INSIDE MACINTOSH

Mac OS 8
Menu Manager Reference

Updated for Appearance 1.0.2

November 16, 1998
Technical Publications

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

Simultaneously published in the United States and Canada.

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

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

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

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

Contents

Figures, Tables, and Listings

CHAPTER 1

Menu Manager Reference

Contents

Menus allow users to view or choose from a list of choices and commands that your application provides. You can use the Menu Manager to create, display, and manage the drawing and behavior of pull-down, hierarchical, and contextual menus.

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

- "Contextual Menu Gestalt Selector Constants" (page 9)
- "Menu Manager Functions" (page 10)
- "Menu Manager Data Types" (page 43)
- "Menu Manager Constants" (page 56)

For descriptions of the parts of the Menu Manager API that are unaffected by Mac OS 8 or Appearance Manager 1.0, see *Inside Macintosh: Macintosh Toolbox Essentials*. For a description of the Mac OS 8.5 Menu Manager API, see *Mac OS 8.5 Menu Manager Reference*.

Contextual Menu Gestalt Selector Constants

Before calling any contextual menu functions, your application should pass the selector `gestaltContextualMenuAttr` to the `Gestalt` function to determine whether contextual menu functions are available.

```
enum{
    gestaltContextualMenuAttr   = 'cmnu'
};
```

Constant description

`gestaltContextualMenuAttr`
The `Gestalt` selector passed to the `Gestalt` function to determine whether contextual menu functions are available. Produces a value whose bits you should test to determine whether the contextual menu functions are available.

Your program can use the following value to test for the presence of contextual menu functions:

```
enum{
    gestaltContextualMenuTrapAvailable  = 1
};
```

Constant description

`gestaltContextualMenuTrapAvailable`
If this bit is set, the contextual menu functions are available to 68K applications. If this bit is not set, these functions are not available to 68K applications.

Menu Manager Functions

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

- "Initializing the Menu Manager" (page 10)
- "Creating Menus" (page 13)
- "Responding to the User's Choice of a Menu Command" (page 15)
- "Manipulating and Accessing Menu Item Characteristics" (page 20)
- "Defining Your Own Contextual Menu Plug-In" (page 36)

Initializing the Menu Manager

The following Menu Manager functions for initializing the Menu Manager are new, changed, or not recommended with Appearance Manager 1.0:

- `InitContextualMenus` (page 11) adds a program to the system registry of contextual menu clients. New with Appearance Manager 1.0.
- `ProcessIsContextualMenuClient` (page 11) determines whether a given program is a contextual menu client. New with Appearance Manager 1.0.

- `InitProcMenu` (page 12) sets the `mbResID` field of the current menu list to the resource ID of a custom `'MBDF'` resource. Changed with Appearance Manager 1.0.

InitContextualMenus

Adds a program to the system registry of contextual menu clients.

```
pascal OSStatus InitContextualMenus (void);
```

function result A result code; see "Result Codes" (page 61).

DISCUSSION

Your program should call the `InitContextualMenus` function early in your startup code to register your application as a contextual menu client. If you do not register your program, some system-level functions may respond as though your program does not use contextual menus. Not registering your program may also cause `ProcessIsContextualMenuClient` (page 11) to return an incorrect value.

If you have a 68K program, you must pass the selector `gestaltContextualMenuAttr` to the `Gestalt` function before calling the `InitContextualMenus` function. If the `Gestalt` function returns a bit field with the `gestaltContextualTrapAvailable` bit set, `InitContextualMenus` can be called; see "Contextual Menu Gestalt Selector Constants" (page 9).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

ProcessIsContextualMenuClient

Determines whether a given program is a contextual menu client.

```
pascal Boolean ProcessIsContextualMenuClient(ProcessSerialNumber* inPSN);
```

`inPSN` A pointer to the ID of the process containing the program.

function result A Boolean value. `ProcessIsContextualMenuClient` returns `true` if the program in the process uses contextual menus; otherwise, `false`.

DISCUSSION

The `ProcessIsContextualMenuClient` function checks the system registry of contextual menu clients and returns `true` if the program in the given process supports contextual menus. However, the program must have been registered as a client using `InitContextualMenus` (page 11).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

“Contextual Menu Gestalt Selector Constants” (page 9).

InitProcMenu

Sets the `mbResID` field of the current menu list to the resource ID of a custom `'MBDF'` resource.

```
pascal void InitProcMenu (short resID);
```

`resID` The resource ID of your application’s menu bar definition function in the upper 13 bits of this parameter; the variant in the lower 3 bits. You must use a resource ID greater than 0x100. Resource IDs 0x000 through 0x100 are reserved for the use of Apple Computer, Inc.

DISCUSSION

If your application provides its own menu bar definition function, use the `InitProcMenu` function to associate your custom `'MBDF'` resource with the current menu list. In general, you should not use a custom menu bar definition unless absolutely necessary. `InitProcMenu` creates the current menu list if it hasn’t already been created by a previous call to `InitMenus` .

You can also call `InitProcMenu` to bypass mapping of the pre-Appearance menu resource ID constant `textMenuProc` to its corresponding Appearance-compliant menu resource ID constant `kMenuStdMenuProc` when mapping is enabled.

SPECIAL CONSIDERATIONS

The resource ID of your application's menu bar definition function is maintained in the current menu list until your application next calls `InitMenus`; `InitMenus` initializes the `mbResID` field with the resource ID of the standard menu bar definition function. This can affect applications such as development environments that control other applications which may call `InitMenus`.

VERSION NOTES

Bypasses definition function mapping under Appearance Manager 1.0 and later.

Creating Menus

The following Menu Manager function for creating menus is changed with Appearance Manager 1.0:

- `GetMenu` (page 13) creates a menu from the specified menu and extended menu resources. Changed with Appearance Manager 1.0.

GetMenu

Creates a menu from the specified menu and extended menu resources.

```
pascal MenuHandle GetMenu (short resourceID);
```

`resourceID` The resource ID of the menu and extended menu that defines the characteristics of the menu. You typically use the same number for a menu's resource ID as the number that you specify for the menu ID in the menu resource.

function result Returns a handle to the menu structure for the menu. You can use the returned menu handle to refer to this menu in most Menu Manager functions. If `GetMenu` is unable to read the menu or menu definition function from the resource file, `GetMenu` returns `nil`.

DISCUSSION

In addition to creating a menu, the `GetMenu` function also creates a menu structure for the menu. `GetMenu` reads the menu definition function into memory (if not already present) and stores a handle to the menu definition function in the menu structure. `GetMenu` does not insert the newly created menu into the current menu list.

Note
You typically use the `GetMenu` function only when you create submenus; you can create all your pull-down menus at once using the function `GetNewMBar`, and you can create pop-up menus using the standard pop-up menu button control definition function.

After reading the `'MENU'` resource (page 44), `GetMenu` searches for an extended menu resource and an `'mctb'` resource with the same resource ID as the `'MENU'` resource. If the specified `'mctb'` resource exists, `GetMenu` uses `SetMCEntries` to add the entries defined by the resource to the application's menu color information table. If the `'mctb'` resource does not exist, `GetMenu` uses the default colors specified in the menu bar entry of the application's menu color information. If neither a menu bar entry nor a `'mctb'` resource exists, `GetMenu` uses the standard colors for the menu.

Storing the definitions of your menus in resources (especially menu titles and menu items) makes your application easier to localize.

▲ **WARNING**
Menus in a resource must not be purgeable nor should the resource lock bit be set. Do not define a "circular" hierarchical menu—that is, a hierarchical menu in which a submenu has a submenu whose submenu is a hierarchical menu higher in the chain.

SPECIAL CONSIDERATIONS

To release the memory associated with a menu that you created using `GetMenu`, first call `DeleteMenu` to remove the menu from the current menu list and to remove any entries for this menu in your application's menu color information table; then call `DisposeMenu` to dispose of the menu structure. After disposing of a menu, use `DrawMenuBar` to update the menu bar.

VERSION NOTES

Changed with Appearance Manager 1.0 to support the extended menu resource.

Responding to the User's Choice of a Menu Command

The following Menu Manager functions for responding to the user's choice of a menu command are new, changed, or not recommended with Appearance Manager 1.0:

- `MenuEvent` (page 15) maps a keyboard combination from the event structure to the keyboard equivalent of a menu item in a menu in the current menu list. New with Appearance Manager 1.0.
- `MenuKey` (page 17) maps a character key with the Command key to determine the keyboard equivalent of a menu item in a menu in the current menu list. Not recommended with Appearance Manager 1.0.
- `IsShowContextualMenuClick` (page 17) returns whether a particular event could invoke a contextual menu. New with Appearance Manager 1.0.
- `ContextualMenuSelect` (page 18) displays a contextual menu. New with Appearance Manager 1.0.

MenuEvent

Maps a keyboard combination from the event structure to the keyboard equivalent of a menu item in a menu in the current menu list.

```
pascal UInt32 MenuEvent (EventRecord* inEvent);
```

`inEvent` A pointer to the event structure containing the keyboard equivalent.

function result Returns a value that indicates the menu ID and menu item that the user chose. If the given character does not map to an enabled menu item in the current menu list, `MenuEvent` returns 0 in its high-order word and the low-order word is undefined.

DISCUSSION

When the Appearance Manager is available, your program should call the `MenuEvent` function instead of the `MenuKey` function. `MenuEvent` maps the keyboard equivalent character in the event structure to its corresponding menu and menu item. Unlike `MenuKey`, the `MenuEvent` function supports the Shift, Option, and Control modifier keys in addition to the Command key.

The `MenuEvent` function does not distinguish between uppercase and lowercase letters. This allows a user to invoke a keyboard equivalent command, such as the Copy command, by pressing the Command key and “c” or “C”. For consistency between applications, you should define the keyboard equivalents of your commands so that they appear in uppercase in your menus.

If the given character maps to an enabled menu item in the current menu list, `MenuEvent` highlights the menu title of the chosen menu, returns the menu ID in the high-order word of its function result, and returns the chosen menu item in the low-order word of its function result. After performing the chosen task, your application should unhighlight the menu title using the `HiliteMenu` function.

You should not define menu items with identical keyboard equivalents. The `MenuEvent` function scans the menus from right to left and the items from top to bottom. If you have defined more than one menu item with identical keyboard equivalents, `MenuEvent` returns the first one it finds.

The `MenuEvent` function first searches the regular portion of the current menu list for a menu item with a keyboard equivalent matching the given key. If it doesn’t find one there, it searches the submenu portion of the current menu list. If the given key maps to a menu item in a submenu, `MenuEvent` highlights the menu title in the menu bar that the user would normally pull down to begin traversing to the submenu. Your application should perform the desired command and then unhighlight the menu title.

You shouldn’t assign a Command–Shift–number key sequence to a menu item as its keyboard equivalent; Command–Shift–number key sequences are reserved for use as `'FKEY'` resources. Command–Shift–number key sequences are not returned to your application, but instead are processed by the Event Manager. The Event Manager invokes the `'FKEY'` resource with a resource ID that corresponds to the number that activates it.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

MenuKey

Maps a character key with the Command key to determine the keyboard equivalent of a menu item in a menu in the current menu list. When the Appearance Manager is available, call `MenuEvent` (page 15) instead of `MenuKey`, in order to map the keyboard equivalent character in the event structure to its corresponding menu and menu item. Unlike `MenuKey`, the `MenuEvent` function supports the Shift, Option, and Control modifier keys in addition to the Command key.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

IsShowContextualMenuClick

Returns whether a particular event could invoke a contextual menu.

```
pascal Boolean IsShowContextualMenuClick(const EventRecord* inEvent);
```

`inEvent`	A pointer to the event structure that describes the event to examine.
function result	Returns a Boolean value indicating whether or not a contextual menu should be displayed. If `true`, the contextual menu should be displayed; if `false`, not.

DISCUSSION

Before calling the `IsShowContextualMenuClick` function, you should call `InitContextualMenus` (page 11). If no error is returned, you can then call `IsShowContextualMenuClick`.

Applications should call `IsShowContextualMenuClick` when they receive non-null events. If `IsShowContextualMenuClick` returns `true`, your application should generate its own menu and Apple Event descriptor (`AEDesc`), and then call `ContextualMenuSelect` (page 18) to display and track the contextual menu, and then handle the user's choice.

If the mouse-down event did not invoke a contextual menu, then the application should check to see if the event occurred in the menu bar (using the

FindWindow function) and, if so, call MenuSelect to allow the user to choose a command from the menu bar.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

"Contextual Menu Gestalt Selector Constants" (page 9).

ContextualMenuSelect

Displays a contextual menu.

```
pascal OSStatus ContextualMenuSelect (
                    MenuHandle inMenu,
                    Point inGlobalLocation,
                    Boolean inReserved,
                    UInt32 inHelpType,
                    ConstStr255Param inHelpItemString,
                    const AEDesc* inSelection,
                    UInt32* outUserSelectionType,
                    SInt16* outMenuID,
                    UInt16* outMenuItem);
```

`inMenu` Pass a handle to a menu containing application commands to display. The caller creates this menu based on the current context, the mouse location, and the current selection (if it was the target of the mouse). If you pass `nil`, only system commands will be displayed. The menu should be added to the menu list as a pop-up menu (using the `InsertMenu` function).

`inGlobalLocation`
Pass the location (in global coordinates) of the mouse near which the menu is to be displayed.

`inReserved` Reserved for future use. Pass `false` for this parameter.

`inHelpType` Pass an identifier specifying the type of help to be provided by the application; see "Contextual Menu Help Type Constants" (page 57).

`inHelpItemString`
Pass the string containing the text to be displayed for the help menu item. This string is unused unless you also pass the constant `kCMOtherHelp` in the `inHelpType` parameter.

`inSelection` Pass a pointer to an object specifier for the current selection. This allows the system to examine the selection and add special system commands accordingly. Passing a value of `nil` indicates that no selection should be examined, and most likely, no special system actions will be included.

`outUserSelectionType`
Pass a pointer to an unsigned 32-bit value. On return, the value indicates what the user selected from the contextual menu; see "Contextual Menu Help Type Constants" (page 57).

`outMenuID` Pass a pointer to a signed 16-bit value. On return, if `outUserSelectionType` is set to `kCMMenuItemSelected`, the value is set to the menu ID of the chosen item.

`outMenuItem` Pass a pointer to an unsigned 16-bit value. On return, if `outUserSelectionType` is set to `kCMMenuItemSelected`, the value is set to the menu item chosen.

function result A result code; see "Result Codes" (page 61). `ContextualMenuSelect` returns the result code `userCanceledErr` and sets `outUserSelectionType` to `kCMNothingSelected` to indicate that the user did not select anything from the contextual menu and no further processing is needed.

DISCUSSION

If the `IsShowContextualMenuClick` function returns `true`, you should call the `ContextualMenuSelect` function after generating your own menu and preparing an Apple Event descriptor (`AEDesc`) that describes the item for which your application is displaying a contextual menu. This descriptor may contain an object specifier or raw data and will be passed to all contextual menu plug-ins.

The system will add other items before displaying the contextual menu, and it will remove those items before returning, leaving the menu in its original state.

After all the system commands are added, the contextual menu is displayed and tracked. If the user selects one of the system items, it is handled by the system and the call returns as though the user didn't select anything from the menu. If the user selects any other item (or no item at all), the Menu Manager passes back appropriate values in the parameters `outUserSelectionType`, `outMenuID`, and `outMenuItem`.

Your application should provide visual feedback indicating the item that was clicked upon. For example, a click on an icon should highlight the icon, while a click on editable text should not eliminate the current selection.

If the `outUserSelectionType` parameter contains `kCMMenuItemSelected`, you should look at the `outMenuID` and `outMenuItem` parameters to determine what menu item the user chose and handle it appropriately. If the user selected `kCMHelpItemSelected`, you should open the proper Apple Guide sequence or other form of custom help.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

"Contextual Menu Gestalt Selector Constants" (page 9).

Manipulating and Accessing Menu Item Characteristics

The following Menu Manager functions for manipulating and accessing menu item characteristics are new, changed, or not recommended with Appearance Manager 1.0:

- `SetItemCmd` (page 22) sets the value of the keyboard equivalent field of a menu item. Not recommended with Appearance Manager 1.0.
- `SetMenuItemCommandID` (page 22) sets a menu item's command ID. New with Appearance Manager 1.0.
- `GetMenuItemCommandID` (page 23) obtains a menu item's command ID. New with Appearance Manager 1.0.
- `SetMenuItemFontID` (page 24) sets the font for a menu item. New with Appearance Manager 1.0.

- `GetMenuItemFontID` (page 24) obtains a menu item's font ID. New with Appearance Manager 1.0.
- `SetMenuItemHierarchicalID` (page 25) attaches a submenu to a menu item. New with Appearance Manager 1.0.
- `GetMenuItemHierarchicalID` (page 26) obtains the menu ID of a specified submenu. New with Appearance Manager 1.0.
- `SetMenuItemIconHandle` (page 26) sets a menu item's icon. New with Appearance Manager 1.0.
- `GetMenuItemIconHandle` (page 27) obtains a handle to a menu item's icon. New with Appearance Manager 1.0.
- `SetMenuItemKeyGlyph` (page 28) substitutes a keyboard glyph for the glyph normally displayed for a menu item's keyboard equivalent. New with Appearance Manager 1.0.
- `GetMenuItemKeyGlyph` (page 29) obtains the keyboard glyph for a menu item's keyboard equivalent. New with Appearance Manager 1.0.
- `SetMenuItemModifiers` (page 30) sets the modifier key(s) that must be pressed with a character key to select a particular menu item. New with Appearance Manager 1.0.
- `GetMenuItemModifiers` (page 31) obtains the modifier keys that must be pressed with a character key to select a particular menu item. New with Appearance Manager 1.0.
- `SetMenuItemRefCon` (page 32) sets application-specific information for a menu item. New with Appearance Manager 1.0.
- `GetMenuItemRefCon` (page 32) obtains application-specific information for a menu item. New with Appearance Manager 1.0.
- `SetMenuItemRefCon2` (page 33) sets additional application-specific information for a menu item. New with Appearance Manager 1.0.
- `GetMenuItemRefCon2` (page 34) obtains application-specific information for a menu item. New with Appearance Manager 1.0.
- `SetMenuItemTextEncoding` (page 35) sets the text encoding for a menu item's text. New with Appearance Manager 1.0.
- `GetMenuItemTextEncoding` (page 36) obtains the text encoding used for a menu item's text. New with Appearance Manager 1.0.

SetItemCmd

Sets the value of the keyboard equivalent field of a menu item. When the Appearance Manager is available, you should call `SetMenuItemModifiers` (page 30), `SetMenuItemHierarchicalID` (page 25), and `SetMenuItemTextEncoding` (page 35) instead of `SetItemCmd` to set a menu item's keyboard equivalent and text encoding and to indicate that a menu item has a submenu.

VERSION NOTES

Not recommended with Appearance Manager 1.0 and later.

SetMenuItemCommandID

Sets a menu item's command ID.

```
pascal OSErr SetMenuItemCommandID (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt32 inCommandID);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to set a command ID.
`inItem`	An integer representing the item number of the menu item.
`inCommandID`	An integer representing the command ID that you wish to set.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

You can use a menu item's command ID as a position-independent method of signalling a specific action in an application.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemCommandID` (page 23).

GetMenuItemCommandID

Obtains a menu item's command ID.

```
pascal OSErr GetMenuItemCommandID (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt32* outCommandID);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to get a command ID.
`inItem`	An integer representing the item number of the menu item.
`outCommandID`	Pass a pointer to an unsigned 32-bit integer value. On return, the value is set to the item's command ID.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

After a successful call to `MenuSelect`, `MenuEvent` (page 15), or `MenuKey` (page 17), call the `GetMenuItemCommandID` function to get a menu item's command ID. You can use a menu item's command ID as a position-independent method of signalling a specific action in an application.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemCommandID` (page 22).

SetMenuItemFontID

Sets the font for a menu item.

```
pascal OSErr SetMenuItemFontID (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    SInt16 inFontID);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to set the font.
`inItem`	An integer representing the item number of the menu item.
`inFontID`	An integer representing the font ID that you wish to set.
function result	A result code; see “Result Codes” (page 61).

DISCUSSION

The `SetMenuItemFontID` function enables you to set up a font menu with each item being drawn in the actual font.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemFontID` (page 24).

GetMenuItemFontID

Obtains a menu item’s font ID.

```
pascal OSErr GetMenuItemFontID (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    SInt16* outFontID);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to get a font ID.

`inItem`	An integer representing the item number of the menu item.
`outFontID`	Pass a pointer to a signed 16-bit integer value. On return, the value is set to the font ID for the menu item.
function result	A result code; see “Result Codes” (page 61).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemFontID` (page 24).

SetMenuItemHierarchicalID

Attaches a submenu to a menu item.

```
pascal OSErr SetMenuItemHierarchicalID (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    SInt16 inHierID)
```

`inMenu`	A handle to the menu that contains the menu item to which you wish to attach a submenu.
`inItem`	An integer representing the item number of the menu item.
`inHierID`	An integer representing the menu ID of the submenu you wish to attach. This menu should be inserted into the menu list by calling `InsertMenu`.
function result	A result code; see “Result Codes” (page 61).

DISCUSSION

The `SetMenuItemHierarchicalID` function should be called instead of setting the keyboard equivalent to 0x1B. When the Appearance Manager is available, you should call `SetMenuItemHierarchicalID` instead of `SetItemMark` to set the menu ID of a menu item’s submenu. However, you can still use `SetItemMark` to set the mark of a menu item.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Setting a submenu with an menu ID greater than 255 is only supported under Mac OS 8.5 and later.

GetMenuItemHierarchicalID

Obtains the menu ID of a specified submenu.

```
pascal OSErr GetMenuItemHierarchicalID (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    SInt16 *outHierID)
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to get the submenu's menu ID.
`inItem`	An integer representing the item number of the menu item.
`outHierID`	Pass a pointer to a signed 16-bit integer value. On return, the value is set to the menu ID of the submenu.
function result	A result code; see "Result Codes" (page 61).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SetMenuItemIconHandle

Sets a menu item's icon.

```
pascal OSErr SetMenuItemIconHandle (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt8 inIconType,
                    Handle inIconHandle);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to set an icon.

`inItem`	An integer representing the item number of the menu item.
`inIconType`	Pass a value representing the type of icon (`'ICON'`, `'cicn'`, `'SICN'`, icon suite, or `IconRef`) you wish to attach; see "Menu Item Icon Type Constants" (page 59) for descriptions of possible values.
`inIconHandle`	Pass a handle to the icon you wish to attach to a menu item.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

The `SetMenuItemIconHandle` function sets the icon of a menu item with an icon handle instead of a resource ID. `SetMenuItemIconHandle` allows you to set icons of type `'ICON'`, `'cicn'`, `'SICN'`, as well as icon suites. To set resource-based icons for a menu item, call `SetItemIcon`.

▲ **WARNING**
Disposing of the menu will not dispose of the icon handles set by this function. To prevent memory leaks, your application should dispose of the icons when you dispose of the menu.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemIconHandle` (page 27).

GetMenuItemIconHandle

Obtains a handle to a menu item's icon.

```
pascal OSErr GetMenuItemIconHandle (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt8* outIconType,
                    Handle* outIconHandle);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to obtain the handle.
`inItem`	An integer representing the item number of the menu item.
`outIconType`	Pass a pointer to an unsigned 8-bit value. On return, the value specifies the type of icon (`'ICON'`, `'cicn'`, `'SICN'`, icon suite, or `IconRef`) for which you are obtaining a handle. If the menu item has no icon attached, this parameter will contain `kMenuNoIcon`. See "Menu Item Icon Type Constants" (page 59) for descriptions of possible values.
`outIconHandle`	Pass a pointer to a handle. On return, `outIconHandle` contains a handle to the icon that is attached to the menu item. If the menu item has no icon attached, this parameter contains `nil`.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

The `GetMenuItemIconHandle` function gets the icon handle and type of icon of the specified menu item. If you wish to get a resource-based menu item icon, call `GetItemIcon`.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemIconHandle` (page 26).

SetMenuItemKeyGlyph

Substitutes a keyboard glyph for the glyph normally displayed for a menu item's keyboard equivalent.

```
pascal OSErr SetMenuItemKeyGlyph (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    SInt16 inGlyph)
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to substitute a keyboard glyph.
`inItem`	An integer representing the item number of the menu item.
`inGlyph`	An integer representing the substitute glyph to display. Pass 0 if you wish no substitution to occur. For a description of keyboard glyphs and a list of the keyboard font character codes, see `'xmnu'` (page 50).
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

The `SetMenuItemKeyGlyph` function overrides the character that would normally be displayed in a menu item's keyboard equivalent with a substitute keyboard glyph. This is useful if the keyboard glyph in the font doesn't match the actual character generated. For example, you might use this function to display function keys.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemKeyGlyph` (page 29).

GetMenuItemKeyGlyph

Obtains the keyboard glyph for a menu item's keyboard equivalent.

```
pascal OSErr GetMenuItemKeyGlyph (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    SInt16 *outGlyph)
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to get the keyboard glyph.
`inItem`	An integer representing the item number of the menu item.

`outGlyph`	A pointer to a signed 16-bit integer value. On return the value is set to the modifier key glyph. For a description of keyboard glyphs and a list of the keyboard font character codes, see `'xmnu'` (page 50).
function result	A result code; see "Result Codes" (page 61).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemIconHandle` (page 26).

SetMenuItemModifiers

Sets the modifier key(s) that must be pressed with a character key to select a particular menu item.

```
pascal OSErr SetMenuItemModifiers (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt8 inModifiers);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to set the modifier key(s).
`inItem`	An integer representing the item number of the menu item.
`inModifiers`	A value representing the modifier key(s) to be used in selecting the menu item; see "Modifier Key Mask Constants" (page 58).
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

You can call the `SetMenuItemModifiers` function to change the modifier key(s) you can include with a character key to create your keyboard equivalent. For example, you can change Command-*x* to Command-Option-Shift-*x*. By default, the Command key is always specified; however, you can remove the Command

key by setting the `kMenuNoCommand` flag in the modifier keys field of an extended menu item entry in the `'xmnu'` resource; see `'xmnu'` (page 50).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemModifiers` (page 31).

GetMenuItemModifiers

Obtains the modifier keys that must be pressed with a character key to select a particular menu item.

```
pascal OSErr GetMenuItemModifiers (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt8* outModifiers);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to get the modifier key(s).
`inItem`	An integer representing the item number of the menu item.
`outModifiers`	A pointer to an unsigned 8-bit value. On return, the bits of the value are set to indicate the modifier keys that can be used in selecting the menu item; see “Modifier Key Mask Constants” (page 58).
function result	A result code; see “Result Codes” (page 61).

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemModifiers` (page 30).

SetMenuItemRefCon

Sets application-specific information for a menu item.

```
pascal OSErr SetMenuItemRefCon (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt32 inRefCon);
```

`inMenu`	A handle to the menu that contains the menu item with which you wish to associate information.
`inItem`	An integer representing the item number of the menu item.
`inRefCon`	An unsigned 32-bit integer value. Pass a reference constant to associate with the menu item.
function result	A result code; see “Result Codes” (page 61).

DISCUSSION

If you have any data you want to associate with a menu item, you can do so using the `SetMenuItemRefCon` function.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemRefCon` (page 32).

GetMenuItemRefCon

Obtains application-specific information for a menu item.

```
pascal OSErr GetMenuItemRefCon (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt32* outRefCon);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to get information.
`inItem`	An integer representing the item number of the menu item.
`outRefCon`	A pointer to an unsigned 32-bit integer value. On return, the value is set to the reference constant associated with the menu item.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

If you have assigned any data to a menu item using `SetMenuItemRefCon` function, you can read it using the `GetMenuItemRefCon` function.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemRefCon` (page 32).

SetMenuItemRefCon2

Sets additional application-specific information for a menu item.

```
pascal OSErr SetMenuItemRefCon2 (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt32 inRefCon);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to set information.
`inItem`	An integer representing the item number of the menu item.
`inRefCon`	An unsigned 32-bit integer value. Pass a reference constant to associate with the menu item.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

If you have data you want to associate with a menu item in addition to that set with the `SetMenuItemRefCon` (page 32) function, you can do so using the `SetMenuItemRefCon2` function.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemRefCon2` (page 34).

GetMenuItemRefCon2

Obtains application-specific information for a menu item.

```
pascal OSErr GetMenuItemRefCon2 (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    UInt32* outRefCon);
```

`inMenu`	A handle to the menu that contains the menu item for which you wish to retrieve information.
`inItem`	An integer representing the item number of the menu item.
`outRefCon`	A pointer to an unsigned 32-bit integer value. On return, the value is set to the reference constant associated with the menu item.
function result	A result code; see “Result Codes” (page 61).

DISCUSSION

If you have assigned any data to a given menu item using `SetMenuItemRefCon2` function, you can read it using the `GetMenuItemRefCon` function.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemRefCon2` (page 33).

SetMenuItemTextEncoding

Sets the text encoding for a menu item's text.

```
pascal OSErr SetMenuItemTextEncoding (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    TextEncoding inScriptID);
```

`inMenu`	A handle to the menu that contains the menu item whose text encoding you wish to set.
`inItem`	An integer representing the item number of the menu item.
`inScriptID`	The script code that corresponds to the text encoding you wish to set.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

To set the text encoding for a menu item's text, call the `SetMenuItemTextEncoding` function instead of `SetItemCmd`. If a menu item has a command code of 0x1C when `SetMenuItemTextEncoding` is called, the values in the command and icon fields of the menu resource are cleared and replaced with the value in the `inScriptID` parameter of `SetMenuItemTextEncoding`.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`GetMenuItemTextEncoding` (page 36).

GetMenuItemTextEncoding

Obtains the text encoding used for a menu item's text.

```
pascal OSErr GetMenuItemTextEncoding (
                    MenuHandle inMenu,
                    SInt16 inItem,
                    TextEncoding* outScriptID);
```

`inMenu`	A handle to the menu that contains the menu item whose text encoding you wish to get.
`inItem`	An integer representing the item number of the menu item.
`outScriptID`	A pointer to a `TextEncoding` value. On return, the value is set to the script code of the text encoding used in the menu item's text.
function result	A result code; see "Result Codes" (page 61).

DISCUSSION

If a menu item has a command code of 0x1C when `GetMenuItemTextEncoding` is called, `GetMenuItemTextEncoding` gets the value in the icon field of the menu item's menu resource.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

SEE ALSO

`SetMenuItemTextEncoding` (page 35).

Defining Your Own Contextual Menu Plug-In

The following Menu Manager methods for defining your own contextual menu plug-in are new with Appearance Manager 1.0:

- `Initialize` (page 38) performs any required plug-in initialization.
- `ExamineContext` (page 39) examines the context chosen by the user and determines possible menu commands appropriate to the context.
- `HandleSelection` (page 42) executes the contextual menu item chosen by the user.
- `PostMenuCleanup` (page 43) performs any necessary cleanup when the contextual menu is dismissed.

Note
A contextual menu plug-in is implemented as a SOMObject object inside a shared library. (SOMObjects for the Mac OS platform is the Mac OS implementation of the System Object Model.) Typically your development environment can compile directly to a SOMObject object, so you do not need to create your own SOM interfaces.

A contextual menu plug-in is a subclass of `AbstractCMPlugin : SOMObject`. It consists of four methods described above. Each subclass of the `AbstractCMPlugin` must have an extended `'cfrg'` resource, through which it identifies itself as a SOMObject object which derives from the `AbstractCMPlugin` class. See *Mac OS Runtime Architectures* for information about the extended `'cfrg'` resource.

In addition you must register the plug-in class as a SOMObject object so that the Menu Manager can instantiate it by name. Typically you can do this in a fragment's initialization function.

Listing 1-1 shows a sample initialization function that registers the plug-in.

Listing 1-1 Registering a contextual menu plug-in

```
pascal OSErr MyPluginInitialize(CFragInitBlockPtr init)
{
    /* If your compiler creates a default initialization function,*/
    /* you should call it here */

    /* Now register our class with SOM */
        somNewClass(MyPlugIn);
```

```
    return noErr;
}
```

The class declaration for a contextual menu definition plug-in is as follows:

```
class AbstractCMPlugin: SOMObject
{
    OSStatus Initialize(FSSpec *inFileSpec);
    OSStatus ExamineContext(AEDesc* inContextDescriptor,
                    SInt32 inTimeOutInTicks
                    AEDescList* ioCommandPairs,
                    Boolean* outNeedMoreTime);
    OSStatus HandleSelection(AEDesc* inContextDescriptor, SInt32
                    inCommandID);
    OSStatus PostMenuCleanup(void);
}
```

When writing your own contextual menu plug-in, you must follow this declaration and include the specified methods. The following sections describe these methods in detail.

Initialize

Performs any required plug-in initialization. If you write a contextual menu plug-in, you may include an `Initialize` method with the following form:

```
OSStatus Initialize (FSSpec *inFileSpec);
```

`inFileSpec` A pointer to a file system specification record for the file that contains the plug-in.

method result A result code. See “Result Codes” (page 61) for a list of possible values. If this value is not `noErr` then the Menu Manager does not use the plug-in.

DISCUSSION

The `Initialize` method is called when the Menu Manager builds its registry of available plug-ins (typically at system startup). You should use the `Initialize`

method to check for available resources before the plug-in is actually required. To maintain a small memory footprint, the `Initialize` method should not allocate any memory, buffers, or so on. Instead, you should allocate memory as needed when examining the context or acting on the selection.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

ExamineContext

Examines the context chosen by the user and determines possible menu commands appropriate to the context. If you write a contextual menu plug-in, it must contain an `ExamineContext` method with the following form:

```
OSStatus ExamineContext (AEDesc* inContextDescriptor,
                    SInt32 inTimeOutInTicks,
                    AEDescList* ioCommandPairs,
                    Boolean* outNeedMoreTime);
```

`inContextDescriptor`
The context chosen by the user. The Menu Manager passes this in the form of a pointer to an Apple Event descriptor. See *Inside Macintosh: Interapplication Communication* for information about the form of this descriptor. If there is no selection to examine, the pointer is `NULL`.

`inTimeOutInTicks`
The amount of time the plug-in is allowed to examine the context and create menu items.

`ioCommandPairs`
A pointer to an Apple Event descriptor list containing the commands allowed for this context.

`outNeedMoreTime`
Not currently used.

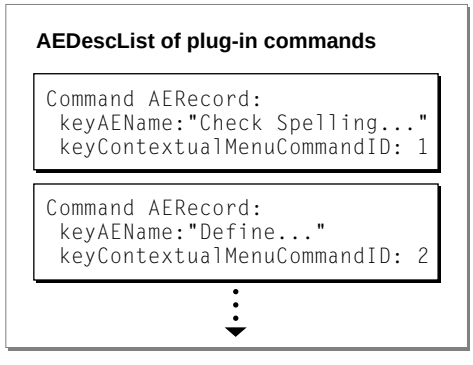
method result A result code. See “Result Codes” (page 61) for a list of possible values. If this value is not `noErr` then the Menu Manager does not use the plug-in in this case. However, it will call `ExamineContext` again the next time a contextual menu is invoked.

DISCUSSION

When the user activates a contextual menu, each module in the registry has its `ExamineContext` method called so it can inspect the context and add menu items as appropriate. After examining the context, the plug-in should then fill the `AEDescList` array with every command that it wants to add to the menu. This `AEDescList` will be created and disposed of for the plug-in; it will be empty when the plug-in receives it.

Each menu command that the plug-in can perform on the selection is described in an `AERecord` with two keyword-specified descriptor records. The structure of the `AERecord` is shown in Figure 1-1.

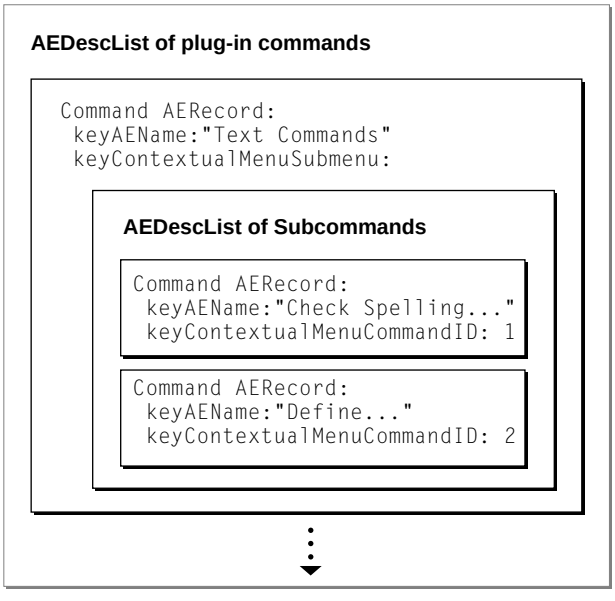
Figure 1-1 A menu command list in the `AEDescList` array



The first descriptor (`keyAEName`) is of `typeIntlText` and contains the text of the menu item to be added to the menu. The second descriptor (`keyContextualMenuCommandID`) is of type `typeLongInteger` and must contain a value specific to the plug-in that uniquely identifies this menu item.

If a plug-in wants to display a submenu for a particular menu item, it must use a variation of the `AERecord` used to describe a normal menu item. Figure 1-2 shows this variation.

Figure 1-2 A menu record showing submenus



The first descriptor (`keyAEName`) is the same, but the second descriptor uses a different keyword (`keyContextualMenuSubmenu`) and is of type `typeAEList`. It must contain an `AEDescList` with an `AERecord` for every command to be added to the submenu. Submenu items can themselves have submenus by recursively using this technique. The depth of the submenus is limited only by the constraints of the Menu Manager.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

HandleSelection

Executes the contextual menu item chosen by the user. If you write a contextual menu plug-in, it must contain a `HandleSelection` method with the following form:

```
OSStatus HandleSelection(AEDesc* inContextDescriptor,
                         SInt32 inCommandID);
```

`inContextDescriptor`
: The context chosen by the user. This data is the same as that passed in the `inContextDescriptor` parameter for the `ExamineContext` method. The Menu Manager passes this in the form of a pointer to an Apple Event descriptor. See *Inside Macintosh: Interapplication Communication* for information about the form of this descriptor.

`inCommandID`
: A long integer assigned to the chosen menu item via the `keyContextualMenuCommandID` descriptor, passed in by the Menu Manager.

method result
: A result code. See "Result Codes" (page 61) for a list of possible values. If this value is not `noErr` then the Menu Manager does not use the plug-in in this case. However, it will call `HandleSelection` again the next time an action is selected.

DISCUSSION

If one of the plug-in's menu items is chosen, the Menu Manager calls the plug-in's `HandleSelection` method to execute the action. The plug-in should then perform the appropriate action.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

PostMenuCleanup

Performs any necessary cleanup when the contextual menu is dismissed. If you write a contextual menu plug-in, it must contain a `PostMenuCleanup` method with the following form:

```
OSStatus PostMenuCleanup(void);
```

method result A result code. See "Result Codes" (page 61) for a list of possible values.

DISCUSSION

When a contextual menu is dismissed (regardless of whether or not the user made a selection), the Menu Manager calls each plug-in's `PostMenuCleanup` method. The `PostMenuCleanup` method should do any necessary cleanup or memory deallocation. For example, a plug-in that allocated a buffer in the `ExamineContext` method should dispose of that buffer when `PostMenuCleanup` is called.

VERSION NOTES

Available with Appearance Manager 1.0 and later.

Menu Manager Data Types

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

- `MCEntry` (page 44)
- `'mctb'` (page 44)
- `'MENU'` (page 44)
- `'xmnu'` (page 50)

MCEntry

The menu color information table defines the standard color for the menu bar, menu titles, menu items, and the background color of a displayed menu. If you do not add any menu color entries to this table, the Menu Manager draws your menus using the current default colors. Using the menu color information table to define custom colors for your menus is not recommended with Appearance Manager 1.0 and later.

When the Appearance Manager is available and you are using standard menus, if you do not include a menu bar entry in your menu color information table, only the menu title color and menu item text color values from menu color entries are used. If you do include a menu bar entry in your menu color information table, all menu colors are used, and the menus revert to a standard System 7 appearance.

If you are creating your own custom menu definition function, all entries in the table are used.

'mctb'

The menu color information table ('mctb') resource defines the standard color for the menu bar, menu titles, menu items, and the background color of a displayed menu. If you do not add any menu color entries to this resource, the Menu Manager draws your menus using the current default colors. Using the menu color information table resource to define custom colors for your menus is not recommended with Appearance Manager 1.0 and later.

When the Appearance Manager is available and you are using standard menus, if you do not include a menu bar entry in your menu color information table resource, only the menu title color and menu item text color values from menu color entries are used. If you do include a menu bar entry in your menu color information table resource, all menu colors are used, and the menus revert to a standard System 7 appearance.

If you are creating your own custom menu definition function, all entries in the menu color information table resource are used.

'MENU'

A menu `('MENU')` resource describes the initial characteristics of a menu and its menu items. With Appearance Manager 1.0,the recommended usage for certain

fields of the variable-length data portion of the menu resource has been changed, and the fields may take on different values, depending on the type of information you wish to display with your menu item. However, the format of the resource remains identical.

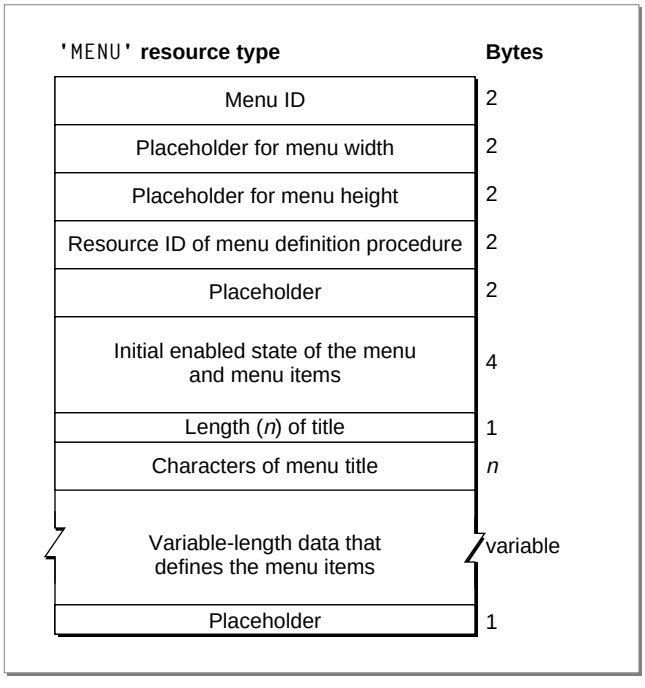
You can provide descriptions of your menus in menu resources and use the functions `GetMenu` (page 13) or `GetNewMBar` to read the descriptions of your menus. After reading in the resource description, the Menu Manager stores the information about specific menus in menu structures. When you use a menu resource to define a menu, you should check for the presence of an extended menu resource with the same resource ID.

▲ **WARNING**

Menus in a resource must not be purgeable nor should they have the resource lock bit set. They must have resource ID numbers greater than 127. Do not define a “circular” hierarchical menu—that is, a hierarchical menu in which a submenu has a submenu whose submenu is a hierarchical menu higher in the chain.

Figure 1-3 shows the format of a compiled `'MENU'` resource.

Figure 1-3 Structure of a compiled menu (`'MENU'`) resource



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

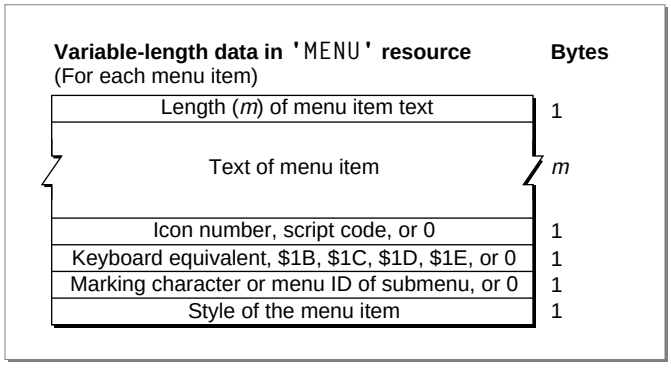
- Menu ID. Each menu in your application should have a unique menu ID (this can be the menu's resource ID). A negative value indicates that the menu (but not a submenu) belongs to a driver such as a desk accessory. A menu ID from 1 through 235 indicates a menu (or submenu) of an application; a menu ID from 236 through 255 indicates a submenu of a driver. Apple reserves the menu ID of 0.
- Placeholder (two integers containing 0) for the menu's width and height. After reading in the resource data, the Menu Manager requests the menu definition function to calculate the width and height of the menu and to store these values in the `menuWidth` and `menuHeight` fields of the menu structure.
- Resource ID of the menu's menu definition function; see "Menu Definition IDs" (page 56). If the integer 63 appears here, as specified by the `kMenuStdMenuProc` constant in the Rez input file, the Menu Manager uses the

standard Appearance-compliant menu definition function to manage the menu. If you provide your own menu definition function, its resource ID should appear in this field. After reading in the menu's resource data, the Menu Manager reads in the menu definition function, if necessary. The Menu Manager stores a handle to the menu definition function in the `menuProc` field of the menu structure.

- Placeholder (an integer containing 0).
- The initial enabled state of the menu and first 31 menu items. This is a 32-bit value, where bits 1–31 indicate if the corresponding menu item is disabled or enabled, and bit 0 indicates whether the menu as a whole is enabled or disabled. The Menu Manager automatically enables menu items greater than 31 when a menu is created.
- The length (in bytes) of the menu title.
- The title of the menu.
- Variable-length data that describes the menu items. If you provide your own menu definition function you can define and provide this data according to the needs of your function. The Menu Manager simply reads in the data for each menu item and stores it as variable data at the end of the menu structure. The menu definition function is responsible for interpreting the contents of the data. For example, the standard menu definition function interprets this data according to the description given in the following paragraphs.
- Placeholder (a byte containing 0) to indicate the end of the menu item definitions.

Figure 1-4 shows the variable-length data portion of a compiled `'MENU'` resource that uses the standard menu definition function.

Figure 1-4 Variable-length data portion of a compiled 'MENU' resource



For the standard menu definition function, the variable-length data contains the following elements:

- Length (in bytes) of the menu item's text.
- Text of the menu item.
- A 1-byte field containing one of the following:
  - An icon number. The icon number is a number from 1 through 255 (or from 1 through 254 for small or reduced icons). The Menu Manager adds 256 to the icon number to generate the resource ID of the menu item's icon. If a menu item has an icon, you should also provide a `'cicn'`, `'SICN'`, or an `'ICON'` resource with the resource ID equal to the icon number plus 256. If you want the Menu Manager to reduce an `'ICON'` resource to the size of a small icon, you must also provide the value 0x1D in the keyboard equivalent field. If you provide a `'SICN'` resource, provide 0x1E in the keyboard equivalent field. Otherwise, the Menu Manager looks first for a `'cicn'` resource with the calculated resource ID and uses that icon.
  - A text encoding value. (Not recommended with Appearance.) If you want the Menu Manager to draw the item's text in a script other than the system script, specify the text encoding here and also provide 0x1C in the keyboard equivalent field. If the script system for the specified script is installed, the Menu Manager draws the item's text using that script.
  - 0 (as specified by the `noicon` constant in a Rez input file) if the menu item doesn't contain an icon and uses the system script.

A menu item can have an icon or be drawn in a script other than the system script, but not both.

- Keyboard equivalent (specified as a 1-byte character). This can be enhanced with modifier key constants in the modifier keys field of the extended menu resource; see "'xmnu'" (page 50). In some cases, this field may take on one of the following values instead:
  - 0x1B (as specified by the constant `hierarchicalMenu` in a Rez input file) if the item has a submenu. (Not recommended with Appearance.)
  - 0x1C if the item uses a script other than the system script. (Not recommended with Appearance.)
  - 0x1D if you want the Menu Manager to reduce an `'ICON'` resource to the size of a small icon.
  - 0x1E if you want the Menu Manager to use an `'SICN'` resource for the item's icon.
  - 0 (as specified by the `nokey` constant in a Rez input file) if the item has neither a keyboard equivalent nor a submenu and uses the system script.

  The values 0x01 through 0x1A as well as 0x1F and 0x20 are reserved for use by Apple; your application should not use any of these reserved values in this field.
- A 1-byte field containing one of the following:
  - A marking character. Special marking characters are available to indicate the marks associated with a menu item.
  - The menu ID of the item's submenu. (Not recommended with Appearance.) Submenus of an application should have menu IDs from 1 through 235; submenus of a driver (such as a desk accessory) should have menu IDs from 236 through 255. If you choose a submenu, you must also set the keyboard equivalent field to 0x1B.
  - 0 (as specified by the `nomark` constant in a Rez input file) if the item has neither a mark nor a submenu.

  A menu item can have a mark or a submenu, but not both.
- Font style of the menu item. The constants `bold`, `italic`, `plain`, `outline`, and `shadow` can be used in a Rez input file to define their corresponding styles.

If you provide your own menu definition function, you should use the same format for your resource descriptions of menus as shown in Figure 1-3. You can use the same format or one of your choosing to describe menu items. You can

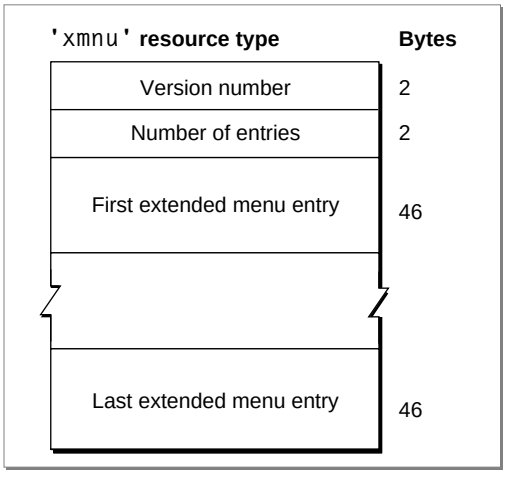
also use bits 1–31 of the `enableFlags` field of the menu structure as you choose; however, bit 0 must still indicate whether the menu is enabled or disabled.

'xmnu'

You can use the extended menu (`'xmnu'`) resource to create menus with modifier key keyboard glyphs and icons attached to menu items and Appearance-compliant menu backgrounds. The extended menu resource is available with Appearance Manager 1.0 and later.

After reading in a `'MENU'` resource, `GetMenu` (page 13) looks for an extended menu resource of type `'xmnu'` with the same resource ID. The information is set for specified menu items; it is not necessary to create an extended menu entry for each item. At this point, the information can be purged or released. Figure 1-5 shows the format of a compiled `'xmnu'` resource.

Figure 1-5 Structure of a compiled extended menu (`'xmnu'`) resource



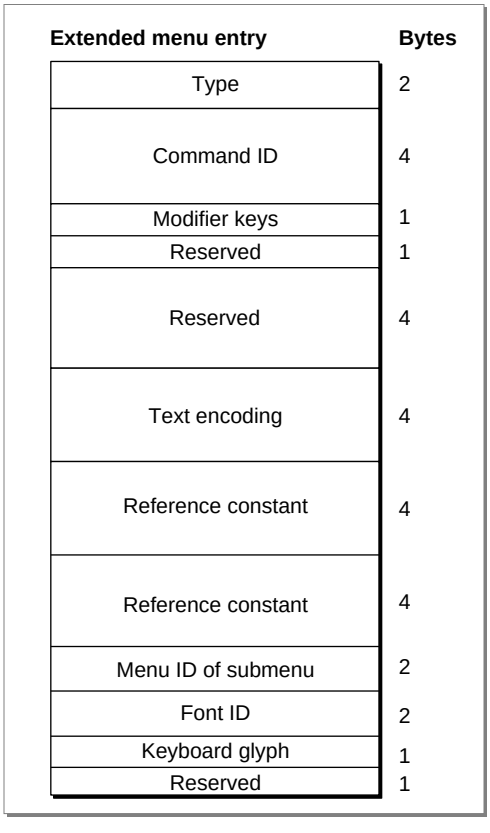
A compiled version of an `'xmnu'` resource contains the following elements:

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

- Number of entries. An integer that specifies the number of entries in the resource. Each entry is an extended menu item structure.
- Extended menu item entries. A series of extended menu item structures, each of which consists of a type, command ID, modifier keys, text encoding, reference constants, menu ID of submenu, font ID, and keyboard glyph.

Figure 1-6 shows the format of an extended menu item entry.

Figure 1-6 Structure of an extended menu item entry



Each entry in a `'xmnu'` resource corresponds to a menu item and contains the following:

- Type. An integer that specifies whether there is menu item information for the item in the `'MENU'` entry. If this is 0, there is no information for the item in the corresponding `'MENU'` entry, and the rest of the record is skipped. If this is 1, there is information for the item in the corresponding `'MENU'` entry, and the rest of the record is read.
- Command ID. A unique value which you set to identify the menu item (instead of referring to it using the menu ID and item number). You can also call `SetMenuItemCommandID` (page 22) to set the command ID of a menu item. After a successful call to `MenuSelect`, `MenuEvent` (page 15), or `MenuKey` (page 17), you can call `GetMenuItemCommandID` (page 23) to determine its current value.
- Modifier keys. A mask that determines which modifier keys are used in a keyboard equivalent to select a menu item; see "Modifier Key Mask Constants" (page 58).
- Reserved. Set to 0.
- Reserved. Set to 0.
- Text encoding. A long integer which indicates the text encoding which your item text will use. Use `currScript` for the default text encoding. To change this value, call `SetMenuItemTextEncoding` (page 35). You can call `GetMenuItemTextEncoding` (page 36) to determine its current value. This should be used instead of setting a menu item's modifier key to 0x1C and its icon ID to the script code.
  - If you wish the text of the menu item to use the system script, this value should be -1. This should be used as the default.
  - If you wish the text of the menu item to use the current script, this value should be -2.
- Reference constant. Any value that an application wishes to store. To change this value, call `SetMenuItemRefCon` (page 32). You can call `GetMenuItemRefCon` (page 32) to determine its current value.
- Reference constant. Any additional value that an application wishes to store. To change this value, call `SetMenuItemRefCon2` (page 33). You can call `GetMenuItemRefCon2` (page 34) to determine its current value.
- Menu ID of submenu. A value between 1 and 235, identifying the application submenu.
- Font ID. An integer representing the ID of the font family. If this value is 0, then the system font ID is used.

- Keyboard glyph. A symbol representing a menu item's modifier key. In Appearance 1.0, if the value in this field is zero, the keyboard glyph uses the system font. In Appearance 1.0.1, if the value in this field is zero, the keyboard glyph uses the keyboard font; see Table 1-1 (page 54). Use of the keyboard font (rather than the system font) provides a consistent user interface across applications, since a modifier key's symbol will not change regardless of what system font is running. If the value in this field is nonzero, you can override the character code to be displayed with a substitute glyph.
- Reserved. Set to 0.

Table 1-1 Keyboard font character codes

Character code	Description
0x00	Null (always glyph 1)
0x01	Unassigned (reserved for 2 bytes)
0x02	Tab to the right key (for left-to-right script systems)
0x03	Tab to the left key (for right-to-left script systems)
0x04	Enter key
0x05	Shift key
0x06	Control key
0x07	Option key
0x08	Null (always glyph 1)
0x09	Space (always glyph 3) key
0x0A	Delete to the right key (for right-to-left script systems)
0x0B	Return key (for left-to-right script systems)
0x0C	Return key (for right-to-left script systems)
0x0D	Nonmarking return key
0x0E	Unassigned
0x0F	Pencil key
0x10	Downward dashed arrow key
0x11	Command key
0x12	Checkmark key
0x13	Diamond key
0x14	Apple logo key (filled)
0x15	Unassigned (paragraph in Korean)
0x16	Unassigned
0x17	Delete to the left key (for left-to-right script systems)
0x18	Leftward dashed arrow key
0x19	Upward dashed arrow key
0x1A	Rightward dashed arrow key

Character code	Description
0x1B	Escape key
0x1C	Clear key
0x1D	Unassigned (left double quotes in Japanese)
0x1E	Unassigned (right double quotes in Japanese)
0x1F	Unassigned (trademark in Japanese)
0x61	Blank key
0x62	Page up key
0x63	Caps lock key
0x64	Left arrow key
0x65	Right arrow key
0x66	Northwest arrow key
0x67	Help key
0x68	Up arrow key
0x69	Southeast arrow key
0x6A	Down arrow key
0x6B	Page down key
0x6C	Apple logo key (outline)
0x6D	Contextual menu key
0x6E	Power key
0x6F	F1 key
0x70	F2 key
0x71	F3 key
0x72	F4 key
0x73	F5 key
0x74	F6 key
0x75	F7 key
0x76	F8 key
0x77	F9 key
0x78	F10 key
0x79	F11 key

Character code	Description
0x7A	F12 key
0x87	F13 key
0x88	F14 key
0x89	F15 key
0x8A	Control key (ISO standard)

Menu Manager Constants

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

- “Menu Definition IDs” (page 56)
- “Contextual Menu Help Type Constants” (page 57)
- “Contextual Menu Selection Type Constants” (page 58)
- “Modifier Key Mask Constants” (page 58)
- “Menu Item Icon Type Constants” (page 59)

Menu Definition IDs

A menu definition ID is supplied to the menu resource (page 44) or a menu-creation function such as `NewMenu` to specify which menu definition function to use in creating the menu. The menu definition ID contains the resource ID of the menu definition function.

Menu definition IDs are changed with Appearance Manager 1.0 to support Appearance-compliant menus and menu bars. When mapping is enabled, the pre-Appearance menu definition ID `textmenuProc` will be mapped to `kMenuStdMenuProc`, its Appearance-compliant equivalent.

```
enum {
    textmenuProc        = 0,
    kMenuStdMenuProc    = 63,
    kMenuStdMenuBarProc = 63
};
```

Constant descriptions

`textmenuProc` The menu definition ID for menus that are not Appearance-compliant.

`kMenuStdMenuProc` The menu definition ID for Appearance-compliant menus. Available with Appearance Manager 1.0 and later.

`kMenuStdMenuBarProc` The menu bar definition ID for Appearance-compliant menu bars. Available with Appearance Manager 1.0 and later.

Contextual Menu Help Type Constants

You can pass the following constants in the `inHelpType` parameter of the function `ContextualMenuSelect` (page 18) to specify the kind of help the application supports. Contextual menu help type constants are available with Appearance Manager 1.0 and later.

```
enum{
    kCMHelpItemNoHelp       = 0,
    kCMHelpItemAppleGuide   = 1,
    kCMHelpItemOtherHelp    = 2
};
```

Constant descriptions

`kCMHelpItemNoHelp` The application does not support any help. The Menu Manager will put an appropriate help string into the menu and disable it.

`kCMHelpItemAppleGuide` The application supports Apple Guide help. The Menu Manager will put the name of the main Guide file into the menu and enable it.

`kCMHelpItemOtherHelp` The application supports some other form of help. In this

case, the application must also pass a valid string into the `inHelpItemString` parameter of `ContextualMenuSelect`. This string will be the text of the help item in the menu, and the help item will be enabled.

Contextual Menu Selection Type Constants

The following constants are returned in the `outUserSelectionType` parameter of the function `ContextualMenuSelect` (page 18) to specify what the user selected from the contextual menu. Contextual menu selection type constants are available with Appearance Manager 1.0 and later.

```
enum{
    kCMNothingSelected      = 0,
    kCMMenuItemSelected     = 1,
    kCMShowHelpSelected     = 3
};
```

Constant descriptions

`kCMNothingSelected` The user did not choose an item from the contextual menu and the application should do no further processing of the event.

`kCMMenuItemSelected`
The user chose one of the application's items from the menu. The application can examine the `outMenuID` and `outMenuItem` parameters of `ContextualMenuSelect` to see what the menu selection was, and it should then handle the selection appropriately.

`kCMShowHelpSelected`
The user chose the Help item from the menu. The application should open an Apple Guide database to a section appropriate for the selection. If the application supports some other form of help, it should be presented instead.

Modifier Key Mask Constants

You can use one or more of the following mask constants in the modifier keys field of the `'xmnu'` resource (page 50) to determine which modifier key(s) must

be pressed along with a character key to create a keyboard equivalent for selecting a menu item. These constants are also passed in and obtained by `SetMenuItemModifiers` (page 30) and `GetMenuItemModifiers` (page 31), respectively. Modifier key mask constants are available with Appearance Manager 1.0 and later.

```
enum {
    kMenuNoModifiers        = 0,
    kMenuShiftModifier      = (1 << 0),
    kMenuOptionModifier     = (1 << 1),
    kMenuControlModifier    = (1 << 2),
    kMenuNoCommandModifier  = (1 << 3)
};
```

Constant descriptions

`kMenuNoModifiers` If no bit is set, only the Command key is used in the keyboard equivalent.

`kMenuShiftModifier` If this bit (bit 0) is set, the Shift key is used in the keyboard equivalent.

`kMenuOptionModifier`
If this bit (bit 1) is set, the Option key is used in the keyboard equivalent.

`kMenuControlModifier`
If this bit (bit 2) is set, the Control key is used in the keyboard equivalent.

`kMenuNoCommandModifier`
If this bit (bit 3) is set, the Command key is not used in the keyboard equivalent.

Menu Item Icon Type Constants

The following constants specify the type of an icon attached to a menu item. They are passed in `SetMenuItemIconHandle` (page 26) and obtained by `GetMenuItemIconHandle` (page 27). Menu item icon type constants are available with Appearance Manager 1.0 and later.

```
enum {
    kMenuNoIcon         = 0,
    kMenuIconType       = 1,
```

```
    kMenuShrinkIconType = 2,
    kMenuSmallIconType  = 3,
    kMenuColorIconType  = 4,
    kMenuIconSuiteType  = 5,
    kMenuIconRefType    = 6
};
```

Constant descriptions

`kMenuNoIcon` No icon.

`kMenuIconType` Identifies an icon of type `'ICON'`.

`kMenuShrinkIconType` Identifies a 32-by-32-pixel icon of type `'ICON'`, shrunk (at display time) to 16-by-16.

`kMenuSmallIconType` Identifies an icon of type `'SICN'`.

`kMenuColorIconType` Identifies an icon of type `'cicn'`.

`kMenuIconSuiteType` Identifies an icon suite.

`kMenuIconRefType` Identifies an icon of type `IconRef`. This value is supported under Mac OS 8.5 and later.

Menu Definition Message and Feature Constants

With Appearance Manager 1.0, the Menu Manager may pass a new constant in the `message` parameter of your menu definition function to specify the action that your function must perform. In response, your menu definition function may report a new feature flag in the `whichItem` parameter.

```
enum {
    kMenuThemeSavvyMsg      = 7,
    kThemeSavvyMenuResponse = 0x7473
};
```

Constant descriptions

`kMenuThemeSavvyMsg` Identify whether your menu definition function is theme-compliant. If so, your menu definition function should respond by passing back `kThemeSavvyMenuResponse` in the `whichItem` parameter. The Menu Manager then draws the menu background as appropriate for the current theme.

`kThemeSavvyMenuResponse`
: If this bit is set, the menu supports the `kMenuThemeSavvyMsg` message.

Result Codes

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

`noErr`	0	No error
`paramErr`	–50	Error in parameter list
`memFullErr`	–108	Not enough memory
`resNotFound`	–192	Unable to read resource
`hmHelpManagerNotInited`	–855	Help manager not set up

Version History

This document has had the following releases:

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

Version	Notes
Nov. 16, 1998	Removed "Menu Manager Reference" chapter from the *Mac OS 8 Toolbox Reference* document. *Mac OS 8 Menu Manager Reference* is now available as an independent document. The following corrections were made: "Contextual Menu Gestalt Selector Constants" (page 9). Removed unsupported value `gestaltContextualMenuPresent`. `SetItemMark`. Recategorized from "not recommended with the Appearance Manager" to "unchanged" and, therefore, removed from this delta document. `SetMenuItemHierarchicalID` (page 25). Noted that menu IDs >255 are supported under Mac OS 8.5 and later. `SetMenuItemIconHandle` (page 26). Corrected type of `inIconType` parameter to be `UInt8`. `GetMenuItemIconHandle` (page 27). Corrected type of `outIconType` parameter to be `UInt8*`. Corrected description of `outIconHandle` parameter to specify that `nil` is only produced when no icon is attached to the menu item. `SetMenuItemModifiers` (page 30). Corrected type of `inModifiers` parameter to be `UInt8`. `GetMenuItemModifiers` (page 31). Corrected type of `outModifiers` parameter to be `UInt8*`. `SetMenuItemRefCon` (page 32). Corrected type of `inRefCon` parameter to be `UInt32`. `GetMenuItemRefCon` (page 32). Corrected type of `outRefCon` parameter to be `UInt32`.

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

Version	Notes
Nov. 16, 1998 (continued)	`SetMenuItemRefCon2` (page 33). Corrected type of `inRefCon` parameter to be `UInt32`.
	`GetMenuItemRefCon2` (page 34). Corrected type of `outRefCon` parameter to be `UInt32`.
	`MCEntry` (page 44) and `'mctb'` (page 44). Expanded description of menu color table behavior with the Appearance Manager.
	"Modifier Key Mask Constants" (page 58). Corrected the constant name for value 0; was `kMenuCommandModifiers`, should have been `kMenuNoModifiers`.
	"Menu Item Icon Type Constants" (page 59). Renamed from "Menu Icon Handle Constants". Added missing constant, `kMenuIconRefType`.
	"Menu Definition Message and Feature Constants" (page 60). Added a previously undocumented message and feature flag.
Jan. 15, 1998	The following corrections were made:
	Noted Appearance 1.0.2 where applicable.
Dec. 10, 1997	The following corrections were made:
	"Keyboard font character codes" (page 54). Clarified the descriptions of those character codes that differ in right-to-left and left-to-right script systems.
Dec. 2, 1997	PDF formatting improved.
Nov. 3, 1997	First document release.

Index

M

P

R

S

T

X

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

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

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

ILLUSTRATORS
David Arrigoni and Karin Stroud

PRODUCTION EDITOR
Glen Frank

PROJECT MANAGER
Tony Francis

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