMask (
   DialogRef inDialog,
   EventMask inMask
);
```

Parameters

inDialog

A pointer to the dialog box for which you wish to set the event mask.

inMask

The desired mask value(s) for the event(s) you wish the dialog box to receive.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

Discussion
Your application can use the `SetModalDialogEventMask` function to specify the events received by the `ModalDialog` function for a given modal dialog box. This allows your application to specify additional events that are not by default received by `ModalDialog`, such as disk-inserted events and operating-system events. If you use `SetModalDialogEventMask` to change the `ModalDialog` function's event mask, you should pass `ModalDialog` a pointer to your own event filter function to handle any added events.

Also see the function `GetModalDialogEventMask` (page 35).

Version Notes
This function is available with Mac OS 8.5 and later.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

SetPortDialogPort

```
void SetPortDialogPort (
   DialogRef dialog
);
```

Parameters

dialog

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In
Dialogs.h

ShortenDITL

Removes items from an existing dialog box while your application is running.

```
void ShortenDITL (
   DialogRef theDialog,
   DialogItemIndex numberItems
);
```

Parameters

theDialog

A pointer to a dialog structure.

numberItems

The number of items to remove (starting from the last item in the item list resource).

Discussion
The ShortenDITL function removes the specified number of items from the dialog box. This function is especially useful if several dialog boxes share a single item list resource, because you can use ShortenDITL to remove items as necessary for individual dialog boxes.

You typically create an invisible dialog box, call the ShortenDITL function, then make the dialog box visible by using the Window Manager function ShowWindow. Note that ShortenDITL does not automatically resize the dialog box; you can use AutoSizeDialog (page 16) or the Window Manager function SizeWindow if you need to resize the dialog box.

Special Considerations

The ShortenDITL function is available in System 7 and in earlier versions of the Communications Toolbox. Before calling ShortenDITL, you should make sure that it is available by using the Gestalt function with the gestaltDITLExtAttr selector. Test the bit indicated by the gestaltDITLExtPresent constant in the response parameter. If the bit is set, then ShortenDITL is available.

Carbon Porting Notes

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
Dialogs.h

ShowDialogItem

Redisplays an item that has been hidden by HideDialogItem.

```
void ShowDialogItem (
   DialogRef theDialog,
   DialogItemIndex itemNo
);
```

Parameters

theDialog

On input, a pointer to a dialog structure.

itemNo

On input, a number corresponding to the position of an item in the dialog box's item list resource.

Return Value

Discussion

The `ShowDialogItem` function redisplays the item specified in the `itemNo` parameter by restoring the display rectangle the item had prior to `HideDialogItem` (page 39). If the left coordinate of the item's display rectangle is greater than 8192, `ShowDialogItem` subtracts 16,384 from both the left and right coordinates of the rectangle. If the item is already visible (that is, if the left coordinate is less than 8192), `ShowDialogItem` does nothing.

The `ShowDialogItem` function adds the rectangle that contained the item to the update region so that it will be drawn. Note that if the item is a control you define in a control (' `CNTL` ') resource, the rectangle added to the update region is the rectangle defined in the control resource, not the display rectangle defined in the item list resource. If the item is an edit text item, `ShowDialogItem` activates it by calling the TextEdit function `TEActivate`.

Availability

Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Declared In

`Dialogs.h`

SizeDialogItem

Sizes a dialog item.

```
OSErr SizeDialogItem (
   DialogRef inDialog,
   DialogItemIndex inItemNo,
   SInt16 inWidth,
   SInt16 inHeight
);
```

Parameters

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 27) 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.

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

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
This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

StandardAlert

Displays a standard alert box.

```
OSErr StandardAlert (
   AlertType inAlertType,
   ConstStr255Param inError,
   ConstStr255Param inExplanation,
   const AlertStdAlertParamRec *inAlertParam,
   SInt16 *outItemHit
);
```

Parameters

inAlertType

A constant indicating the type of alert box you wish to create; see "Alert Type Constants" (page 85).

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 `null` to indicate no secondary text.

inAlertParam

A pointer to the standard alert structure; see `AlertStdAlertParamRec` (page 75). Pass `null` 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 a signed 16-bit integer value. On return, the value indicates the alert button pressed; see "Alert Button Constants" (page 82).

Return Value
A result code. See "Dialog Manager Result Codes" (page 94).

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.

Special Considerations

This function is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.

Not available to 64-bit applications.

Related Sample Code
BSDLLCTest

QTMetaData

Declared In
`Dialogs.h`

StdFilterProc

Handles standard event filtering for a dialog box.

```
Boolean StdFilterProc (
   DialogRef theDialog,
   EventRecord *event,
   DialogItemIndex *itemHit
);
```

Parameters

`theDialog`

On input, a pointer to a dialog structure for an alert box or a modal dialog box.

`event`

On output, a pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

`itemHit`

On output, a pointer to a short integer. `StdFilterProc` returns a number corresponding to the position of an item in the item list resource for the alert or modal dialog box.

Return Value
A Boolean value representing whether the standard filter proc handled the event. `true` means handled; otherwise `false`.

Discussion
To use the standard filter function from within your own filter function, you can call `GetStdFilterProc` (page 38) , then dispatch the event to the standard filter function yourself; or you can call `StdFilterProc`, which performs both steps for you. Calling `StdFilterProc` is equivalent to calling `GetStdFilterProc` (page 38) and then calling `ModalFilterProcPtr` (page 69).

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

StopAlert

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

```
DialogItemIndex StopAlert (
   SInt16 alertID,
   ModalFilterUPP modalFilter
);
```

Parameters

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'` 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 43) function. If you set this parameter to `null`, the Dialog Manager uses the standard event filter function.

Return Value
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 82). See the description of the `DialogItemIndex` data type.

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` (page 12) function 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.

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`.

See also the functions `NoteAlert` (page 52) and `CautionAlert` (page 17).

Version Notes
This function was changed with Appearance Manager 1.0 to support the extended alert (`'alrx'`) resource.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

UpdateDialog

Redraws the update region of a specified dialog box.

```
void UpdateDialog (
   DialogRef theDialog,
   RgnHandle updateRgn
);
```

Parameters

`theDialog`
A pointer to a dialog structure.

`updateRgn`
A handle to the window region that needs to be updated.

Discussion
The `UpdateDialog` function redraws only the region in a dialog box specified in the `updateRgn` parameter. Your application generally should not use `UpdateDialog`. The Dialog Manager generally handles update events in alert and dialog boxes. `Alert` (page 12). `StopAlert` (page 67) , `NoteAlert` (page 52) , and `CautionAlert` (page 17) handle update events on their own.

Instead of drawing the entire contents of the specified dialog box, `UpdateDialog` draws only the items in the specified update region. You can use `UpdateDialog` in response to an update event, and you should usually bracket it by calls to the Window Manager functions `BeginUpdate` and `EndUpdate`. `UpdateDialog` uses the QuickDraw function `SetPort` to make the dialog box the current graphics port.

Availability
Available in Mac OS X v10.0 and later.
Not available to 64-bit applications.

Declared In
`Dialogs.h`

Callbacks by Task

Accessing and Modifying Low-Memory Data

`UserItemProcPtr` (page 73)

ModalFilterProcPtr (page 69)

SoundProcPtr (page 72)
Defines a pointer to your sound callback function.

Miscellaneous

ModalFilterYDProcPtr (page 71)

QTModelessCallbackProcPtr (page 72)

Callbacks

ModalFilterProcPtr

```
typedef Boolean (*ModalFilterProcPtr)
(
    DialogRef theDialog,
    EventRecord * theEvent,
    DialogItemIndex * itemHit
);
```

If you name your function `MyModalFilterProc`, you would declare it like this:

```
Boolean MyModalFilterProc (
    DialogRef theDialog,
    EventRecord * theEvent,
    DialogItemIndex * itemHit
);
```

Parameters

theDialog
A pointer to a dialog structure for an alert box or a modal dialog box.

theEvent
A pointer to an event structure returned by an Event Manager function such as `WaitNextEvent`.

itemHit
A pointer to a short integer. Your event filter function should return a number corresponding to the position of an item in the item list resource for the alert or modal dialog box.

Return Value

A Boolean value. After receiving an event that it does not handle, your function should return `false`. When your function returns `false`, `ModalDialog` handles the event, which you pass in the parameter `theEvent`. (Your function can also change the event to simulate a different event and return `false`, which passes the event to the Dialog Manager for handling.) If your function does handle the event, your function should return `true`, and through the `itemHit` parameter return the number of the item that it handled.

Discussion
To supplement the Dialog Manager's ability to handle events in the Mac OS multitasking environment, you should provide an event filter function that the Dialog Manager calls whenever it displays alert boxes and modal dialog boxes. This function can receive all events that are sent to your application.

The `ModalDialog` (page 43) function and, in turn, the `Alert` (page 12) , `NoteAlert` (page 52) , `StopAlert` (page 67) , and `CautionAlert` (page 17) functions return the item number that your event filter function returns in the `itemHit` parameter in their own itemHit parameters.

For alert and modal dialog boxes, the Dialog Manager provides a standard event filter function that 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 edit text in a dialog box, and optionally changes the cursor to an I-beam whenever this is the case

If the dialog box is movable modal and the `kDialogHandleMovable` bit is set, your filter function will receive all events (including apple events and update events) that your application receives.

Your own filter function should use the standard filter function to accomplish these tasks. To do so, you can call `GetStdFilterProc` (page 38) , and dispatch the event to the standard filter function yourself; or you can call `StdFilterProc` (page 66) , which obtains a `ModalFilterUPP` for the standard filter function and then dispatches the function.

Your event filter function should also perform the following tasks:

- update your windows in response to update events and return `false`. If you do not handle update events for all the windows in your application, other processes won't get time.
- return `false` for all events that your event filter function doesn't handle

You can also use the event filter function to test for and respond to keyboard equivalents and more complex events—for instance, the user dragging the cursor in an application- defined item. For example, if you provide an application-defined item that requires you to measure how long the user holds down the mouse button or how far the user drags the cursor, use the event filter function to handle events inside that item.

Movable modal dialog boxes receive all events (not just those masked by the Event message mask).

In all alert and dialog boxes, any buttons that are activated by key sequences should highlight to indicate which item has been selected. Use the Control Manager function `HiliteControl` to highlight a button for 8 ticks, long enough to be noticeable but not so long as to be annoying. The Control Manager performs this action whenever users click a button, and your application should do this whenever the user presses the keyboard equivalent of a button click.

For modal dialog boxes that contain edit text items, your application should handle menu bar access to allow use of your Edit menu and its Cut, Copy, Paste, Clear, and Undo commands. Your event filter function should then test for and handle clicks in your Edit menu and keyboard equivalents for the appropriate commands in your Edit menu. Your application should respond by using the functions `DialogCut` (page 22) , `DialogCopy` (page 22) , `DialogPaste` (page 23) , and `DialogDelete` (page 23) to support the Cut, Copy, Paste, and Clear commands.

For an alert box, you specify a universal procedure pointer to your event filter function in a parameter that you pass to the `Alert` (page 12) , `StopAlert` (page 67) , `CautionAlert` (page 17) , and `NoteAlert` (page 52) functions. For a modal dialog box, specify a pointer to your event filter function in a parameter that you pass to `UpdateDialog`.

The Dialog Manager defines the data type `ModalFilterUPP` to identify this application-defined function:

```
typedef UniversalProcPtr ModalFilterUPP;
```

You typically use the `NewModalFilterProc` macro like this:

```
ModalFilterUPP myEventFilterProc;

myEventFilterProc = NewModalFilterProc(MyEventFilter);
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

ModalFilterYDProcPtr

```
typedef Boolean (*ModalFilterYDProcPtr)
(
    DialogRef theDialog,
    EventRecord * theEvent,
    short * itemHit,
    void * yourDataPtr
);
```

If you name your function `MyModalFilterYDProc`, you would declare it like this:

```
Boolean MyModalFilterYDProc (
    DialogRef theDialog,
    EventRecord * theEvent,
    short * itemHit,
    void * yourDataPtr
);
```

Parameters

`theDialog`

`theEvent`

`itemHit`

`yourDataPtr`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

QTModelessCallbackProcPtr

```
typedef void (*QTModelessCallbackProcPtr)
(
    EventRecord *theEvent,
    DialogRef theDialog,
    DialogItemIndex itemHit
);
```

If you name your function `MyQTModelessCallbackProc`, you would declare it like this:

```
void MyQTModelessCallbackProc (
    EventRecord *theEvent,
    DialogRef theDialog,
    DialogItemIndex itemHit
);
```

Parameters

theEvent

theDialog

itemHit

Carbon Porting Notes

This QuickTime function for manipulating dialog boxes is not supported in Carbon.

SoundProcPtr

Defines a pointer to your sound callback function.

```
typedef void (*SoundProcPtr) (
    SInt16 soundNumber
);
```

You should provide a sound callback function if you want the Dialog Manager to play sounds other than the system alert sound. If you name your function `MySoundProc`, you would declare it like this:

```
void MySoundProc (
    SInt16 soundNumber
);
```

Parameters

soundNumber

An integer from 0 to 3, representing the four possible alert stages.

Return Value

Discussion

For each of the four alert stages that can be reported in the `soundNumber` parameter, your function can emit any sound that you define. When the Dialog Manager calls your function, it passes 0 as the sound number for alert sounds specified by the silent constant in the alert resource. The Dialog Manager passes 1 for sounds specified by the `sound1` constant, 2 for sounds specified by the `sound2` constant, and 3 for sounds specified by the `sound3` constant.

The Dialog Manager defines the universal procedure pointer `SoundUPP` to identify this application-defined function:

```
typedef UniversalProcPtr SoundUPP; /
```

You typically use the `NewSoundProc` macro like this:

SoundUPP mySoundProc;

mySoundProc = NewSoundProc(MyAlertSound)

Special Considerations

When the Dialog Manager detects a click outside an alert box or a modal dialog box, it uses the Sound Manager function SysBeep to play the system alert sound. By changing settings in the Sound control panel, the user can select which sound to play as the system alert sound. For consistency with system software and other Macintosh applications, your sound function should call SysBeep whenever your sound function receives sound number 1 (which you can represent with the sound1 constant).

Version Notes

Not recommended with Appearance Manager 1.0 and later.

Carbon Porting Notes

Using custom sounds in dialog boxes is not supported in Carbon.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

UserItemProcPtr

```
typedef void (*UserItemProcPtr) (
    DialogRef theDialog,
    DialogItemIndex itemNo
);
```

If you name your function `MyUserItemProc`, you would declare it like this:

```
void MyUserItemProc (
    DialogRef theDialog,
    DialogItemIndex itemNo
);
```

Parameters

`theDialog`

On input, a pointer to the dialog structure for the dialog box containing an application-defined item. If your function can draw in more than one dialog box, this parameter tells your function which one to draw in.

itemNo

On input, a number corresponding to the position of an item in the item list resource for the specified dialog box. If your function draws more than one item, this parameter tells your function which one to draw.

Return Value

Discussion

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 `MyUserPaneDrawCallback` instead of the user item drawing function `MyUserItemCallback` 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.

When calling your draw function, the Dialog Manager sets the current port to the dialog box's graphics port. Normally, you create an invisible dialog box and then use the Window Manager function `ShowWindow` to display the dialog box.

Before you display the dialog box, use `SetDialogItem` (page 59) to install this function in the dialog structure. Before using `SetDialogItem`, you must first use `GetDialogItem` to obtain a handle to an item of type `userItem`.

If you enable the application-defined item that you draw with this function, `UpdateDialog` and `StdFilterProc` (page 66) return the item's number when the user clicks that item. If your application needs to respond to a user action more complex than this (for example, if your application needs to measure how long the user holds down the mouse or how far the user drags the cursor), your application must track the cursor itself. If you use `ModalDialog`, your event filter function must handle events inside the item; if you use `DialogSelect`, your application must handle events inside the item before handing events to `DialogSelect`.

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

```
typedef UniversalProcPtr UserItemUPP;
```

You typically use the `NewUserItemProc` macro like this:

```
UserItemUPP myItemProc;

myItemProc = NewUserItemProc (MyItem);
```

Version Notes

This function is not recommended with Appearance Manager 1.0 and later.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

Data Types

AlertStdAlertParamRec

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

Fields

`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 43). 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 `null`, the Dialog Manager uses the standard event filter function.

`defaultText`

Text for button in OK position; see "Alert Default Text Constants" (page 82). 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 `null`.

`cancelText`

Text for button in Cancel position; see "Alert Default Text Constants" (page 82). 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 `null`.

`otherText`

Text for button in leftmost position; see "Alert Default Text Constants" (page 82). 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 `null`.

`defaultButton`

Specifies which button acts as the default button; see "Alert Button Constants" (page 82).

`cancelButton`

Specifies which button acts as the Cancel button (can be 0); see "Alert Button Constants" (page 82).

`position`

The alert box position, as defined by a window positioning constant. In this structure, the constant `kWindowDefaultPosition` is equivalent to the constant `kWindowAlertPositionParentWindowScreen`.

Discussion
A standard alert structure of type `AlertStdAlertParamRec` can be used when you call the function `StandardAlert` (page 65) to customize the alert box. The `AlertStdAlertParamRec` type is available with Appearance Manager 1.0 and later.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

AlertStdCFStringAlertParamRec

Defines an alert or sheet.

```
struct AlertStdCFStringAlertParamRec {
    UInt32 version;
    Boolean movable;
    Boolean helpButton;
    CFStringRef defaultText;
    CFStringRef cancelText;
    CFStringRef otherText;
    SInt16 defaultButton;
    SInt16 cancelButton;
    UInt16 position;
    OptionBits flags;
};
typedef struct AlertStdCFStringAlertParamRec AlertStdCFStringAlertParamRec;
typedef AlertStdCFStringAlertParamRec * AlertStdCFStringAlertParamPtr;
```

Fields

`version`
The version of this parameter record. Set this field to `kStdCFStringAlertVersionOne`.

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

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

`defaultText`
Text for button in the OK position. The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1` (see "Alert Default Text Constants" (page 82) for values). To indicate that no button should be displayed, pass `null`

`cancelText`
Text for button in the Cancel position; see "Alert Default Text Constants" (page 82). The button automatically sizes and positions itself in the alert box. To specify that the default button names should be used, pass `-1` see "Alert Default Text Constants" (page 82) for values). To indicate that no button should be displayed, pass `null`.

`otherText`
Text for button in the other (leftmost) position; see "Alert Default Text Constants" (page 82). 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 `null`

`defaultButton`
Specifies which button acts as the default button; see "Alert Button Constants" (page 82).

`cancelButton`
Specifies which button acts as the default button; see "Alert Button Constants" (page 82).

`position`
The alert box position, as defined by a window positioning constant. In this structure, the constant `kWindowDefaultPosition` is equivalent to the constant `kWindowAlertPositionParentWindowScreen`. See the Window Manager Reference for other possible positioning constants.

`flags`
Options for this alert. See "Standard Alert and Sheet Option Flags" (page 93) for possible values.

Discussion
You pass this structure when calling `CreateStandardAlert` (page 20) or `CreateStandardSheet` (page 20).

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

AlertTemplate

```
struct AlertTemplate {
    Rect boundsRect;
    SInt16 itemsID;
    StageList stages;
};
typedef struct AlertTemplate AlertTemplate;
typedef AlertTemplate * AlertTPtr;
```

Fields
`boundsRect`
`itemsID`
`stages`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

AlertType

```
typedef SInt16 AlertType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogItemIndex

```
typedef SInt16 DialogItemIndex;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogItemIndexZeroBased

```
typedef SInt16 DialogItemIndexZeroBased;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogItemType

```
typedef SInt16 DialogItemType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogPeek

```
typedef DialogRecord* DialogPeek;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

DialogRecord

```
struct DialogRecord {
    WindowRecord window;
    Handle items;
    TEHandle textH;
    SInt16 editField;
    SInt16 editOpen;
    SInt16 aDefItem;
};
typedef struct DialogRecord DialogRecord;
```

Fields

Discussion

A dialog structure of type `DialogRecord` is created whenever you call the functions `Alert` (page 12) or `GetNewDialog` (page 36). These functions incorporate information from your item list resource and your alert resource or dialog resource into this structure. Your application generally should not create a dialog structure or directly access its fields.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

DialogRef

```
typedef DialogPtr DialogRef;
```

Availability

Available in Mac OS X v10.0 and later.

Declared In

`Dialogs.h`

DialogTemplate

```
struct DialogTemplate {
    Rect boundsRect;
    SInt16 procID;
    Boolean visible;
    Boolean filler1;
    Boolean goAwayFlag;
    Boolean filler2;
    SInt32 refCon;
    SInt16 itemsID;
    Str255 title;
};
typedef struct DialogTemplate DialogTemplate;
typedef DialogTemplate * DialogTPtr;
```

Fields

`boundsRect`

`procID`

`visible`

`filler1`

`goAwayFlag`

`filler2`

`refCon`

`itemsID`

`title`

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

ModalFilterUPP

```
typedef ModalFilterProcPtr ModalFilterUPP;
```

Discussion
For more information, see the description of the ModalFilterUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

ModalFilterYDUPP

```
typedef ModalFilterYDProcPtr ModalFilterYDUPP;
```

Discussion
For more information, see the description of the ModalFilterYDUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

QTModelessCallbackUPP

```
typedef QTModelessCallbackProcPtr QTModelessCallbackUPP;
```

SoundUPP

```
typedef SoundProcPtr SoundUPP;
```

Discussion
For more information, see the description of the SoundUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

StageList

```
typedef SInt16 StageList;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

UserItemUPP

```
typedef UserItemProcPtr UserItemUPP;
```

Discussion
For more information, see the description of the UserItemUPP () callback function.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Dialogs.h`

Constants

Alert Button Constants

Define standard button types for alerts and sheets.

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

Constants

`kAlertStdAlertOKButton`
The OK button. The default text for this button is “OK”.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertStdAlertCancelButton`
The Cancel button (optional). The default text for this button is “Cancel”.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertStdAlertOtherButton`
A third button (optional). The default text for this button is “Don’t Save”.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertStdAlertHelpButton`
The Help button (optional).
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

Discussion

You can use these constants in the `defaultButton` and `cancelButton` fields in the standard alert structure 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 65). Alert button constants are available with Appearance Manager 1.0 and later.

Alert Default Text Constants

Defines the default text for alerts and sheets.

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

Constants

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion

You can use these constants in the `defaultText`, `cancelText`, and `otherText` fields of the standard alert structure 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.

Alert Feature Flag Constants

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

Constants

`kAlertFlagsUseThemeBackground`

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

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Dialogs.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Dialogs.h`.

`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.

  Available in Mac OS X v10.0 and later.

  Declared in `Dialogs.h`.

Discussion
You can set these bits in the alert flags field of the extended alert resource to specify the alert box's Appearance-compliant features. Alert feature flag constants are available with Appearance Manager 1.0 and later.

Alert Icon Resource ID Constants

```
enum {
    kStopIcon = 0,
    kNoteIcon = 1,
    kCautionIcon = 2,
    stopIcon = kStopIcon,
    noteIcon = kNoteIcon,
    cautionIcon = kCautionIcon
};
```

Constants
`kStopIcon`
: Resource ID for the standard stop icon.

  Available in Mac OS X v10.0 and later.

  Declared in `Dialogs.h`.

`kNoteIcon`
: Resource ID for the standard note icon.

  Available in Mac OS X v10.0 and later.

  Declared in `Dialogs.h`.

`kCautionIcon`
: Resource ID for the standard caution icon.

  Available in Mac OS X v10.0 and later.

  Declared in `Dialogs.h`.

`stopIcon`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`noteIcon`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`cautionIcon`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

Discussion
You can pass these constants in the `alertID` parameter of `StopAlert` (page 67) , `NoteAlert` (page 52), and `CautionAlert` (page 17) to specify the resource ID of the alert box icon you wish displayed.

Alert Type Constants

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

Constants
`kAlertStopAlert`
Stop alert box.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertNoteAlert`
Note alert box.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertCautionAlert`
Caution alert box.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kAlertPlainAlert`
Alert box with no icon.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

Discussion
You can pass constants of type `AlertType` in the `inAlertType` parameter of `StandardAlert` (page 65) to specify the type of alert box you wish to create. Alert type constants are available with Appearance Manager 1.0 and later.

ctrlItem

```
enum {
    ctrlItem = 4,
    btnCtrl = 0,
    chkCtrl = 1,
    radCtrl = 2,
    resCtrl = 3,
    statText = 8,
    editText = 16,
    iconItem = 32,
    picItem = 64,
    userItem = 0,
    itemDisable = 128
};
```

Constants

`ctrlItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`btnCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`chkCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`radCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`resCtrl`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`statText`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`editText`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`iconItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`picItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`userItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`itemDisable`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Dialog Feature Flag Constants

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

Constants

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

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
You can set these bits in the dialog flags field of the extended dialog resource or pass them in the `inFlags` parameter of `NewFeaturesDialog` (page 49) to specify the dialog box's Appearance-compliant features. Dialog feature flag constants are available with Appearance Manager 1.0 and later.

Dialog Font Flag Constants

```
enum {
    kDialogFontNoFontStyle = 0,
    kDialogFontUseFontMask = 0x0001,
    kDialogFontUseFaceMask = 0x0002,
    kDialogFontUseSizeMask = 0x0004,
    kDialogFontUseForeColorMask = 0x0008,
    kDialogFontUseBackColorMask = 0x0010,
    kDialogFontUseModeMask = 0x0020,
    kDialogFontUseJustMask = 0x0040,
    kDialogFontUseAllMask = 0x00FF,
    kDialogFontAddFontSizeMask = 0x0100,
    kDialogFontUseFontNameMask = 0x0200,
    kDialogFontAddToMetaFontMask = 0x0400
};
```

Constants

`kDialogFontNoFontStyle`

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

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseFaceMask`

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

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseJustMask`

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

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontUseAllMask`

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

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`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.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kDialogFontAddToMetaFontMask`

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion

You can set the following bits in the dialog font table resource 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.

Dialog Item Constants

```
enum {
    kControlDialogItem = 4,
    kButtonDialogItem = kControlDialogItem | 0,
    kCheckBoxDialogItem = kControlDialogItem | 1,
    kRadioButtonDialogItem = kControlDialogItem | 2,
    kResourceControlDialogItem = kControlDialogItem | 3,
    kStaticTextDialogItem = 8,
    kEditTextDialogItem = 16,
    kIconDialogItem = 32,
    kPictureDialogItem = 64,
    kUserDialogItem = 0,
    kHelpDialogItem = 1,
    kItemDisableBit = 128
};
```

Constants

`kControlDialogItem`
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kButtonDialogItem`
Standard button control.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kCheckBoxDialogItem`
Standard checkbox control.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kRadioButtonDialogItem`
Standard radio button control.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kResourceControlDialogItem`
Control defined in control resource.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kStaticTextDialogItem`
Static text item.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kEditTextDialogItem`
Edit text item.
Available in Mac OS X v10.0 and later.
Declared in `Dialogs.h`.

`kIconDialogItem`
Icon.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kPictureDialogItem`
QuickDraw picture.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kUserDialogItem`
Application-defined item.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kHelpDialogItem`
Help balloon, as defined by the Help Manager.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kItemDisableBit`
Add to disable any other constant, except `kHelpDialogItem`.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
These constants are returned in the `itemType` parameter of `GetDialogItem` (page 30) and can be passed to `SetDialogItem` (page 59) and the dialog item list resource to specify dialog item type.

Dialog Item List Display Constants

Specify methods of appending new items to a dialog.

```
typedef SInt16 DITLMethod;
enum {
    overlayDITL = 0,
    appendDITLRight = 1,
    appendDITLBottom = 2
};
```

Constants
`overlayDITL`
Superimpose the appended items over the dialog box.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`appendDITLRight`
Position the items to the right of the dialog box and relative to its upper-right coordinate.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`appendDITLBottom`
Position the items below the dialog box and relative to its lower-left coordinate.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Discussion
You can pass a constant value of type `DITLMethod` to the function `AppendDITL` (page 14) to specify how you want appended dialog items displayed.

kDialogFontUseThemeFontIDMask

```
enum {
    kDialogFontUseThemeFontIDMask = 0x0080
};
```

Constants
`kDialogFontUseThemeFontIDMask`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

kHICommandOther

```
enum {
    kHICommandOther = 'othr'
};
```

Constants
`kHICommandOther`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

kOkItemIndex

```
enum {
    kOkItemIndex = 1,
    kCancelItemIndex = 2
};
```

Constants
`kOkItemIndex`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kCancelItemIndex`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Standard Alert and Sheet Option Flags

Define flags used in the `AlertStdCFStringAlertParamRec` (page 76) structure.

```
enum {
    kStdAlertDoNotDisposeSheet = 1 << 0,
    kStdAlertDoNotAnimateOnDefault = 1 << 1,
    kStdAlertDoNotAnimateOnCancel = 1 << 2,
    kStdAlertDoNotAnimateOnOther = 1 << 3,
    kStdAlertDoNotCloseOnHelp     = 1 << 4
};
```

Constants

`kStdAlertDoNotDisposeSheet`

Do not dispose of the sheet window after closing it. This option allows the sheet to be used again when calling the Window Manager function `ShowSheetWindow`.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotAnimateOnDefault`

Do not animate hiding the sheet window when the user presses the default button.

Available in Mac OS X v10.1 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotAnimateOnCancel`

Do not animate hiding the sheet window when the user presses the Cancel button.

Available in Mac OS X v10.1 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotAnimateOnOther`

Do not animate hiding the sheet window when the user presses the other button.

Available in Mac OS X v10.1 and later.

Declared in `Dialogs.h`.

`kStdAlertDoNotCloseOnHelp`

Specifies that the alert stay up even after the user clicks the Help button. Normally, it would close immediately. It is not necessary to set this option for sheets, as they merely send the `HICommandHelp` command to the event target provided. When you specify this option, `RunStandardAlert` (page 55) returns with the Help button item in the `outItemHit` parameter, but the alert remains up. You can then perform whatever help function you wish and then call `RunStandardAlert` again.

Declared in `Dialogs.h`.

Available in Mac OS X 10.4 or later.

Standard Alert Structure Version Constant

Indicates the version of the `AlertStdCFStringAlertParamRec` (page 76) structure.

```
enum {
    kStdCFStringAlertVersionOne = 1
};
```

Constants

`kStdCFStringAlertVersionOne`
First version. Pass this into the `version` field of the `AlertStdCFStringAlertParamRec` structure.

Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

kStdOkItemIndex

```
enum {
    kStdOkItemIndex = 1,
    kStdCancelItemIndex = 2,
    ok = kStdOkItemIndex,
    cancel = kStdCancelItemIndex
};
```

Constants

`kStdOkItemIndex`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`kStdCancelItemIndex`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`ok`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

`cancel`
Available in Mac OS X v10.0 and later.

Declared in `Dialogs.h`.

Result Codes

The result codes defined for the Dialog Manager are listed below.

Result Code	Value	Description
`dialogNoTimeoutErr`	-5640	No timeout has been set for this dialog. Available in Mac OS X v10.0 and later.

Gestalt Constants

You can check for version and feature availability information by using the Dialog Manager selectors defined in the Gestalt Manager. For more information, see *Gestalt Manager Reference*.

Document Revision History

This table describes the changes to *Dialog Manager Reference*.

Date	Notes
2007-10-31	Made minor technical corrections and format changes.
2005-07-07	Added bug fixes. Documented AlertStdCFStringAlertParamRec structure.
2003-05-20	Added note to `RunStandardAlert` (page 55) indicating that the function disposes of the dialog you pass to it before returning.
2003-02-15	Updated formatting and linking.

Document Revision History

Index

H

I

K

M

N

O

P

Q

R

S

U