Here is the structure of a menu record:

```
TYPE  MenuInfo =                {menu record}
      RECORD
         menuID:      Integer;   {number that identifies the menu}
         menuWidth:   Integer;   {width (in pixels) of the menu}
         menuHeight:  Integer;   {height (in pixels) of the menu}
         menuProc:    Handle;    {menu definition procedure}
         enableFlags: LongInt;   {indicates whether menu and }
                                 { menu items are enabled}
         menuData:    Str255;    {title of menu}
         {itemDefinitions}       {variable-length data that }
                                 { defines the menu items}
      END;
```

Field descriptions

`menuID`	A number that identifies the menu. Each menu in your application must have a unique menu ID. Your application specifies the menu ID when you create the menu. Thereafter you can use the menu ID and the `GetMenuHandle` function to get a handle to the menu's menu record. When you define hierarchical menus, you must use a number from 1 through 235 for the menu ID of a submenu of an application; use a number from 236 through 255 for the submenu of a desk accessory.
`menuWidth`	The horizontal dimensions of the menu, in pixels.
`menuHeight`	The vertical dimensions of the menu, in pixels.
`menuProc`	A handle to the menu definition procedure of the menu. The Menu Manager uses this menu definition procedure to draw the menu.
`enableFlags`	A value that represents the enabled state of the menu title and the first 31 items in the menu. All menu items greater than 31 are enabled by default and can be disabled only by disabling the entire menu.
`menuData`	A string that defines the title of the menu. Although the `menuData` field is defined by the data type `Str255` in the `MenuInfo` data structure, the Menu Manager allocates only the storage necessary for the title: the number of characters in the title of the string plus 1.
`itemDefinitions`	Variable-length data that defines the characteristics of each menu item in the menu. If the menu uses the standard menu definition procedure, this data can be conceptually defined in this manner:

```
itemData: ARRAY[1..X] OF
   itemString:  String; {text of menu item}
   itemIcon:    Byte;   {icon number minus 256}
```

```
         itemCmd:     Char;   {keyboard equivalent or }
                              { value ($1B) indicating }
                              { item has a submenu, or }
                              { ($1C) if item has }
                              { a script code, or }
                              { ($1D) if item's 'ICON' }
                              { should be reduced, or }
                              { ($1E) if item has an }
                              { 'SICN' icon}
         itemMark:    Char;   {marking character or }
                              { menu ID of submenu}
         itemStyle:   Style;  {style of menu text}
      endMarker:      Byte;   {contains 0 if no }
                              { more menu items}
```

The menu definition procedure maintains the information about the menu items. You typically define your menu items in `'MENU'` resources, and the Menu Manager stores information describing your items in the menu's menu record.

Your application should not directly change the values of any fields in a menu record. Use Menu Manager routines to change the characteristics of menu items or to make other changes to a menu.

The Menu List

The menu list contains information about the menus in a menu bar, about submenus, and about pop-up menus. A menu list contains handles to the menu records of zero, one, or more menus and contains other information that the Menu Manager uses to manage menus.

The `InitMenus` procedure creates the current menu list of an application. The current menu list contains handles to the menu records of all menus currently in the menu bar and handles to the menu records of any submenus or pop-up menus inserted into the menu list by your application. The menu bar shows the titles, in order, of all menus (other than submenus or pop-up menus) in the menu list.

The initial menu list created by `InitMenus` does not contain handles to any menus. The Menu Manager dynamically allocates storage in a menu list as menus are added to and deleted from the menu list.

Your application should not directly change or access the information in a menu list. You should use Menu Manager routines to create a menu list and to add menus to or remove menus from the current menu list.

You typically define your application's menu bar in an `'MBAR'` resource and create a menu list using the `GetNewMBar` function. The `GetNewMBar` function returns a handle to a menu list. You can set the current menu list to the menu list returned by `GetNewMBar` using the `SetMenuBar` procedure.

The structure of the menu list is private to the Menu Manager. For conceptual purposes, however, its general structure is defined here.

```
TYPE  DynamicMenuList =
      RECORD
         lastMenu:   Integer;    {offset to last pull-down menu}
         lastRight:  Integer;    {pixel location of right edge }
                                 { of rightmost menu in menu bar}
         mbResID:    Integer;    {upper 13 bits are the resource ID of menu }
                                 { bar defn function, low 3 bits the variant}
         menu:       ARRAY[1..X] {variable array with one record for }
                     OF MenuRec; { each menu}
         lastHMenu:  Integer;    {offset to last submenu or pop-up menu}
         menuTitleSave:          {handle to bits behind inverted menu title}
                     pixMapHandle;
         hMenu:      ARRAY[1..Y] {variable array with one record for }
                     OF HMenuRec;{ each submenu or pop-up menu}
      END;
```

The Menu Manager dynamically allocates the records that contain handles to the menu records of menus in the menu bar, submenus, and pop-up menus. These records can be defined conceptually as the `MenuRec` and `HMenuRec` data types. The Menu Manager uses a data structure similar to that of the `MenuRec` data type to store information about pull-down menus in the menu list.

```
TYPE  MenuRec =
      RECORD
         menuOH:     MenuHandle; {handle to menu's menu record}
         menuLeft:   Integer;    {pixel location of left edge }
                                 { of this menu}
      END;
```

The Menu Manager stores information about submenus and pop-up menus at the end of a menu list in a data structure similar to that of the `HMenuRec` data type.

```
TYPE  HMenuRec =
      RECORD
         menuHOH:    MenuHandle; {handle to menu's menu record}
         reserved:   Integer;    {reserved}
      END;
```

The Menu Color Information Table Record

Your application's **menu color information table** defines the standard color for the menu bar, titles of menus, text and characteristics of menu items, and background color of a displayed menu. If you do not add any entries to this table, the Menu Manager draws your menus using the default colors, black on white. You can add colors to your menus by adding entries to your application's menu color information table by using Menu Manager routines or by defining these entries in an 'mctb' resource. Note that the menu color information table uses a format that is different from the standard color table format.

The Menu Manager maintains information about an application's menu color information table as an array of menu color entry records.

```
TYPE  MCTable = ARRAY[0..0] OF MCEntry;        {menu color table}
      MCTablePtr = ^MCTable;      {pointer to a menu color table}
      MCTableHandle = ^MCTablePtr;{handle to a menu color table}
```

A menu color entry is defined by the MCEntry data type.

```
TYPE  MCEntry =                    {menu color entry}
      RECORD
         mctID:      Integer;      {menu ID or 0 for menu bar}
         mctItem:    Integer;      {menu item number or 0 for }
                                   { menu title}
         mctRGB1:    RGBColor;     {usage depends on mctID and }
                                   { mctItem}
         mctRGB2:    RGBColor;     {usage depends on mctID and }
                                   { mctItem}
         mctRGB3:    RGBColor;     {usage depends on mctID and }
                                   { mctItem}
         mctRGB4:    RGBColor;     {usage depends on mctID and }
                                   { mctItem}
         mctreserved:Integer;      {reserved}
      END;

      MCEntryPtr = ^MCEntry;       {pointer to a menu color entry}
```

The first two fields of a menu color entry record, mctID and mctItem, define whether the entry is a menu bar entry, a menu title entry, or a menu item entry. The following four fields specify color information for whatever type of entry the mctID and mctItem fields describe. The value of the mctID field in the last entry in a menu color information table is –99, and the rest of the fields of the last entry are reserved. The Menu Manager automatically creates the last entry in a menu color information table; your application should not use the value –99 as the menu ID of a menu if you wish to add a menu color entry for it.

The Menu Manager creates your application's menu color information table the first time your application calls InitMenus or InitProcMenu. It creates the menu color information table as initially empty except for the last entry, which indicates the end of the table.

Table 3-7 shows how the Menu Manager interprets the `mctID` and `mctItem` fields for each type of menu color entry in a menu color information table.

Table 3-7 Color information for menu entries

	ID	Item	RGB1	RGB2	RGB3	RGB4
Menu bar	0	0	Default menu title color	Default background color of menus	Default item color	Default bar color
Menu title	*N*<>0	0	Menu title color	Bar color	Default item color	Background color of menu
Menu item	*N*<>0	*M*<>0	Mark color	Item text color	Keyboard equivalent color	Background color of menu
Last entry	–99	Reserved	Reserved	Reserved	Reserved	Reserved

A **menu bar entry** is defined by a menu color entry record that contains 0 in both the `mctID` and `mctItem` fields. You can define only one menu bar entry in a menu color information table. If you don't provide a menu bar entry for your application's menu color information table, the Menu Manager uses the standard menu bar colors (black text on a white background), and it uses the standard colors for the other menu elements. You can provide a menu bar entry to specify default colors for the menu title, the background of a displayed menu, the items in a menu, and the menu bar. The color information fields for a menu bar entry are interpreted as follows:

- `mctRGB1` specifies the default color for menu titles. If a menu doesn't have a menu title entry, the Menu Manager uses the value in this field as the color of the menu title.
- `mctRGB2` specifies the default color for the background of a displayed menu. If a menu doesn't have a menu title entry, the Menu Manager uses the value in this field as the color of the menu's background when it is displayed.
- `mctRGB3` specifies the default color for the items in a displayed menu. If a menu item doesn't have a menu item entry or a default color defined in a menu title entry, the Menu Manager uses the value in this field as the color of the menu item.
- `mctRGB4` specifies the default color for the menu bar. If a menu doesn't have a menu bar entry (and doesn't have any menu title entries), the Menu Manager uses the standard colors for the menu bar.

A **menu title entry** is defined by a menu color entry record that contains a menu ID in the `mctID` field and 0 in the `mctItem` field. You can define only one menu title entry for each menu. If you don't provide a menu title entry for a menu in your application's menu color information table, the Menu Manager uses the colors defined by the menu bar entry. If a menu bar entry doesn't exist, the Menu Manager uses the standard colors (black on white). You can provide a menu title entry to specify a color for the title and background of a specific menu and a default color for its items. The color information fields for a menu title entry are interpreted as follows:

- `mctRGB1` specifies the color for the menu title of the specified menu. If a menu doesn't have a menu title entry, the Menu Manager uses the default value defined in the menu bar entry.
- `mctRGB2` specifies the default color for the menu bar. If a menu color information table doesn't have a menu bar entry, the Menu Manager uses the value in this field as the color of the menu bar. If a menu bar entry already exists, the Menu Manager replaces the value in the `mctRGB2` field of the menu title entry with the value defined in the `mctRGB4` field of the menu bar entry.
- `mctRGB3` specifies the default color for the items in the menu. If a menu item doesn't have a menu item entry or a default color defined in a menu bar entry, the Menu Manager uses the value in this field as the color of the menu item.
- `mctRGB4` specifies the color for the background of the menu.

A **menu item entry** is defined by a menu color entry record that contains a menu ID in the `mctID` field and an item number in the `mctItem` field. You can define only one menu item entry for each menu item. If you don't provide a menu item entry for an item in your application's menu color information table, the Menu Manager uses the colors defined by the menu title entry (or by the menu bar entry if the menu containing the item doesn't have a menu title entry). If neither a menu title entry nor a menu bar entry exists, the Menu Manager draws the mark, text, and keyboard equivalent in black. You can provide a menu item entry to specify a color for the mark, text, and keyboard equivalent of a specific menu item. The color information fields for a menu item entry are interpreted as follows:

- `mctRGB1` specifies the color for the mark of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry.
- `mctRGB2` specifies the color for the text of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry. The Menu Manager also draws a black-and-white icon of a menu item using the same color as defined by the `mctRGB2` field. (Use a `'cicn'` resource to provide a menu item with a color icon.)
- `mctRGB3` specifies the color for the keyboard equivalent of the menu item. If a menu item doesn't have a menu item entry, the Menu Manager uses the default value defined in the menu title entry or the menu bar entry.
- `mctRGB4` specifies the color for the background of the menu. If the menu color information table doesn't have a menu title entry for the menu this item is in, or doesn't have a menu bar entry, the Menu Manager uses the value in this field as the background color of the menu. If a menu title entry already exists, the Menu Manager replaces the value in the `mctRGB4` field of the menu item entry with the value defined in the `mctRGB4` field of the menu title entry (or with the `mctRGB2` field of the menu bar entry).

You can use the GetMCInfo function to get a copy of your application's menu color information table and the SetMCEntries procedure to set entries of your application's menu color information table, or you can provide 'mctb' resources that define the color entries for your menus.

The GetMenu, GetNewMBar, and ClearMenuBar routines can also modify the entries in the menu color information table. The GetMenu function looks for an 'mctb' resource with a resource ID equal to the value in the menuID parameter. If it finds one, it adds the entries to the application's menu color information table.

The GetNewMBar function builds a new menu color information table when it creates the new menu list. If you want to save the current menu color information table, call GetMCInfo before calling GetNewMBar.

The ClearMenuBar procedure reinitializes both the current menu list and the menu color information table.

Menu Manager Routines

The Menu Manager includes routines for creating menus, changing the characteristics of menu items, and handling user choice of menu commands. The Menu Manager also provides routines for adding items to and deleting items from menus, counting the number of items in a menu, getting a handle to a menu's menu record, disposing of menus, calculating the dimensions of a menu, highlighting the menu bar, and managing entries in your application's menu color information table.

Some Menu Manager routines can be accessed using more than one spelling of the routine's name, depending on the interface files supported by your development environment. For example, GetMenuHandle is also available as GetMHandle. Table 3-8 provides a mapping between the previous name of a routine and its new equivalent name.

Table 3-8 Mapping between new and previous names of Menu Manager routines

New name	Previous name
AppendResMenu	AddResMenu
DeleteMCEntries	DelMCEntries
DeleteMenuItem	DelMenuItem
DisposeMCInfo	DispMCInfo
GetMenuHandle	GetMHandle
GetMenuItemText	GetItem
InsertMenuItem	InsMenuItem
SetMenuItemText	SetItem

Initializing the Menu Manager

You can use the `InitMenus` procedure to initialize the Menu Manager.

You can use the `InitProcMenu` procedure to set the current menu list so that it uses a custom menu bar definition function if necessary.

InitMenus

The `InitMenus` procedure allocates space for your application's current menu list in your application's heap. Your application needs to call `InitMenus` only once to initialize the Menu Manager and the current menu list for your application.

```
PROCEDURE InitMenus;
```

DESCRIPTION

The `InitMenus` procedure creates the current menu list with no menus, submenus, or pop-up menus. `InitMenus` also creates your application's menu color information table. After allocating the menu color information table, `InitMenus` looks for an `'mctb'` resource with resource ID 0. You can provide an `'mctb'` resource with a resource ID of 0 as one of your application's resources if you want to use colors other than the default colors for your application's menu bar and menus. If `InitMenus` finds and successfully loads an `'mctb'` resource, it adds the information contained in that resource to the menu color information table (using `SetMCEntries`).

The `InitMenus` procedure also draws an empty menu bar.

SPECIAL CONSIDERATIONS

Your application must initalize QuickDraw, the Font Manager, and the Window Manager (using the `InitGraf`, `InitFonts`, and `InitWindows` procedures) before initializing the Menu Manager.

SEE ALSO

To set up the menus for your application's menu bar, use `GetNewMBar` and `SetMenuBar`, described on page 3-111 and page 3-112, respectively. You can also add menus to the current menu list using the `InsertMenu` procedure, described on page 3-108.

To remove all menus from the current menu list, use the `ClearMenuBar` procedure, described on page 3-110.

If your application uses its own menu bar definition function, use the `InitProcMenu` procedure to set the `mbResID` field of the current menu list to the resource ID of your custom `'MBDF'` resource.

See “The Menu Color Information Table Resource” on page 3-155 for a description of the `'mctb'` resource.

See the chapter “Window Manager” in this book for a description of the `InitWindows` procedure. See *Inside Macintosh: Imaging* and *Inside Macintosh: Text* for descriptions of the `InitGraf` and `InitFonts` procedures.

InitProcMenu

Apple recommends that you use the standard menu bar definition function. However, if your application provides its own menu bar definition function, use the `InitProcMenu` procedure to set the `mbResID` field of the current menu list to the resource ID of your custom `'MBDF'` resource.

```
PROCEDURE InitProcMenu (resID: Integer);
```

`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 $100. For resources of type `'MBDF'`, Apple reserves resource IDs $000 through $100 for its own use.

DESCRIPTION

The `InitProcMenu` procedure creates the current menu list if it hasn’t already been created by a previous call to `InitMenus`. The `InitProcMenu` procedure stores the resource ID that you specify in the `mbResID` field of the current menu list. The Menu Manager uses the menu bar definition function referred to in this field to draw the menu bar and to perform basic operations on menus.

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 that may call `InitMenus`.

SEE ALSO

See the description of the `InitMenus` procedure on page 3-103; you should use `InitMenus` if your application uses the standard menu bar definition function.

Creating Menus

You can use the `NewMenu` or `GetMenu` function to create a pull-down menu, although you usually create all the menus in your menu bar at once by providing an `'MBAR'` resource and using the `GetNewMBar` function. See "Getting and Setting the Menu Bar" on page 3-112 for information on creating a menu bar. You typically use the `NewMenu` or `GetMenu` function to create submenus or pop-up menus.

The `NewMenu` function creates a menu with the specified title, assigns it the specified menu ID, and creates a menu record for the menu. Use `AppendMenu`, `InsertMenuItem`, `AppendResMenu`, or `InsertResMenu` to add items to menus you create with `NewMenu`.

The `GetMenu` function creates a menu with the title, items, and characteristics defined in a specified `'MENU'` resource.

Both `NewMenu` and `GetMenu` allocate space in your application's heap for the menu record and return a handle to the menu's newly created menu record.

To add menus created by `NewMenu` or `GetMenu` to the current menu list, use the `InsertMenu` procedure. To update the menu bar with any new menu titles, use `DrawMenuBar`.

NewMenu

You can use the `NewMenu` function to create an empty menu with a specified title and menu ID. In most cases you should store information about your menus (such as their titles, items, and characteristics) in resources; use the `GetMenu` or `GetNewMBar` function to create menus from resource definitions.

```
FUNCTION NewMenu (menuID: Integer; menuTitle: Str255): MenuHandle;
```

`menuID`	The menu ID of the menu. (Note that this is not the resource ID of a `'MENU'` resource.) The menu ID is a number that identifies the menu. Use positive menu IDs for menus belonging to your application. Use negative menu IDs for desk accessories (except for submenus of a desk accessory). Submenus must have menu IDs from 1 through 255. For submenus of an application, use menu IDs from 1 through 235; for submenus of a desk accessory, use menu IDs from 236 through 255. Apple reserves the menu ID of 0.
`menuTitle`	The title of the new menu. Note that in most cases you should store the titles of menus in resources, so that your menu titles can be more easily localized.

DESCRIPTION

The `NewMenu` function creates a menu with the specified title, assigns it the specified menu ID, creates a menu record for the menu, and returns a handle to the menu record. It sets up the menu record to use the standard menu definition procedure (and it reads the standard menu definition procedure into memory if it isn't already there). The `NewMenu` function does not insert the newly created menu into the current menu list.

After creating a menu with `NewMenu`, use `AppendMenu`, `InsertMenuItem`, `AppendResMenu`, or `InsertResMenu` to add menu items to the menu. To add a menu created by `NewMenu` to the current menu list, use the `InsertMenu` procedure. To update the menu bar with any new menu titles, use the `DrawMenuBar` procedure.

SPECIAL CONSIDERATIONS

To release the memory associated with a menu that you created using `NewMenu`, 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's menu record. After disposing of a menu, use `DrawMenuBar` to update the menu bar.

If the `NewMenu` function is unable to create the menu record, it returns `NIL` as its function result.

SEE ALSO

For information on how to add items to a menu, see the description of `AppendMenu` on page 3-124, `InsertMenuItem` on page 3-126, `AppendResMenu` on page 3-128, and `InsertResMenu` on page 3-129. For information on `InsertMenu`, see page 3-108. To dispose of a menu, see the description of `DeleteMenu` on page 3-109 and `DisposeMenu` on page 3-140.

GetMenu

Use the `GetMenu` function to create a menu with the title, items, and other characteristics defined in a `'MENU'` resource with the specified resource ID. You typically use this function only when you create submenus; you can create all your pull-down menus at once using the `GetNewMBar` function, and you can create pop-up menus using the standard pop-up control definition function.

```
FUNCTION GetMenu (resourceID: Integer): MenuHandle;
```

`resourceID` The resource ID of the `'MENU'` resource that defines the characteristics of the menu. (You usually use the same number for a menu's resource ID as the number that you specify for the menu ID in the menu resource.)

DESCRIPTION

The GetMenu function creates a menu according to the specified menu resource, and it also creates a menu record for the menu. It reads the menu definition procedure (specified in the menu resource) into memory if it isn't already in memory, and it stores a handle to the menu definition procedure in the menu record. The GetMenu function does not insert the newly created menu into the current menu list.

After reading the 'MENU' resource, the GetMenu function searches for an 'mctb' resource with the same resource ID as the 'MENU' resource. If GetMenu finds this 'mctb' resource, it uses the information in the 'mctb' resource to add entries for this menu to the application's menu color information table. The GetMenu function uses SetMCEntries to add the entries defined by the 'mctb' resource to the application's menu color information table. If GetMenu doesn't find this 'mctb' resource, it uses the default colors specified in the menu bar entry of the application's menu color information, or, if the menu bar entry doesn't exist, it uses the standard colors for the menu.

The GetMenu function returns a handle to the menu record of the menu. You can use the returned menu handle to refer to this menu in most Menu Manager routines. If GetMenu is unable to read the menu or menu definition procedure from the resource file, GetMenu returns NIL.

After creating a menu with GetMenu, you can use AppendMenu, InsertMenuItem, AppendResMenu, or InsertResMenu to add more menu items to the menu if necessary.

To add a menu created by GetMenu to a menu list, use the InsertMenu procedure. To update the menu bar with any new menu titles, use the DrawMenuBar procedure.

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. ▲

SPECIAL CONSIDERATIONS

To release the memory associated with a menu that you read from a resource file using GetMenu, first call DeleteMenu to remove the menu from the menu list and to remove any menu title entry or menu item entries for this menu in the application's menu color information table, then call the Resource Manager procedure ReleaseResource to dispose of the menu's menu record. Use DrawMenuBar to update the menu bar.

▲ **WARNING**
Call GetMenu only once for a particular menu. If you need the handle of a menu currently in the menu list, use GetMenuHandle or the Resource Manager function GetResource. ▲

SEE ALSO

For a description of the `'MENU'` resource, see "The Menu Resource" on page 3-151; for a sample `'MENU'` resource in Rez format, see Listing 3-2 on page 3-48. For information on the `'mctb'` resource, see "The Menu Color Information Table Resource" on page 3-155.

For details on how to add items to a menu, see the description of `AppendMenu` on page 3-124, `InsertMenuItem` on page 3-126, `AppendResMenu` on page 3-128, and `InsertResMenu` on page 3-129. To remove a menu, see the description of `DeleteMenu` on page 3-109. To update the menu bar, use the `DrawMenuBar` procedure, described on page 3-113.

Adding Menus to and Removing Menus From the Current Menu List

After creating a menu with `NewMenu` or `GetMenu`, use the `InsertMenu` procedure to insert the menu into the current menu list. Use the `DeleteMenu` procedure to delete a menu from the current menu list; use the `ClearMenuBar` procedure to remove all menus from the current menu list.

InsertMenu

Use the `InsertMenu` procedure to insert an existing menu into the current menu list.

```
PROCEDURE InsertMenu (theMenu: MenuHandle; beforeID: Integer);
```

`theMenu`	A handle to the menu record of the menu. The `NewMenu` and `GetMenu` functions return a handle to a menu record that you can use in this parameter.
`beforeID`	A number that indicates where in the current menu list the menu should be inserted. `InsertMenu` inserts the menu into the current menu list before the menu whose menu ID equals the number specified in the `beforeID` parameter. If the number in the `beforeID` parameter is 0 (or it isn't the ID of any menu in the menu list), `InsertMenu` adds the new menu after all others (except before the Help, Keyboard, and Application menus). If the menu is already in the current menu list or the menu list is already full, `InsertMenu` does nothing. You can specify –1 for the `beforeID` parameter to insert a submenu into the current menu list. The submenus in the submenu portion of the menu list do not have to be currently associated with a hierarchical menu item; you can store submenus in the menu list and later specify that a menu item has a submenu if needed. However, note that the `MenuKey` function scans all menus in the menu list for keyboard equivalents, including submenus that are not associated with any menu item. You should not define keyboard equivalents for submenus that are in the current menu list but not associated with a menu item.

You can also specify –1 for the `beforeID` parameter to insert a pop-up menu into the current menu list. However, if you use the standard pop-up control definition function, the pop-up control automatically inserts the menu into the current menu list according to the needs of the pop-up control.

DESCRIPTION

The `InsertMenu` procedure inserts into the current menu list the menu identified by the specified handle to a menu record. To update the menu bar to reflect the new menu, use `DrawMenuBar`.

SEE ALSO

For details on how to update your application's menu bar, see the description of `DrawMenuBar` on page 3-113.

DeleteMenu

Use the `DeleteMenu` procedure to delete an existing menu from the current menu list.

```
PROCEDURE DeleteMenu (menuID: Integer);
```

`menuID`	The menu ID of the menu to delete from the current menu list. If the menu list does not contain a menu with the specified menu ID, `DeleteMenu` does nothing.

DESCRIPTION

The `DeleteMenu` procedure deletes the menu identified by the specified menu ID from the current menu list, and it removes all color entries for that menu from the application's menu color information table. `DeleteMenu` does not release the memory occupied by the menu's menu record. To release the memory occupied by the menu's associated data structures, use `DisposeMenu` if you created the menu using `NewMenu`; use the Resource Manager procedure `ReleaseResource` if you created the menu using `GetMenu` or you read the resource in using `GetNewMBar`.

The `DeleteMenu` procedure first checks the submenu portion of the current menu list for a menu ID with the specified ID. If it finds such a menu, it deletes that menu and returns. If `DeleteMenu` doesn't find the menu in the submenu portion, it checks the regular portion of the current menu list. This allows a desk accessory to delete a submenu without deleting an application's menu whose menu ID might conflict with the menu ID defined by a desk accessory.

After deleting a menu, use `DrawMenuBar` to update the menu bar to reflect the changes to the current menu list.

SEE ALSO

For details on how to dispose of a menu's associated data structures using `DisposeMenu`, see "Disposing of Menus" on page 3-140. For information on the `ReleaseResource` procedure, see the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.

ClearMenuBar

Use the `ClearMenuBar` procedure to delete all menus from the current menu list.

```
PROCEDURE ClearMenuBar;
```

DESCRIPTION

The `ClearMenuBar` procedure deletes all menus from the current menu list and deletes all color entries from the application's menu color information table. `ClearMenuBar` does not release the memory occupied by any of the menus' menu records or the menu color information table. To release the memory occupied by the data structures associated with the menus, use `DisposeMenu` for each menu you created using `NewMenu`; use `ReleaseResource` for each menu you created using `GetMenu` or if you read the resource in using `GetNewMBar`.

After deleting all menus from the current menu list, use `DrawMenuBar` to update the appearance of the menu bar.

SEE ALSO

To update your application's menu bar, see the description of `DrawMenuBar` on page 3-113. For information on the `ReleaseResource` procedure, see the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox*.

Getting a Menu Bar Description From an 'MBAR' Resource

You usually create your application's menu bar by doing the following:

- defining the order and resource ID of your menus in an `'MBAR'` resource
- defining the menus in `'MENU'` resources
- reading in these descriptions using the `GetNewMBar` function
- setting the current menu list to the menu list returned by `GetNewMBar`
- updating the menu bar using `DrawMenuBar`

GetNewMBar

Use the `GetNewMBar` function to read in the definition of a menu bar from an `'MBAR'` resource.

```
FUNCTION GetNewMBar (menuBarID: Integer): Handle;
```

`menuBarID`	The resource ID of an `'MBAR'` resource that specifies the menus for a menu bar.

DESCRIPTION

The `GetNewMBar` function reads in the definition of a menu bar and its associated menus from an `'MBAR'` resource. The `'MBAR'` resource identifies the order of menus contained in its menu bar. For each menu, it also specifies the menu's resource ID. The `GetNewMBar` function reads in each menu from the `'MENU'` resource with the resource ID specified in the `'MBAR'` resource.

The `GetNewMBar` function creates a menu list for the menu bar defined by the `'MBAR'` resource and returns a handle to the menu list. (If the resource isn't already in memory, `GetNewMBar` reads it into memory.) If `GetNewMBar` can't read the resource, `GetNewMBar` returns `NIL`. `GetNewMBar` uses `GetMenu` to read in each individual menu.

After reading in menus from an `'MBAR'` resource, use `SetMenuBar` to make the menu list created by `GetNewMBar` the current menu list. Then use `DrawMenuBar` to update the menu bar.

To release the memory occupied by the data structures associated with the menus in a menu list, use `DisposeMenu` for each menu you created using `NewMenu`; use the Resource Manager procedure `ReleaseResource` for each menu you created using `GetMenu` or if you read the resource in using `GetNewMBar`. To release the memory occupied by a menu list, use the Memory Manager procedure `DisposeHandle`.

SPECIAL CONSIDERATIONS

The `GetNewMBar` function first saves the current menu list and then clears the current menu list and your application's menu color information table. It then creates a new menu list. Before returning a handle to the new menu list, the `GetNewMBar` function restores the current menu list to the previously saved menu list, but `GetNewMBar` does not restore the previous menu color information table. To save and then restore your application's current menu color information table, call the `GetMCInfo` function before `GetNewMBar` and call the `SetMCInfo` procedure afterward.

While you supply only the resource ID of an `'MBAR'` resource to the `GetNewMBar` function, your application often needs to use the menu IDs defined in each of your menus' `'MENU'` resources. Most Menu Manager routines require either a menu ID or a handle to a menu record to perform operations on a specific menu. For menus in the current menu list, you can use the `GetMenuHandle` function to get the handle to a menu record of a menu with a given menu ID.

SEE ALSO

For a description of the `'MENU'` resource, see "The Menu Resource" on page 3-151; for a sample `'MENU'` resource in Rez format, see Listing 3-2 on page 3-48. For a description of the `'MBAR'` resource, see "The Menu Bar Resource" on page 3-155; for a sample `'MBAR'` resource in Rez format, see Listing 3-4 on page 3-49. For information on the `'mctb'` resource, see "The Menu Color Information Table Resource" on page 3-155. For information about the Resource Manager, see *Inside Macintosh: More Macintosh Toolbox.*

Getting and Setting the Menu Bar

You can use the `GetMenuBar` function to get a handle to a copy of the current menu list. Use the `SetMenuBar` procedure to set the current menu bar to a menu list previously returned by `GetMenuBar` or `GetNewMBar`. You can get the height of the menu bar using the `GetMBarHeight` function.

GetMenuBar

Use the `GetMenuBar` function to get a handle to a copy of the current menu list.

```
FUNCTION GetMenuBar: Handle;
```

DESCRIPTION

The `GetMenuBar` function creates a copy of the current menu list and returns a handle to the copy. You can save the returned menu list and then add menus to or remove menus from the current menu list (using `InsertMenu`, `DeleteMenu`, or `ClearMenuBar`). You can later restore the saved menu list using `SetMenuBar`.

To release the memory occupied by a saved menu list, use the Memory Manager's `DisposeHandle` procedure.

▲ **WARNING**
`GetMenuBar` doesn't copy the menu records, just the menu list (which contains handles to the menu records). Do not dispose of any menus in a saved menu list if you wish to restore the menu list later. ▲

SetMenuBar

Use the `SetMenuBar` procedure to set the current menu list to a specified menu list.

```
PROCEDURE SetMenuBar (menuList: Handle);
```

`menuList`	A handle to a menu list that specifies the menus for a menu bar. You should specify a handle returned by `GetMenuBar` or `GetNewMBar`.

DESCRIPTION

The `SetMenuBar` procedure copies the given menu list to the current menu list. As with `GetMenuBar`, `SetMenuBar` doesn't copy the menu records, just the menu list (which contains handles to the menu records).

You can use `SetMenuBar` to restore a menu list that you previously saved using `GetMenuBar` or to set the current menu list to a menu list created by `GetNewMBar`.

The `SetMenuBar` procedure sets only the current menu list; to update the menu bar according to the new menu list, use the `DrawMenuBar` procedure.

GetMBarHeight

Use the `GetMBarHeight` function if you need to determine the current height of the menu bar. When the Roman script system is the current system script, the menu bar is 20 pixels high. If a non-Roman script is the current system script, the menu bar may be greater than 20 pixels high to accommodate the current system font.

```
FUNCTION GetMBarHeight: Integer;
```

DESCRIPTION

The `GetMBarHeight` function returns the current height, in pixels, of the menu bar.

Drawing the Menu Bar

Whenever your application adds menus to or removes menus from the current menu list, you should update the titles of the menus in the menu bar using the `DrawMenuBar` procedure. If you change the enabled state of a menu, you should call `DrawMenuBar` to update the menu title accordingly. Alternatively, you can use the `InvalMenuBar` procedure instead of `DrawMenuBar` to invalidate the menu bar; this causes the Event Manager to redraw the menu bar as part of its normal processing of update events.

DrawMenuBar

Use the `DrawMenuBar` procedure to draw the menu bar based on the current menu list.

```
PROCEDURE DrawMenuBar;
```

DESCRIPTION

The `DrawMenuBar` procedure draws (or redraws) the menu bar according to the current menu list. You must call `DrawMenuBar` to update the menu bar after adding menus to or deleting menus from the current menu list using `InsertMenu` or `DeleteMenu`, after setting the current menu list using `SetMenuBar`, after changing the enabled state of a menu, or after any other routine that changes the current menu list.

InvalMenuBar

Use the `InvalMenuBar` procedure to invalidate the menu bar.

```
PROCEDURE InvalMenuBar;
```

DESCRIPTION

The `InvalMenuBar` procedure marks the menu bar as changed and in need of updating. When the Event Manager scans update regions for regions that require updating, the Event Manager also checks to determine whether the menu bar requires updating (because of a call to `InvalMenuBar`). If the menu bar needs updating, the Event Manager calls the `DrawMenuBar` procedure to draw the menu bar.

You can use `InvalMenuBar` instead of `DrawMenuBar` to minimize blinking in the menu bar. For example, if you have several application-defined routines that can change the enabled state of a menu and each calls `DrawMenuBar`, you can replace the calls to `DrawMenuBar` with calls to `InvalMenuBar`. In this way the menu bar is redrawn only once instead of multiple times in quick succession. If you need to make immediate changes to the menu bar, use `DrawMenuBar`. If you want to redraw the menu bar at most once each time through your event loop, use `InvalMenuBar`. The `InvalMenuBar` procedure is available only in System 7.

Responding to the User’s Choice of a Menu Command

When the user presses the mouse button while the cursor is in the menu bar, your application should call the `MenuSelect` function to allow the user to choose a command from the menu bar. If the user presses the mouse button while the cursor is over a pop-up menu that does not use the standard pop-up control definition function, your application should call the `PopUpMenuSelect` function to allow the user to make a choice from the pop-up menu.

You should also allow the user to choose a menu command by typing a keyboard equivalent. When the user presses a key on the keyboard, your application should determine if the Command key was pressed at the same time, and, if so, your application should call the `MenuKey` function to map this keyboard combination to any corresponding Command-key equivalent.

If the user chooses an item, both the `MenuSelect` and `MenuKey` functions highlight the title of the menu containing the chosen item and report the user’s choice to your application. Your application should perform the corresponding command and, when finished, should unhighlight the menu title using the `HiliteMenu` procedure to indicate to the user that the command is completed.

If the user releases the mouse button while the cursor is over a disabled item or types the keyboard equivalent of a disabled item, `MenuSelect` and `MenuKey` do not report the menu ID or menu item of the item. To determine if the user chose a disabled item (for example, so that your application can provide assistance to the user or explain to the user why the command is disabled), you can use the `MenuChoice` function to return the menu ID and menu item of the disabled menu command.

Your application should adjust its menus before calling `MenuSelect` or `MenuKey`. For example, you should enable or disable menu items as appropriate and add any applicable checkmarks or dashes to items that show attributes.

MenuSelect

Use the `MenuSelect` function to allow the user to choose a menu item from the menus in your application's menu bar.

```
FUNCTION MenuSelect (startPt: Point): LongInt;
```

`startPt`	The point (in global coordinates) representing the location of the cursor at the time the mouse button was pressed.

DESCRIPTION

When the user presses the mouse button while the cursor is in the menu bar, your application receives a mouse-down event. To handle mouse-down events in the menu bar, pass the location of the cursor at the time of the mouse-down event as the `startPt` parameter to `MenuSelect`. The `MenuSelect` function displays and removes menus as the user moves the cursor over menu titles in the menu bar, and it handles all user interaction until the user releases the mouse button.

As the user drags the cursor through the menu bar, the `MenuSelect` function highlights the title of the menu the cursor is currently over and displays all items in that menu. If the user moves the cursor so that it is over a different menu, the `MenuSelect` function removes the previous menu and unhighlights its menu title.

The `MenuSelect` function highlights and unhighlights menu items as the user drags the cursor over the items in a menu. The `MenuSelect` function highlights a menu item if the item is enabled and the cursor is currently over it; it removes such highlighting when the user moves the cursor to another menu item. The `MenuSelect` function does not highlight disabled menu items.

If the user chooses an enabled menu item (including any item from a submenu), the `MenuSelect` function returns a value as its function result that indicates which menu and menu item the user chose. The high-order word of the function result contains the menu ID of the menu, and the low-order word contains the item number of the menu item chosen by the user. The `MenuSelect` function leaves the menu title highlighted; after performing the chosen task your application should unhighlight the menu title using the `HiliteMenu` procedure.

If the user chooses an item from a submenu, `MenuSelect` returns the menu ID of the submenu in the high-order word and the item chosen by the user in the low-order word of its function result. The `MenuSelect` function also highlights the title of the menu in the menu bar that the user originally displayed in order to begin traversing to the submenu. After performing the chosen task, your application should unhighlight the menu title.

If the user releases the mouse button while the cursor is over a disabled item, in the menu bar, or outside of any menu, the `MenuSelect` function returns 0 in the high-order word of its function result and the low-order word is undefined. If it is necessary for your application to find the item number of the disabled item, your application can call `MenuChoice` to return the menu ID and menu item.

If the user chooses an enabled item in a menu that a desk accessory has inserted into your application's menu list, `MenuSelect` uses the `SystemMenu` procedure to process this occurrence and returns 0 to your application in the high-order word.

SPECIAL CONSIDERATIONS

When the `MenuSelect` function pulls down a menu, it stores the bits behind the menu as a relocatable object in the application heap of your application.

ASSEMBLY-LANGUAGE INFORMATION

The `InitMenus` and `InitProcMenu` procedures initialize the `MenuHook` and `MBarHook` global variables to 0. If you choose, you can store the addresses of routines that `MenuSelect` calls in these global variables. The `MenuHook` global variable contains the address (if any) of a routine that `MenuSelect` calls repeatedly while the mouse button is down. `MenuSelect` does not pass any parameters to this routine.

The `MBarHook` global variable contains the address (if any) of a routine that `MenuSelect` calls after a menu title is highlighted and the menu rectangle is calculated but before the menu is drawn. The menu rectangle is the rectangle (in global coordinates) in which the menu will be drawn. `MenuSelect` passes a pointer to the menu rectangle on the stack. If you provide the address of a routine in the `MBarHook` global variable, it should normally return 0 in the D0 register, indicating that `MenuSelect` should continue; returning 1 causes `MenuSelect` to cancel its operation and return immediately to the application.

The `MenuSelect` function uses the global variable `MBarEnable` to determine if all menus in the current menu bar belong to a desk accessory or an application. If the `MBarEnable` global variable is nonzero, then all menus in the current menu bar belong to a desk accessory. If the `MBarEnable` global variable is 0, then all menus in the current menu bar belong to an application. If you're writing a desk accessory, you may need to set the `MBarEnable` global variable to a nonzero value; if you're writing an application, you should not change the value of the `MBarEnable` global variable.

The global variable `TheMenu` contains the ID of the currently highlighted menu in the menu bar. If the user chooses an item from a submenu, `TheMenu` contains the menu ID of the submenu, not the menu to which the submenu is attached.

SEE ALSO

For information on adjusting your application's menus before calling `MenuSelect`, see "Adjusting the Menus of an Application" beginning on page 3-73.

See the description of the `HiliteMenu` procedure on page 3-119 for details on how to unhighlight a menu. For information on how to determine if the user chose a disabled item, see the description of the `MenuChoice` function on page 3-118.

MenuKey

If the user presses another key while holding down the Command key, call the `MenuKey` function to determine if the keyboard combination maps to the keyboard equivalent of a menu item in a menu in the current menu list.

```
FUNCTION MenuKey (ch: Char): LongInt;
```

`ch`	The 1-byte character representing the key pressed by the user in combination with the Command key.

DESCRIPTION

The `MenuKey` function maps the given character to the menu and menu item with that keyboard equivalent. The `MenuKey` function returns as its function result a value that indicates the menu ID and menu item that has the keyboard equivalent corresponding to the given character.

The `MenuKey` function does not distinguish between uppercase and lowercase letters. It takes the 1-byte character passed to it and calls the `UpperText` procedure (which provides localizable uppercase conversion of the character). Thus, `MenuKey` translates any lowercase character to uppercase when comparing a keyboard event to keyboard equivalents. 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, `MenuKey` 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` procedure.

If the given character does not map to an enabled menu item in the current menu list, `MenuKey` returns 0 in its high-order word and the low-order word is undefined.

If the given character maps to a menu item in a menu that a desk accessory has inserted into your application’s menu list, `MenuSelect` uses the `SystemMenu` procedure to process this occurrence and returns 0 to your application in the high-order word.

You should not define menu items with identical keyboard equivalents. The `MenuKey` 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, `MenuKey` returns the first one it finds.

The `MenuKey` 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, `MenuKey` 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.

Apple reserves the Command-key codes $1B (Control-[ ) through $1F (Control-_) to indicate meanings other than keyboard equivalents. `MenuKey` ignores these character codes and returns a function result of 0 if you specify any of these values in the `ch` parameter. Your application should not use these character codes for its own use.

The global variable `TheMenu` contains the ID of the currently highlighted menu in the menu bar. If the user chooses an item from a submenu, `TheMenu` contains the menu ID of the submenu, not the menu to which the submenu is attached.

▲ **WARNING**
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. If `MenuKey` detects a circular hierarchical menu, it creates a system error with error number 86. ▲

SEE ALSO

To unhighlight a menu, use the `HiliteMenu` procedure, described on page 3-119. To provide support for keyboard equivalents other than Command-key equivalents, see the discussion of `'KCHR'` resources in *Inside Macintosh: Text*.

MenuChoice

If your application needs to find the item number of a disabled menu item that the user attempted to choose, you can use the `MenuChoice` function to return the chosen menu item.

```
FUNCTION MenuChoice: LongInt;
```

DESCRIPTION

If the user chooses a disabled menu item, the `MenuChoice` function returns a value that indicates which menu and menu item the user chose. The high-order word of the function result contains the menu ID of the menu, and the low-order word contains the item number of the menu item chosen by the user.

The `MenuChoice` function returns 0 as the low-order word of its function result if the mouse button was released while the cursor was in the menu bar or outside the menu.

SPECIAL CONSIDERATIONS

The Menu Manager updates the global variable `MenuDisable` whenever a menu is displayed. As the user moves the cursor over each item, the Menu Manager calls the menu definition procedure of the menu to update the `MenuDisable` global variable to reflect the current menu ID and menu item. The standard menu definition procedure updates the global variable `MenuDisable` appropriately. If your application uses its own menu definition procedure, your menu definition procedure should support this feature; if you use a menu definition procedure that does not update the global variable `MenuDisable` appropriately, the result returned by `MenuChoice` is undefined.

HiliteMenu

You can use the `HiliteMenu` procedure to highlight or unhighlight menu titles. For example, after performing a menu command chosen by the user, use the `HiliteMenu` procedure to unhighlight the menu title.

```
PROCEDURE HiliteMenu (menuID: Integer);
```

`menuID`	The menu ID of the menu whose title should be highlighted. If the menu title of the specified menu is already highlighted, `HiliteMenu` does nothing. If the menu ID is 0 or the specified menu ID isn't in the current menu list, `HiliteMenu` unhighlights whichever menu title is currently highlighted (if any).

DESCRIPTION

The `MenuSelect` and `MenuKey` functions highlight the title of the menu containing the item chosen by the user. After performing the chosen task, your application should unhighlight the menu title by calling `HiliteMenu` and passing 0 in the `menuID` parameter.

The `HiliteMenu` procedure highlights a menu title by first saving the bits behind the title rectangle and then drawing the highlighted title. `HiliteMenu` unhighlights a menu title by restoring the bits behind the menu title.

The global variable `TheMenu` contains the ID of the currently highlighted menu in the menu bar. If the user chooses an item from a submenu, `TheMenu` contains the menu ID of the submenu, not the menu to which the submenu is attached.

SEE ALSO

To highlight the entire menu bar, use the `FlashMenuBar` procedure, described on page 3-141.

PopUpMenuSelect

To display a pop-up menu without using the standard pop-up control definition function, use the `PopUpMenuSelect` function to display the pop-up menu anywhere on the screen. If your application uses the standard pop-up control definition function, your application does not need to use `PopUpMenuSelect`.

```
FUNCTION PopUpMenuSelect (menu: MenuHandle;
                          Top: Integer; Left: Integer;
                          PopUpItem: Integer)
                          : LongInt;
```

`menu`	A handle to the menu record of the menu. The `NewMenu`, `GetMenu`, and `GetMenuHandle` functions return a handle to a specified menu's menu record.
`Top`	The top coordinate of the pop-up box when it is closed. This value should be in global coordinates.
`Left`	The left coordinate of the pop-up box when it is closed. This value should be in global coordinates.
`PopUpItem`	The item number of the current item minus 1. This value should correspond to the user's previous choice from this menu. If the user has not previously made a choice, this value should be set to the default value.

DESCRIPTION

The `PopUpMenuSelect` function uses the location specified by the `Top` and `Left` parameters to determine where to display the specified item of the pop-up menu. The `PopUpMenuSelect` function displays the pop-up menu so that the menu item specified in the `PopUpItem` parameter appears highlighted at the specified location. Figure 3-24 on page 3-34 shows the pop-up title and pop-up box of a pop-up menu.

The `PopUpMenuSelect` function highlights and unhighlights menu items and handles all user interaction until the user releases the mouse button. The `PopUpMenuSelect` function returns the menu ID of the chosen menu in the high-order word of its function result and the chosen menu item in the low-order word.

Your application is responsible for highlighting the pop-up title, setting the mark of the current menu item appropriately, and drawing the text and downward-pointing indicator in the pop-up box before calling `PopUpMenuSelect`. Your application should also make sure the pop-up menu is in the submenu portion of the current menu list before calling `PopUpMenuSelect`. (You can use the `InsertMenu` procedure and specify –1 in the `beforeID` parameter to insert the pop-up menu into the current menu list.)

After calling `PopUpMenuSelect`, your application can delete the pop-up menu from the current menu list or leave it in the current menu list.

Your application is also responsible for storing the current value of the menu item, drawing the text and downward-pointing indicator in the pop-up box, and unhighlighting the pop-up title after calling `PopUpMenuSelect`. If you use the standard pop-up control definition function, these actions are performed for you by the pop-up control and your application does not need to call `PopUpMenuSelect`.

When implementing pop-up menus, you should follow the guidelines for pop-up menus described in *Macintosh Human Interface Guidelines*. For example, you should define the pop-up box of your pop-up menu as a rectangle that is the same height as a menu item, with a one-pixel drop shadow, and should make the pop-up box wide enough to show the currently selected item and a downward-pointing indicator.

SystemMenu

The `MenuSelect` and `MenuKey` functions call the `SystemMenu` procedure when the user chooses an item in a menu that belongs to a desk accessory launched in your application's partition. Your application should not need to call the `SystemMenu` procedure.

```
PROCEDURE SystemMenu (menuResult: LongInt);
```

`menuResult`	The value that indicates the menu and menu item chosen by the user. The menu ID is in the high-order word, and the menu item is in the low-order word. The menu ID for a menu belonging to a desk accessory is a negative number.

DESCRIPTION

The `SystemMenu` procedure directs the desk accessory to perform the appropriate action for the given menu item by calling the desk accessory's control routine and passing the `accMenu` constant in the `csCode` parameter. The desk accessory should perform the desired action and return. See *Inside Macintosh: Devices* for more information on desk accessories.

ASSEMBLY-LANGUAGE INFORMATION

If you're writing a desk accessory, you may need to set the `MBarEnable` global variable to appropriate values. If the `MBarEnable` global variable is nonzero, then all menus in the current menu bar belong to a desk accessory. If the `MBarEnable` global variable is 0, then all menus in the current menu bar belong to an application. If you're writing an application, you should not change the value of the `MBarEnable` global variable.

SystemEdit

When the user chooses one of the standard editing commands in the Edit menu (Undo, Cut, Copy, Paste, and Clear), call the `SystemEdit` function to determine whether the active window belongs to a desk accessory that is launched in your application's partition. If so, the `SystemEdit` function directs the desk accessory to perform the editing command and returns `TRUE`. If the active window does not belong to a desk accessory launched in your application's partition, `SystemEdit` returns `FALSE` and your application should process the command.

```
FUNCTION SystemEdit (editCmd: Integer): Boolean;
```

`editCmd`	The item number of the standard editing command chosen by the user.

Getting a Handle to a Menu Record

Most Menu Manager routines that manage menus require that you specify a handle to the menu record of the menu on which you want to perform an operation. You can use the `HMGetHelpMenuHandle` function to get a handle to your application's Help menu. Use the `GetMenuHandle` function to get a handle to the menu record of any of your application's other pull-down menus or submenus in the current menu list. For pop-up menus that use the standard control definition function, you can access the control record to get the menu's handle.

GetMenuHandle

You can use the `GetMenuHandle` function to get a handle to the menu record of any of your application's menus other than its Help menu. (Use the `HMGetHelpMenuHandle` function to get a handle to the menu record of your application's Help menu.) The `GetMenuHandle` function is also available as the `GetMHandle` function.

```
FUNCTION GetMenuHandle (menuID: Integer): MenuHandle;
```

`menuID`	The menu ID of the menu. (Note that this is not the resource ID, although you often assign the menu ID so that it matches the resource ID.) You assign a menu ID in the `'MENU'` resource of a menu. If you do not define your menus in `'MENU'` resources, you can assign a menu ID using `NewMenu`.

DESCRIPTION

The `GetMenuHandle` function returns a handle to the menu record of the menu having the specified menu ID. If the menu is in the current menu list, `GetMenuHandle` returns a handle to the menu record of the menu as its function result. Otherwise, `GetMenuHandle` returns `NIL` as its function result.

SPECIAL CONSIDERATIONS

To get a handle to a menu record of a pop-up menu that you create using the pop-up control definition function, dereference the `cntrlData` field of the pop-up menu's control record instead of using `GetMenuHandle`.

HMGetHelpMenuHandle

Use the `HMGetHelpMenuHandle` function to get a handle to the menu record of your application's Help menu.

```
FUNCTION HMGetHelpMenuHandle (VAR mh: MenuHandle): OSErr;
```

`mh`	The `HMGetHelpMenuHandle` function returns a copy of a handle to your application's Help menu in this parameter.

DESCRIPTION

The `HMGetHelpMenuHandle` function returns in the `mh` parameter a copy of a handle to the menu record of your application's Help menu. With this handle, you can append items to your application's Help menu by using the `AppendMenu` procedure or other related Menu Manager routines. The Help Manager automatically adds the divider that separates your items from the rest of the Help menu items.

Be sure to define help balloons for your items in the Help menu by creating an `'hmnu'` resource and specifying the `kHMHelpMenuID` constant as its resource ID.

The Menu Manager functions `MenuSelect` and `MenuKey` return a result with the menu ID in the high-order word and the menu item in the low-order word. The `MenuSelect` function (and the `MenuKey` function, if the user chooses an item with a keyboard equivalent) returns the `kHMHelpMenuID` constant in the high-order word when the user chooses an appended item from the Help menu. The menu item number of the appended menu item is returned in the low-order word of the function result. Apple reserves the right to change the number of standard items in the Help menu. To determine the number of items in the Help menu, call the `CountMItems` function.

SPECIAL CONSIDERATIONS

Do not use the `GetMenuHandle` function to get a handle to the menu record of the Help menu. `GetMenuHandle` returns a handle to the menu record of the global Help menu, not the menu record of the Help menu that is specific to your application.

RESULT CODES

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

SEE ALSO

For examples of how to add items to your application's Help menu and how to handle the user's choice of an item in the Help menu, see Listing 3-14 on page 3-68 and Listing 3-26 on page 3-81. See the chapter "Help Manager" in *Inside Macintosh: More Macintosh Toolbox* for information on creating help balloons for the menus of your application.

Adding and Deleting Menu Items

You can add the names of all resources of a specified type to a menu using the `InsertResMenu` or `AppendResMenu` procedure. You can add menu items that you define to a menu using the `AppendMenu` or `InsertMenuItem` procedure. You can also delete menu items using the `DeleteMenuItem` procedure. In most cases you should not insert or delete individual menu items from an already existing menu unless the user expects a menu (such as a list of currently open documents) to change.

If you add menu items using the `AppendMenu` or `InsertMenuItem` procedure, you should define in resources the text and other characteristics of the menu items that you add. This makes your application easier to localize for other regions.

AppendMenu

Use the `AppendMenu` procedure to append one or more items to a menu previously created using `NewMenu`, `GetMenu`, or `GetNewMBar`.

```
PROCEDURE AppendMenu (menu: MenuHandle; data: Str255);
```

`menu`	A handle to the menu record of the menu to which you wish to append the menu item or items.
`data`	A string that defines the characteristics of the new menu item or items. Note that in most cases you should store the text of a menu item in a resource, so that your menu items can be more easily localized. The `AppendMenu` procedure appends the menu items in the order in which they are listed in the `data` parameter.

DESCRIPTION

The `AppendMenu` procedure appends any defined menu items to the specified menu. The menu items are added to the end of the menu. You specify the text of any menu items and their characteristics in the `data` parameter. You can embed metacharacters in the string to define various characteristics of a menu item.

Here are the metacharacters that you can specify in the `data` parameter:

Metacharacter	Description
`;` or Return	Separates menu items.
`^`	When followed by an icon number, defines the icon for the item. If the keyboard equivalent field contains $1C, this number is interpreted as a script code.
`!`	When followed by a character, defines the mark for the item. If the keyboard equivalent field contains $1B, this value is interpreted as the menu ID of a submenu of this menu item.
`<`	When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to Bold, Italic, Underline, Outline, or Shadow, respectively.
`/`	When followed by a character, defines the keyboard equivalent for the item. When followed by $1B, specifies that this menu item has a submenu. To specify that the menu item has a script code, small icon, or reduced icon, use the `SetItemCmd` procedure to set the keyboard equivalent field to $1C, $1D, or $1E, respectively.
`(`	Defines the menu item as disabled.

You can specify any, all, or none of these metacharacters in the text string. The metacharacters that you specify aren't displayed in the menu item. (To use any of these metacharacters in the text of a menu item, first use `AppendMenu`, specifying at least one character as the item's text, and then use the `SetMenuItemText` procedure to set the item's text to the desired string.)

Note
If you add menu items using the `AppendMenu` procedure, you should define the text and any marks or keyboard equivalents in resources for easier localization. ◆

You can specify the first character that defines the text of a menu item as a hyphen to create a divider line. The string in the `data` parameter can be blank (containing one or more spaces), but it should not be an empty string.

If you do not define a specific characteristic of a menu item, the `AppendMenu` procedure assigns the default characteristic to the menu item. If you do not define any characteristic other than the text for a menu item, the `AppendMenu` procedure inserts the menu item so that it appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

You can use `AppendMenu` to append items to a menu regardless of whether the menu is in the current menu list.

SEE ALSO

See "Adding Items to a Menu" on page 3-64 for examples of appending items to a menu.

InsertMenuItem

Use the `InsertMenuItem` procedure to insert one or more items to a menu previously created using `NewMenu`, `GetMenu`, or `GetNewMBar`.

The `InsertMenuItem` procedure is also available as the `InsMenuItem` procedure.

```
PROCEDURE InsertMenuItem (theMenu: MenuHandle; itemString: Str255;
                          afterItem: Integer);
```

`theMenu`	A handle to the menu record of the menu to which you wish to add the menu item or items.
`itemString`	A string that defines the characteristics of the new menu items. Note that in most cases you should store the text of a menu item in a resource, so that your menu items can be more easily localized. You can specify the contents of the `itemString` parameter using metacharacters; the `InsertMenuItem` procedure accepts the same metacharacters as the `AppendMenu` procedure. However, if you specify multiple items, the `InsertMenuItem` procedure inserts the items in the reverse of their order in the `itemString` parameter.
`afterItem`	The item number of the menu item after which the new menu items are to be added. Specify 0 in the `afterItem` parameter to insert the new items before the first menu item; specify the item number of a current menu item to insert the new menu items after it; specify a number greater than or equal to the last item in the menu to append the new items to the end of the menu.

DESCRIPTION

The `InsertMenuItem` procedure inserts any defined menu items to the specified menu. The menu items are inserted according to the location specified by the `afterItem` parameter. You specify the text of any menu items and their characteristics in the `itemString` parameter. You can embed metacharacters in the string you specify to define various characteristics of a menu item. The metacharacters aren't displayed in the menu.

Here are the metacharacters you can specify in the `itemString` parameter:

Metacharacter	Description
`;` or Return	Separates menu items.
`^`	When followed by an icon number, defines the icon for the item. If the keyboard equivalent field contains $1C, this number is interpreted as a script code.
`!`	When followed by a character, defines the mark for the item. If the keyboard equivalent field contains $1B, this value is interpreted as the menu ID of a submenu of this menu item.

Metacharacter	Description
<	When followed by one or more of the characters B, I, U, O, and S, defines the character style of the item to Bold, Italic, Underline, Outline, or Shadow, respectively.
/	When followed by a character, defines the keyboard equivalent for the item. When followed by $1B, specifies that this menu item has a submenu. To specify that the menu item has a script code, small icon, or reduced icon, use the `SetItemCmd` procedure to set the keyboard equivalent field to $1C, $1D, or $1E, respectively.
(	Defines the menu item as disabled.

You can specify any, all, or none of these metacharacters in the text string. The metacharacters that you specify aren't displayed in the menu item. To use any of these metacharacters in the text of a menu item, first use `InsertMenuItem`, specifying at least one character as the item's text, and then use the `SetMenuItemText` procedure to set the item's text to the desired string.

Note
If you add menu items using the `InsertMenuItem` procedure, you should define the text and any marks or keyboard equivalents in resources for easier localization. ◆

You can specify the first character that defines the text of a menu item as a hyphen to create a divider line. The string in the `itemString` parameter can be blank (containing one or more spaces), but it should not be an empty string.

If you do not define a specific characteristic of a menu item, the `InsertMenuItem` procedure assigns the default characteristic to the menu item. If you do not define any characteristic other than the text for a menu item, the `InsertMenuItem` procedure inserts the menu item so that it appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent.

You can use `InsertMenuItem` to insert items into a menu regardless of whether the menu is in the current menu list.

SEE ALSO

See "Adding Items to a Menu" beginning on page 3-64 for examples.

DeleteMenuItem

Use the `DeleteMenuItem` procedure to delete an item from a menu. The `DeleteMenuItem` procedure is also available as the `DelMenuItem` procedure.

```
PROCEDURE DeleteMenuItem (theMenu: MenuHandle; item: Integer);
```

`theMenu`	A handle to the menu record of the menu from which you want to delete the menu item.
`item`	The item number of the menu item to delete. If you specify 0 or a number greater than the last item in the menu, `DeleteMenuItem` does not delete any item from the menu.

DESCRIPTION

The `DeleteMenuItem` procedure deletes a specified menu item from a menu. The `DeleteMenuItem` procedure also deletes the item's menu item entry from your application's menu color information table (if an entry exists). You should not delete items from an existing menu unless the user expects the menu (such as a menu that lists open documents) to change.

AppendResMenu

Use the `AppendResMenu` procedure to search all resource files open to your application for a given resource type and to append the names of any resources it finds to a specified menu. The specified menu must have been previously created using `NewMenu`, `GetMenu`, or `GetNewMBar`.

The `AppendResMenu` procedure is also available as the `AddResMenu` procedure.

```
PROCEDURE AppendResMenu (theMenu: MenuHandle; theType: ResType);
```

`theMenu`	A handle to the menu record of the menu to which to append the names of any resources of a given type that `AppendResMenu` finds.
`theType`	A four-character code that identifies the resource type for which to search.

DESCRIPTION

The `AppendResMenu` procedure searches all resource files open to your application for resources of the type defined by the parameter `theType`. It appends the names of any resources it finds of the given type to the end of the specified menu. `AppendResMenu` appends the names of found resources in alphabetical order; it does not alphabetize items already in the menu. The `AppendResMenu` procedure does not add resources with names that begin with a period (`.`) or a percent sign (`%`) to the menu.

The `AppendResMenu` procedure assigns default characteristics to each menu item. Each appended menu item appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent. To get the name or to change other characteristics of an item appended by `AppendResMenu`, use the Menu Manager routines described in "Getting and Setting the Appearance of Menu Items" beginning on page 3-130.

If you specify that AppendResMenu add resources of type 'DRVR' to your Apple menu, AppendResMenu adds the name (and icon) of each item in the Apple Menu Items folder to the menu.

If you specify that AppendResMenu append resources of type 'FONT' or 'FOND', the Menu Manager performs special processing for any resources it finds that have font numbers greater than $4000. If the script system associated with the font name is installed in the system, AppendResMenu stores information in the itemDefinitions array (in the itemIcon and itemCmd fields for that item) in the menu's menu record. This allows the Menu Manager to display the font name in the correct script.

SPECIAL CONSIDERATIONS

The AppendResMenu procedure calls the Resource Manager procedure SetResLoad (specifying TRUE in the load parameter) before returning. The AppendResMenu procedure reads the resource data of the resources it finds into memory. If your application does not want the Resource Manager to read resource data into memory when your application calls other routines that read resources, you need to call SetResLoad and specify FALSE in the load parameter after AppendResMenu returns.

SEE ALSO

Listing 3-15 on page 3-69 shows a sample that adds items from the Apple Menu Items folder to the Apple menu, and Listing 3-16 on page 3-70 shows a sample that adds font names to a menu. See *Inside Macintosh: More Macintosh Toolbox* for information on the Resource Manager.

InsertResMenu

Use the InsertResMenu procedure to search all resource files open to your application for a given resource type and to insert the names of any resources it finds to a specified menu. The items are inserted after the specified menu item. The specified menu must have been previously created using NewMenu, GetMenu, or GetNewMBar.

```
PROCEDURE InsertResMenu (theMenu: MenuHandle; theType: ResType;
                         afterItem: Integer);
```

theMenu	A handle to the menu record of the menu to which to add the names of any resources of a given type that InsertResMenu finds.
theType	A four-character code that identifies the resource type for which to search.
afterItem	A number that indicates where in the menu to insert the names of any resources of the given type that InsertResMenu finds. Specify 0 in the afterItem parameter to insert the items before the first menu item; specify the item number of a menu item already in the menu to insert the items after the specified item number. If you specify a number greater than or equal to the last item in the menu, the items are inserted at the end of the menu.

DESCRIPTION

The `InsertResMenu` procedure searches all resource files open to your application for resources of the type defined by the parameter `theType`. It inserts the names of any resources it finds of the given type at the specified location in the specified menu. `InsertResMenu` adds the names of found resources in alphabetical order; it does not alphabetize items already in the menu.

The `InsertResMenu` procedure does not add resources with names that begin with a period (`.`) or a percent sign (`%`) to the menu.

The `InsertResMenu` procedure assigns default characteristics to each menu item. Each appended menu item appears in the menu as an enabled item, without an icon or a mark, in the plain character style, and without a keyboard equivalent. To get the name or to change other characteristics of an item appended by `InsertResMenu`, use the Menu Manager routines described in the next section, "Getting and Setting the Appearance of Menu Items."

If you specify that `InsertResMenu` add resources of type `'DRVR'` to your Apple menu, `InsertResMenu` adds the name (and icon) of each item in the Apple Menu Items folder to the menu.

If you specify that `InsertResMenu` add resources of type `'FONT'` or `'FOND'`, the Menu Manager performs special processing for any resources it finds that have font numbers greater than $4000. If the script associated with the font name is currently active, `InsertResMenu` stores information in the `itemDefinitions` array (in the `itemIcon` and `itemCmd` fields for that item) in the menu's menu record that allows the Menu Manager to display the font name in the correct script.

SPECIAL CONSIDERATIONS

The `InsertResMenu` procedure calls the Resource Manager procedure `SetResLoad` (specifying `TRUE` in the `load` parameter) before returning. The `InsertResMenu` procedure reads the resource data of the resources it finds into memory. If your application does not want the Resource Manager to read resource data into memory when your application calls other routines that read resources, you need to call `SetResLoad` and specify `FALSE` in the `load` parameter after `InsertResMenu` returns.

Getting and Setting the Appearance of Menu Items

You can get information about the characteristics of a menu item using Menu Manager routines. For example, you can get an item's text, style, mark, keyboard equivalent, script code, and associated icons. You can also determine if a menu item has a submenu associated with it and the menu ID of the submenu.

You can set the characteristics of a menu item, including associating a submenu with a menu item, using Menu Manager routines. Whenever possible, however, you should define your application's menu items in `'MENU'` resources. This makes your application easier to localize for other regions.

You can also enable and disable menu items or entire menus using Menu Manager routines.

EnableItem

Use the `EnableItem` procedure to enable a menu item or a menu.

```
PROCEDURE EnableItem (theMenu: MenuHandle; item: Integer);
```

`theMenu`	A handle to the menu record of the menu containing the menu item to enable.
`item`	The item number of the menu item to enable, or 0 to enable the entire menu. You cannot individually enable a menu item with an item number greater than 31. If you specify 0 in the `item` parameter, the `EnableItem` procedure enables the menu title and all items in the menu that were not previously individually disabled.

DESCRIPTION

The `EnableItem` procedure enables a specified menu item so that it no longer appears dim and so that the user can choose the menu item.

Note that, if you enable a menu, the `EnableItem` procedure enables the menu title but only enables those menu items that are not currently disabled as a result of your application previously calling `DisableItem` and specifying each item's item number. For example, if all items in your application's Edit menu are enabled, you can disable the Cut and Copy commands individually using `DisableItem`. If you choose to disable the entire menu by passing 0 as the `item` parameter to `DisableItem`, the menu and all its items are disabled. If you then enable the entire menu by passing 0 as the `item` parameter to `EnableItem`, the menu and its items are enabled, except for the Cut and Copy commands, which remain disabled. In this case, to enable the Cut and Copy commands you must enable each one individually using `EnableItem`.

If your application enables a menu using `EnableItem`, it should call `DrawMenuBar` to update the menu bar's appearance.

SEE ALSO

See "Enabling and Disabling Menu Items" on page 3-58 for examples of enabling items in a menu.

DisableItem

Use the `DisableItem` procedure to disable a menu item or an entire menu.

```
PROCEDURE DisableItem (theMenu: MenuHandle; item: Integer);
```

`theMenu`	A handle to the menu record of the menu containing the menu item to disable.
`item`	The item number of the menu item to disable, or 0 to disable the entire menu. You cannot individually disable a menu item with an item number greater than 31. If you specify 0 in the `item` parameter, the `DisableItem` procedure disables the menu title and all items in the menu, including menu items with item numbers greater than 31.

DESCRIPTION

The `DisableItem` procedure disables a specified menu item so that it appears dim and cannot be chosen by the user.

If your application disables a menu using `DisableItem`, your application should call `DrawMenuBar` to update the menu bar's appearance.

SEE ALSO

See "Enabling and Disabling Menu Items" on page 3-58 for examples of disabling items in a menu.

GetMenuItemText

Use the `GetMenuItemText` procedure to get the text of a specific menu item. The `GetMenuItemText` procedure is also available as the `GetItem` procedure.

```
PROCEDURE GetMenuItemText (theMenu: MenuHandle; item: Integer;
                           VAR itemString: Str255);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose text you wish to get.
`item`	The item number of the menu item. The `GetMenuItemText` procedure returns the text of this item.
`itemString`	The `GetMenuItemText` procedure returns the text of the menu item in this parameter.

DESCRIPTION

The `GetMenuItemText` procedure returns the text of the specified menu item in the `itemString` parameter. Use other Menu Manager routines to get information about the other characteristics of a menu item.

SetMenuItemText

Use the `SetMenuItemText` procedure to set the text of a specific menu item to a given string. The `SetMenuItemText` procedure is also available as the `SetItem` procedure.

```
PROCEDURE SetMenuItemText (theMenu: MenuHandle; item: Integer;
                           itemString: Str255);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose text you wish you to set.
`item`	The item number of the menu item. The `SetMenuItemText` procedure sets the text of this item.
`itemString`	The `SetMenuItemText` procedure sets the text of the menu item according to the string specified in the `itemString` parameter. The `SetMenuItemText` procedure does not recognize metacharacters or set any other characteristics of the menu item. The `itemString` parameter can be blank, but it should not be an empty string.

DESCRIPTION

The `SetMenuItemText` procedure sets the text of the specified menu item to the text specified in the `itemString` parameter. The `SetMenuItemText` procedure does not recognize any metacharacters used by the `AppendMenu` and `InsertMenuItem` procedures. Use other Menu Manager routines to set other characteristics of a menu item.

If you set the text of a menu item using the `SetMenuItemText` procedure, you should store the text in a string resource so that your application can be more easily localized.

SEE ALSO

See Listing 3-9 on page 3-59 for an example of setting the text of a menu item.

GetItemStyle

Use the `GetItemStyle` procedure to get the style of the text in a specific menu item.

```
PROCEDURE GetItemStyle (theMenu: MenuHandle; item: Integer;
                        VAR chStyle: Style);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose style you wish to get.
`item`	The item number of the menu item. The `GetItemStyle` procedure returns the style of the text for this item.

`chStyle` The `GetItemStyle` procedure returns the style of the text for this item in the `chStyle` parameter. The `chStyle` parameter is a set defined by the `Style` data type.

```
TYPE
StyleItem = (bold, italic, underline, outline,
             shadow, condense, extend);
Style     = SET OF StyleItem;
```

DESCRIPTION

The `GetItemStyle` procedure returns the style of the text of the specified menu item in the `chStyle` parameter. The returned style can be one or more of the styles defined by the `Style` data type, or it is the empty set if the style of the text is Plain.

SetItemStyle

Use the `SetItemStyle` procedure to set the style of the text in a specific menu item.

```
PROCEDURE SetItemStyle (theMenu: MenuHandle; item: Integer;
                        chStyle: Style);
```

`theMenu` A handle to the menu record of the menu containing the menu item whose style you wish to set.

`item` The item number of the menu item. The `SetItemStyle` procedure sets the style of the text for this item.

`chStyle` The `SetItemStyle` procedure sets the style of the text for this item according to the style described by the `chStyle` parameter. The `chStyle` parameter is a set defined by the `Style` data type.

```
TYPE
StyleItem = (bold, italic, underline, outline,
             shadow, condense, extend);
Style     = SET OF StyleItem;
```

You can set the style to one or more of the styles defined by the `Style` data type, or you can set it to Plain by specifying an empty set in the `chStyle` parameter.

DESCRIPTION

The `SetItemStyle` procedure sets the style of the text of the specified menu item to the style or styles defined by the `chStyle` parameter.

SEE ALSO

See Listing 3-10 on page 3-60 for examples of setting the style of a menu item.

GetItemMark

Use the `GetItemMark` procedure to get the mark of a specific menu item or the menu ID of the submenu associated with the menu item.

```
PROCEDURE GetItemMark (theMenu: MenuHandle; item: Integer;
                        VAR markChar: Char);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose mark or submenu you wish to get.
`item`	The item number of the menu item. The `GetItemMark` procedure returns the mark of this item or, if this item has a submenu associated with it, returns the menu ID of the submenu in the `markChar` parameter.
`markChar`	The `GetItemMark` procedure returns the mark or the submenu of this item in the `markChar` parameter. A menu item can have a mark or a submenu attached to it, but not both. If this menu item has a marking character, the `GetItemMark` procedure returns the mark. If this menu item has a submenu associated with it, the `GetItemMark` procedure returns the menu ID of the submenu. If the item doesn't have a mark or a submenu, `GetItemMark` returns 0 in this parameter.

DESCRIPTION

If the item has a mark or submenu, the `GetItemMark` procedure returns the mark or the menu ID of the submenu of the specified menu item in the `markChar` parameter (or 0 if the item doesn't have a mark or a submenu).

SetItemMark

Use the `SetItemMark` procedure to set the mark of a specific menu item or to change or set the submenu associated with a menu item.

```
PROCEDURE SetItemMark (theMenu: MenuHandle; item: Integer;
                        markChar: Char);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose mark or submenu you wish to set.
`item`	The item number of the menu item. The `SetItemMark` procedure sets the mark or the submenu of this item.
`markChar`	The `SetItemMark` procedure sets the mark or submenu of this item according to the information in the `markChar` parameter.

To set the mark of a menu item, specify the marking character in the `markChar` parameter. You can also use one of these constants to specify that the item has no mark, has a checkmark as the marking character, or has the diamond symbol as the marking character:

```
CONST
noMark       = 0;    {no marking character}
checkMark    = $12;  {checkmark}
diamondMark  = $13;  {diamond symbol}
```

To set the submenu associated with this menu item, specify the menu ID of the submenu in the `markChar` parameter.

DESCRIPTION

The `SetItemMark` procedure sets the mark or the submenu of the specified menu item.

SEE ALSO

See Listing 3-11 on page 3-61 for examples of setting the mark of a menu item.

CheckItem

Use the `CheckItem` procedure to set the mark of a specific menu item to a checkmark or to remove a mark from a menu item.

```
PROCEDURE CheckItem (theMenu: MenuHandle; item: Integer;
                     checked: Boolean);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose mark you wish to set to a checkmark or whose mark you wish to remove.
`item`	The item number of the menu item.
`checked`	The `CheckItem` procedure sets or removes the mark of the item according to the information in the `checked` parameter. To set the mark of a menu item to a checkmark, specify `TRUE` in the `checked` parameter. To remove a checkmark or any other mark from a menu item, specify `FALSE` in the `checked` parameter.

DESCRIPTION

The `CheckItem` procedure sets the mark of the specified menu item to a checkmark or removes any mark from the menu item.

SEE ALSO

See Listing 3-11 on page 3-61 for examples of setting the mark of a menu item.

GetItemIcon

Use the `GetItemIcon` procedure to get the icon or script code of a specific menu item. If the menu item's keyboard equivalent field contains $1C, the returned number represents the script code of the menu item. Otherwise, the returned number represents the item's icon number.

```
PROCEDURE GetItemIcon (theMenu: MenuHandle; item: Integer;
                       VAR iconIndex: Byte);
```

`theMenu` A handle to the menu record of the menu containing the menu item whose icon or script code you wish to get.

`item` The item number of the menu item. The `GetItemIcon` procedure returns the icon number or script code of this item.

`iconIndex` For menu items that do not specify $1C in the keyboard equivalent field, the `GetItemIcon` procedure returns the icon number of the item's icon in this parameter. The icon number returned in this parameter is a value from 1 through 255 if the menu item has an icon associated with it and is 0 otherwise. You can add 256 to the icon number to generate the resource ID of the `'cicn'`, `'ICON'`, or `'SICN'` resource that describes the icon of the menu item. For example, if the `GetItemIcon` procedure returns 5 in this parameter, then the icon of the menu item is described by an icon resource with resource ID 261.

For menu items that contain $1C in the keyboard equivalent field, the `GetItemIcon` procedure returns the script code of the menu item. The Menu Manager displays the menu item using this script code if the corresponding script system is installed.

DESCRIPTION

The `GetItemIcon` procedure returns the icon number or script code of the specified menu item in the `iconIndex` parameter (or 0 if the item doesn't have an icon or a script code).

SetItemIcon

Use the `SetItemIcon` procedure to set the icon number or script code of a specific menu item. Usually you display menu items in the current system script; however, if needed, you can use the `SetItemIcon` procedure to set the script code of a menu item. For an item's script code to be set, the keyboard equivalent field of the item must contain $1C. If the keyboard equivalent field contains any other value, the `SetItemIcon` procedure interprets the specified number as the item's icon number.

```
PROCEDURE SetItemIcon (theMenu: MenuHandle; item: Integer;
                       iconIndex: Byte);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose icon (or script code) you wish to set.
`item`	The item number of the menu item. The `SetItemIcon` procedure sets the icon (or script code) of this item.
`iconIndex`	If the menu item's keyboard equivalent field does not contain $1C, the `SetItemIcon` procedure sets the icon number of the item's icon to the number defined in this parameter. The icon number you specify should be a value from 1 through 255 (or from 1 through 254 if the item has a small or reduced icon) or 0 if the item does not have an icon.

The Menu Manager adds 256 to the icon number to generate the resource ID of the `'cicn'` or `'ICON'` resource that describes the icon of the menu item. For example, if you specify 5 as the value of the `iconIndex` parameter, when the Menu Manager needs to draw the item, it looks for an icon resource with resource ID 261.

If the menu item's keyboard equivalent field contains $1C, the `SetItemIcon` procedure sets the script code of the menu item to the number defined in the `iconIndex` parameter. The Menu Manager displays the menu item using the specified script code if the corresponding script system is installed.

You can specify 0 in the `iconIndex` parameter to indicate that the item uses the current system script and does not have an icon number.

DESCRIPTION

The `SetItemIcon` procedure sets the icon number or script code of the specified menu item to the value in the `iconIndex` parameter.

SEE ALSO

See "Changing the Icon or Script Code of Menu Items" beginning on page 3-62 for examples of setting the icon of a menu item.

GetItemCmd

Use the `GetItemCmd` procedure to get the value of the keyboard equivalent field of a menu item.

```
PROCEDURE GetItemCmd (theMenu: MenuHandle; item: Integer;
                      VAR cmdChar: Char);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose keyboard equivalent field you wish to get.
`item`	The item number of the menu item. The `GetItemCmd` procedure returns the keyboard equivalent field of this item.

`cmdChar`	The value of the item's keyboard equivalent field. The Menu Manager uses this value to map keyboard equivalents to menu commands or to indicate special characteristics of the menu item. If the `cmdChar` parameter contains $1B, the menu item has a submenu; a value of $1C indicates that the item has a script code; a value of $1D indicates that the Menu Manager reduces the item's `'ICON'` resource; and a value of $1E indicates that the item has an `'SICN'` resource.

DESCRIPTION

The `GetItemCmd` procedure returns the value in the keyboard equivalent field of the specified menu item in the `cmdChar` parameter (or 0 if the item doesn't have a keyboard equivalent, submenu, script code, reduced icon, or small icon).

SetItemCmd

Use the `SetItemCmd` procedure to set the value of the keyboard equivalent field of a menu item. You usually define the keyboard equivalents and other characteristics of your menu items in `'MENU'` resources rather than using the `SetItemCmd` procedure.

```
PROCEDURE SetItemCmd (theMenu: MenuHandle; item: Integer;
                      cmdChar: Char);
```

`theMenu`	A handle to the menu record of the menu containing the menu item whose keyboard equivalent field you wish to set.
`item`	The item number of the menu item. The `SetItemCmd` procedure sets the keyboard equivalent field of this item to the value specified in the `cmdChar` parameter.
`cmdChar`	The value of the item's keyboard equivalent field. The Menu Manager uses this value to map keyboard equivalents to menu commands or to define special characteristics of the menu item. To indicate that the menu item has a submenu, specify $1B in the `cmdChar` parameter; specify a value of $1C to indicate that the item has a script code; specify a value of $1D to indicate that the Menu Manager should reduce the item's `'ICON'` resource to the size of a small icon; and specify a value of $1E to indicate that the item has an `'SICN'` resource. The values $01 through $1A, as well as $1F and $20, are reserved for use by Apple. You should not use any of these reserved values in the `cmdChar` parameter.

DESCRIPTION

The `SetItemCmd` procedure sets the value in the keyboard equivalent field of the specified menu item in the `cmdChar` parameter (you can specify 0 if the item doesn't have a keyboard equivalent, submenu, script code, reduced icon, or small icon). If you specify that the item has a submenu, you should provide the menu ID of the submenu as the item's marking character. If you specify that the item has a script code, provide the script code in the icon field of the menu item. If you specify that the item has an `'SICN'` or a reduced `'ICON'` resource, provide the icon number in the icon field of the item.

Disposing of Menus

If you no longer need a menu in the menu list, you can delete the menu using `DeleteMenu`. You should then release the memory associated with that menu using the `DisposeMenu` procedure if you created the menu using `NewMenu`; otherwise, use the Resource Manager procedure `ReleaseResource`. See the chapter "Resource Manager" in *Inside Macintosh: More Macintosh Toolbox* for information on the `ReleaseResource` routine.

DisposeMenu

To release the memory occupied by a menu's associated data structures, use either the `DisposeMenu` procedure or the Resource Manager procedure `ReleaseResource`. Use `DisposeMenu` if you created the menu using `NewMenu`; use `ReleaseResource` if you created the menu using `GetMenu` or read the resource in using `GetNewMBar`.

You should delete the menu from the current menu list using `DeleteMenu` or `ClearMenuBar` before calling the `DisposeMenu` procedure.

```
PROCEDURE DisposeMenu (theMenu: MenuHandle);
```

`theMenu`	A handle to the menu record of the menu you wish to dispose of.

DESCRIPTION

The `DisposeMenu` procedure releases the memory occupied by the specified menu's menu record. The handle that you pass in the parameter `theMenu` is not valid after `DisposeMenu` returns.

SEE ALSO

To delete a menu from the current menu list, see the description of the `DeleteMenu` procedure on page 3-109.

Counting the Items in a Menu

If your application needs to count the number of items in a menu—for example, in a menu that can contain a variable number of menu items such as the Font menu or Help menu—use the `CountMItems` function.

CountMItems

You can count the number of items in a menu using the `CountMItems` function.

```
FUNCTION CountMItems (theMenu: MenuHandle): Integer;
```

`theMenu`	A handle to the menu record of the menu whose items your application needs to count.

DESCRIPTION

The `CountMItems` function counts the number of items in the specified menu and returns as its function result the number of items in the menu.

Highlighting the Menu Bar

You can highlight (invert) a menu title or the entire menu bar using the `FlashMenuBar` procedure. (The `HiliteMenu` procedure highlights only menu titles.) In most cases your application should not highlight the menu bar; use `HiliteMenu` to highlight a menu title.

The user sets the number of times an enabled menu item flashes using the General Controls panel. The `SetMenuFlash` procedure can be used to control the number of times that menu items blink when the user chooses an enabled menu item; usually you should not change the setting chosen by the user.

FlashMenuBar

Use the `FlashMenuBar` procedure to highlight (invert) a menu title or the entire menu bar. You can call `FlashMenuBar` twice in a row to make the menu bar blink.

```
PROCEDURE FlashMenuBar (menuID: Integer);
```

`menuID`	The menu ID of the menu whose title you want to invert. Use 0 in this parameter to invert the entire menu bar. If the specified menu ID does not exist in the current menu list, the `FlashMenuBar` procedure inverts the entire menu bar.

DESCRIPTION

The `FlashMenuBar` procedure inverts the title of the specified menu or inverts the menu bar. To prevent unexpected colors from appearing in the menu bar, you should not call `FlashMenuBar` to invert a menu title while the entire menu bar is inverted.

Only one menu title can be inverted at a time. If no menus are currently highlighted, calling `FlashMenuBar` with a specific menu ID inverts the title of that menu. If you call `FlashMenuBar` again specifying another menu ID that is different from that of the previously inverted menu title, `FlashMenuBar` restores the previously highlighted menu to normal and then inverts the title of the specified menu.

SEE ALSO

You can also highlight a menu using the `HiliteMenu` procedure, described on page 3-119.

SetMenuFlash

Use the `SetMenuFlash` procedure to set the number of times a menu item blinks when the user chooses an enabled menu item. The user sets this value using the General Controls panel, and in most cases your application should not change the value set by the user.

```
PROCEDURE SetMenuFlash (count: Integer);
```

`count`	The number of times an enabled menu item should blink when the user chooses it. This value is initially set to 3 by the General Controls panel. A count of 0 disables the blinking. Values greater than 3 can be slow and distracting to the user.

DESCRIPTION

The `SetMenuFlash` procedure sets the number of times that the Menu Manager causes a menu item to blink when the user chooses an enabled menu item.

The appearance of blinking in a menu item is determined by the menu's menu definition procedure.

ASSEMBLY-LANGUAGE INFORMATION

The global variable `MenuFlash` contains the current count (number of times) a menu item blinks when chosen by the user.

Recalculating Menu Dimensions

The Menu Manager uses the `CalcMenuSize` procedure to recalculate the dimensions of a menu whenever its contents have changed. In most cases your application does not need to use the `CalcMenuSize` procedure.

CalcMenuSize

The `CalcMenuSize` procedure recalculates the horizontal and vertical dimensions of a menu and stores the new values in the `menuWidth` and `menuHeight` fields of the menu record.

```
PROCEDURE CalcMenuSize (theMenu: MenuHandle);
```

`theMenu`	A handle to the menu record of the menu whose dimensions need recalculating.

DESCRIPTION

The `CalcMenuSize` procedure uses the menu definition procedure of the specified menu to calculate the dimensions of the menu.

Managing Entries in the Menu Color Information Table

The Menu Manager maintains color information about an application's menus in a menu color information table. The standard menu definition procedure defines the standard color for the menu bar, titles of menus, text and characteristics of a menu item, and background color of a displayed menu. You can change any of these colors by adding entries to your application's menu color information table. However, note that in most cases your application should use the default colors for its menus.

You can provide an `'mctb'` resource with resource ID 0 as one of your application's resources if you want to use colors other than the default colors for your application's menu bar and menus. (Or you can provide an `'mctb'` resource with the same resource ID as a `'MENU'` resource to define the color entries for a single menu.) You can also add entries to or delete entries from your application's menu color information table using the `SetMCEntries` and `DeleteMCEntries` procedures. You can get information about an entry using the `GetMCEntry` function. To get or set your application's menu color information table, use the `GetMCInfo` function or `SetMCInfo` procedure. To dispose of your application's menu color information table, use the `DisposeMCInfo` procedure.

Note that the menu color information table uses a format that is different from the standard color table format. "The Menu Color Information Table Record" beginning on page 3-98 describes the format of the menu color information table in detail.

GetMCInfo

Use the `GetMCInfo` function to get a handle to a copy of your application's menu color information table.

```
FUNCTION GetMCInfo: MCTableHandle;
```

DESCRIPTION

The `GetMCInfo` function creates a copy of your application's menu color information table and returns a handle to the copy. If the copy fails, `GetMCInfo` returns `NIL`.

SEE ALSO

See "The Menu Color Information Table Record" beginning on page 3-98 for a description of the format of the menu color information table.

SetMCInfo

Use the `SetMCInfo` procedure to set your application's menu color information table.

```
PROCEDURE SetMCInfo (menuCTbl: MCTableHandle);
```

`menuCTbl`	A handle to a menu color information table.

DESCRIPTION

The `SetMCInfo` procedure copies the table specified by the `menuCTbl` parameter to your application's menu color information table. If successful, the `SetMCInfo` procedure is responsible for disposing of your application's current menu color information table, so your application does not need to explicitly dispose of the current table.

Your application should call the Memory Manager function `MemError` to determine whether the `SetMCInfo` procedure successfully copied the table. If the `SetMCInfo` procedure cannot successfully copy the table, it does not dispose of the current menu color information table and the `MemError` function returns a nonzero result code. If the `SetMCInfo` procedure is able to successfully copy the table, it disposes of the current menu color information table and the `MemError` function returns the `noErr` result code.

If the menu color information table specifies a new menu bar color or new menu title colors, your application should call `DrawMenuBar` after calling `SetMCInfo`.

Note that `GetNewMBar` does not save your application's current menu color information table. If your application changes menu bars, you can save and restore your application's current menu color information table by calling `GetMCInfo` before `GetNewMBar` and calling `SetMCInfo` afterward.

SEE ALSO

See "The Menu Color Information Table Record" beginning on page 3-98 for a description of the format of the menu color information table. For an example of using the `GetMCInfo` and `SetMCInfo` routines to save and restore menu color information, see Listing 3-6 on page 3-52. See *Inside Macintosh: Memory* for information on the `MemError` function

DisposeMCInfo

Use the `DisposeMCInfo` procedure to dispose of a menu color information table. The `DisposeMCInfo` procedure is also available as the `DispMCInfo` procedure.

```
PROCEDURE DisposeMCInfo (menuCTbl: MCTableHandle);
```

`menuCTbl`	A handle to a menu color information table.

DESCRIPTION

The `DisposeMCInfo` procedure disposes of the menu color information table referred to by the `menuCTbl` parameter.

GetMCEntry

Use the `GetMCEntry` function to return information about an entry in your application's menu color information table. You can get information about the menu bar entry, a menu title entry, or a menu item entry.

```
FUNCTION GetMCEntry (menuID: Integer; menuItem: Integer)
                    : MCEntryPtr;
```

`menuID`	The menu ID that the `GetMCEntry` function should use to return information about the menu color information table. Specify 0 in the `menuID` parameter (and the `menuItem` parameter) to get the menu bar entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and 0 in the `menuItem` parameter to get a specific menu title entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and an item number in the `menuItem` parameter to get a specific menu item entry.
`menuItem`	The menu item that the `GetMCEntry` function should use to return information about the menu color information table. If you specify 0 in this parameter, `GetMCEntry` returns either the menu bar entry or the menu title entry, depending on the value of the `menuID` parameter. If you specify the item number of a menu item in this parameter and the menu ID of a menu in the current menu list in the `menuID` parameter, `GetMCEntry` returns a specific menu item entry.

DESCRIPTION

The `GetMCEntry` function returns a menu bar entry, a menu title entry, or a menu item entry according to the values specified in the `menuID` and `menuItem` parameters. If the `GetMCEntry` function finds the specified entry in your application's menu color information table, it returns a pointer to a record of data type `MCEntry`. If the specified entry is not found, `GetMCEntry` returns `NIL`.

▲ **WARNING**
The menu color information table is relocatable, so the pointer returned by the `GetMCEntry` function may not be valid across routines that may move or purge memory. Your application should make a copy of the menu color entry record if necessary. ▲

SEE ALSO

"The Menu Color Information Table Record" beginning on page 3-98 describes the entries in a menu color information table.

SetMCEntries

Use the `SetMCEntries` procedure to set entries in your application's menu color information table. You can set any or all of your application's menu item entries and menu title entries or the menu bar entry.

```
PROCEDURE SetMCEntries (numEntries: Integer;
                        menuCEntries: MCTablePtr);
```

`numEntries` The number of entries contained in the array of menu color entry records.

`menuCEntries`
A pointer to an array of menu color entry records. Specify the number of records in the array in the `numEntries` parameter.

DESCRIPTION

The `SetMCEntries` procedure sets any specified menu bar entry, menu title entry, or menu item entry according to the values specified in the menu color entry records. If an entry already exists for a specified menu color entry, the `SetMCEntries` procedure updates the entry in your application's menu color information table with the new values. If the entry doesn't exist, it is added to your application's menu color information table.

If any of the added entries specify a new menu bar color or new menu title colors, your application should call `DrawMenuBar` to update the menu bar with the new colors.

SPECIAL CONSIDERATIONS

The `SetMCEntries` procedure may move or purge memory. Your application should make sure that the array specified by the `menuCEntries` parameter is nonrelocatable before calling `SetMCEntries`.

SEE ALSO

"The Menu Color Information Table Record" beginning on page 3-98 describes the entries in a menu color information table.

DeleteMCEntries

Use the `DeleteMCEntries` procedure to delete one or all entries for a specific menu from your application's menu color information table. You can delete a menu item entry, a menu title entry, the menu bar entry, or all menu item entries of a specific menu. The `DeleteMCEntries` procedure is also available as the `DelMCEntries` procedure.

```
PROCEDURE DeleteMCEntries (menuID: Integer; menuItem: Integer);
```

`menuID` The menu ID that the `DeleteMCEntries` procedure should use to determine which entry to delete from the menu color information table. Specify 0 in the `menuID` parameter (and the `menuItem` parameter) to delete the menu bar entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and 0 in the `menuItem` parameter to delete a specific menu title entry. Specify the menu ID of a menu in the current menu list in the `menuID` parameter and an item number in the `menuItem` parameter to delete a specific menu item entry.

`menuItem` The menu item that the `DeleteMCEntries` procedure should use to determine which entry to delete from the menu color information table. If you specify 0 in this parameter, `DeleteMCEntries` deletes either the menu bar entry or menu title entry, depending on the value of the `menuID` parameter. If you specify the item number of a menu item in this parameter and the menu ID of a menu in the current menu list in the `menuID` parameter, `DeleteMCEntries` deletes a specific menu item entry. You can also delete all menu item entries for a specific menu from your application's menu color information table using this constant:

```
CONST
mctAllItems     = -98;  {delete all menu item entries }
                        { for the specified menu}
```

DESCRIPTION

The `DeleteMCEntries` procedure deletes a menu bar entry, a menu title entry, a menu item entry, or all menu item entries of a given menu, according to the values specified in the `menuID` and `menuItem` parameters. If the `GetMCEntry` function does not find the specified entry in your application's menu color information table, it does not delete the entry. Your application should not delete the last entry in your application's menu color information table.

If any of the deleted entries changes the menu bar color or a menu title color, your application should call `DrawMenuBar` to update the menu bar.

Application-Defined Routine

Apple provides a standard menu definition procedure and standard menu bar definition function. The Menu Manager uses the menu definition procedure and menu bar definition function to display and perform basic operations on menus and the menu bar. Although the Menu Manager allows you to provide your own menu bar definition function, Apple recommends that you use the standard menu bar definition function. Similarly, in most cases the standard menu definition procedure should meet the needs of most applications. However, if your application has special needs, you can choose to provide your own menu definition procedure. If you do so, define your menu definition procedure so that it emulates the standard behavior of menus as much as possible. If you define your own menus, they should follow the guidelines described in this chapter and in *Macintosh Human Interface Guidelines*.

The Menu Definition Procedure

The Menu Manager uses the menu definition procedure of a menu to draw the menu items in the menu, to determine which item the user chose from the menu, and to calculate the menu's dimensions. If you provide your own menu definition procedure, it should also perform these tasks.

Apple provides a standard menu definition procedure, stored as a resource in the System file. The standard menu definition procedure is the `'MDEF'` resource with resource ID 0. When you define your menus, you specify the menu definition procedure the Menu Manager should use when managing them. You'll usually want to use the standard menu definition procedure for your application. However, if you need a feature not provided by the standard menu definition procedure (for example, if you want to include more graphics in your menus), you can choose to write your own menu definition procedure.

MyMenuDef

You can provide your own menu definition procedure if you need special features in a menu other than those provided by the standard menu definition procedure. This section describes how to define your own menu definition procedure, defines the parameters passed to your procedure by the Menu Manager, and describes the general actions your procedure should perform.

```
PROCEDURE MyMenuDef (message: Integer; theMenu: MenuHandle;
                     VAR menuRect: Rect; hitPt: Point;
                     VAR whichItem: Integer);
```

`message` A number that identifies the operation that the menu definition procedure should perform. The `message` parameter can contain any one of these values:

```
CONST
      mDrawMsg       = 0; {draw the menu}
      mChooseMsg     = 1; {tell which item was chosen }
                          { and highlight it}
      mSizeMsg       = 2; {calculate menu dimensions}
      mPopUpMsg      = 3; {calculate rectangle of }
                          { the pop-up box}
```

Your menu definition procedure should not respond to any value other than the four constants listed above.

`theMenu` A handle to the menu record of the menu that the operation should affect.

`menuRect` The rectangle (in global coordinates) in which the menu is located; the Menu Manager provides this information to the menu definition procedure only when the value in the `message` parameter is the `mDrawMsg` or `mChooseMsg` constant.

When the value in the `message` parameter is the `mPopUpMsg` constant, the menu definition procedure should calculate and then return the dimensions of the pop-up box in this parameter. When the value in the `message` parameter is the `mSizeMsg` constant, the menu definition procedure should calculate the horizontal and vertical dimensions of the menu rectangle and store these values in the `menuWidth` and `menuHeight` fields of the menu record.

`hitPt` A mouse location (in global coordinates). The Menu Manager provides information in this parameter to the menu definition procedure when the value in the `message` parameter is the `mChooseMsg` or `mPopUpMsg` constant. When the menu definition procedure receives the `mChooseMsg` constant in the `message` parameter, it should determine whether the mouse location specified in the `hitPt` parameter is in an enabled menu item and highlight or unhighlight the item specified in the `whichItem` parameter appropriately. When the menu definition procedure receives the `mPopUpMsg` constant in the `message` parameter, the `hitPt` parameter contains the top-left coordinates of the closed pop-up box, which your procedure can use to calculate the rectangle of the open pop-up box.

`whichItem` The item number of the last item chosen from this menu (or 0 if an item hasn't been chosen). The Menu Manager provides information in this parameter to the menu definition procedure when the value in the `message` parameter is the `mChooseMsg` constant. When the menu definition procedure receives the `mChooseMsg` constant in the `message` parameter, it should determine whether the mouse location specified in the `hitPt` parameter is in an enabled menu item. If so, the menu definition procedure should unhighlight the item specified by the `whichItem` parameter, highlight the new item, and return the new item number in `whichItem`. If the mouse location isn't in an enabled menu item, the menu definition procedure should unhighlight the item specified by the `whichItem` parameter and return 0 in the `whichItem` parameter.

DESCRIPTION

The Menu Manager calls your menu definition procedure whenever it needs your definition procedure to perform a certain action on a specific menu. The action your menu definition procedure should perform depends on the value of the `message` parameter.

If you provide your own menu definition procedure, store it in a resource of type `'MDEF'` and include its resource ID in the description of each menu that uses your own definition procedure. If you create a menu using `GetMenu` (or `GetNewMBar`), the Menu Manager reads the menu definition procedure into memory and stores a handle to it in the `menuProc` field of the menu's menu record.

If you create a menu using `NewMenu`, the Menu Manager stores a handle to the standard menu definition procedure in the `menuProc` field of the menu's menu record. In this case you must replace the value in the `menuProc` field with a handle to your own procedure and then call the `CalcMenuSize` procedure. If your menu definition procedure is in a resource file, you can get its handle by using the Resource Manager to read it from the resource file into memory. However, note that you should usually store your menus in resources (rather than using `NewMenu`) to make your application easier to localize. See the "Resource Manager" chapter in *Inside Macintosh: More Macintosh Toolbox* for information on the Resource Manager.

The menu definition procedure is responsible for drawing the contents of the menu and its menu items, determining whether the cursor is in a displayed menu, highlighting and unhighlighting menu items, and calculating a menu's dimensions.

When the Menu Manager requests your menu definition procedure to perform an action on a menu, it provides your procedure with a handle to its menu record. This allows your procedure to access the data in the menu record and to use any data in the variable data portion of the menu record to appropriately handle the menu items.

When the Menu Manager creates a menu as a result of an application calling `GetMenu` or `GetNewMBar`, it fills out the `menuID`, `menuProc`, `enableFlags`, `menuTitle`, and `itemDefinitions` fields of the menu record according to its resource definition. If the menu is managed by your menu definition procedure, the Menu Manager calls your procedure (specifying `mSizeMsg`) to calculate and fill in the `menuHeight` and `menuWidth` fields of the menu record. The menu items are described by a variable length field (`itemDefinitions`) in the menu record. Your menu definition procedure can define and use this variable-length data in any manner it chooses.

For pop-up menus that are not implemented as controls, the Menu Manager uses the menu definition procedure to support pop-up menus. If your menu definition procedure supports pop-up menus, it should respond appropriately to the `mPopUpMsg` constant.

The Menu Manager specifies the `mPopUpMsg` constant in the `message` parameter and calls your menu definition procedure whenever it needs to calculate the rectangle bounded by the pop-up box for a pop-up menu that is managed by your menu definition procedure. The parameter `theMenu` contains a handle to the menu record of the pop-up menu, the `hitPt` parameter contains the top-left coordinates of the pop-up box, and `whichItem` contains the previously chosen item. Your menu definition procedure should calculate the rectangle in which the pop-up menu is to appear and return this rectangle in the `menuRect` parameter. If the menu is so large that it scrolls, return the actual top of the menu in the `whichItem` parameter. For pop-up menus, your menu definition procedure also must place the pop-up menu's scrolling information in the global variables `TopMenuItem` and `AtMenuBottom`. Place in `TopMenuItem` the pixel value of the top of the scrollable menu, and place in `AtMenuBottom` the pixel value of the bottom of the scrollable menu.

Note
Your menu definition procedure should not assume that the A5 register is properly set up, so your procedure can't refer to any of the QuickDraw global variables. ◆

SEE ALSO

For additional information on how your menu definition procedure should respond when it receives the `mDrawMsg`, `mChooseMsg`, or `mSizeMsg` constant in the `message` parameter, see "Writing Your Own Menu Definition Procedure" beginning on page 3-87.

Resources

This section describes the menu (`'MENU'`) resource, menu bar (`'MBAR'`) resource, and menu color information table (`'mctb'`) resource. Usually you should define your menus using `'MENU'` resources, define the menus in your menu bar in an `'MBAR'` resource, and use the `GetNewMBar` function to read in the descriptions of your menus and menu bar.

If you want to use colors other than the default colors in a menu, you can provide an `'mctb'` resource with the same resource ID as its corresponding `'MENU'` resource, or you can provide an `'mctb'` resource with resource ID 0 to define colors for all your menus and your menu bar.

If you choose to provide your own menu definition procedure, you should store your routine in an `'MDEF'` resource.

To create a `'MENU'`, an `'MBAR'`, or an `'mctb'` resource, either you can specify the resource description in an input file and compile the resource using a resoure compiler, such as Rez, or you can directly create your resources in a resource file using a tool such as ResEdit. This section describes the structures of these resources after they are compiled by the Rez resource compiler. If you are interested in creating the Rez input files for these resources, see "Using the Menu Manager," beginning on page 3-41, for detailed information.

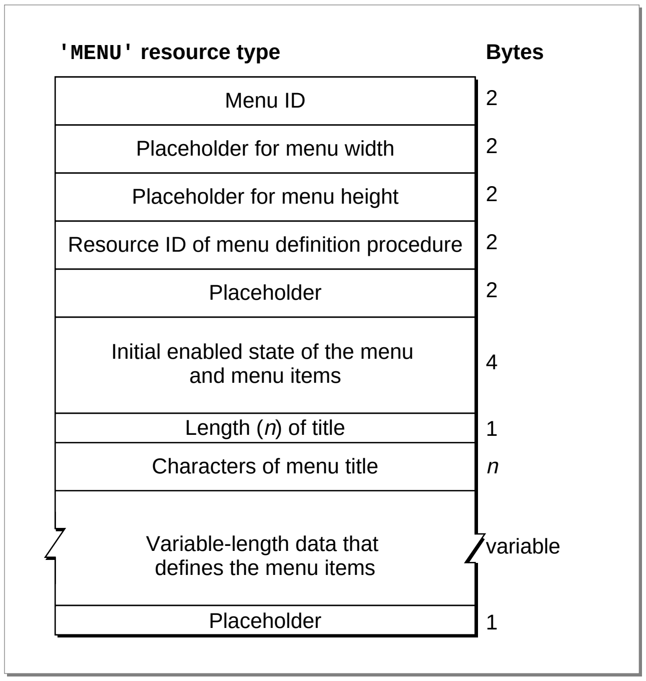
The Menu Resource

You can provide descriptions of your menus in `'MENU'` resources and use the `GetMenu` function or `GetNewMBar` function (if you also provide an `'MBAR'` resource) to read in the descriptions of your menus. After reading in the resource description, the Menu Manager stores the information about specific menus in menu records.

▲ **WARNING**
Menus in a resource must not be purgeable. ▲

Figure 3-37 shows the format of a compiled `'MENU'` resource. See Listing 3-1 on page 3-43 for a description of a `'MENU'` resource in Rez input format.

Figure 3-37 Structure of a compiled menu (`'MENU'`) resource


'MENU' resource type
Bytes
Menu ID
2
Placeholder for menu width
2
Placeholder for menu height
2
Resource ID of menu definition procedure
2
Placeholder
2
Initial enabled state of the menu and menu items
4
Length (n) of title
1
Characters of menu title
n
Variable-length data that defines the menu items
variable
Placeholder
1

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

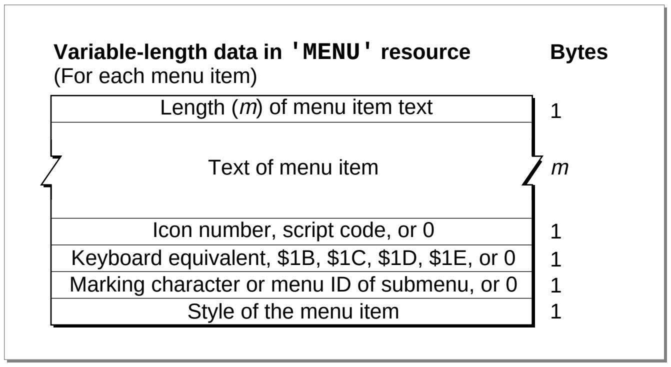
- Menu ID. Each menu in your application should have a unique menu ID. Note that the menu ID does not have to match the resource ID, although by convention most applications assign the same number for a menu's resource ID and menu ID. A negative menu ID indicates a menu belonging to a desk accessory (except for submenus of 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 desk accessory. 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's menu definition procedure to calculate the width and height of the menu and to store these values in the `menuWidth` and `menuHeight` fields of the menu record.
- Resource ID of the menu's menu definition procedure. If the integer 0 appears here (as specified by the `textMenuProc` constant in the Rez input file), the Menu Manager uses the standard menu definition procedure to manage the menu. If you provide your own menu definition procedure, its resource ID should appear in these bytes.

After reading in the menu's resource data, the Menu Manager reads in the menu definition procedure, if necessary. The Menu Manager stores a handle to the menu's menu definition procedure in the `menuProc` field of the menu record.

- 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 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 procedure, you can define and provide this variable-length data according to the needs of your procedure. The Menu Manager simply reads in the data for each menu item and stores it as variable data at the end of the menu record. The menu definition procedure is responsible for interpreting the contents of the data. For example, the standard menu definition procedure 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.

If you use the standard menu definition procedure, your `'MENU'` resource should describe the menu items in this manner. For each menu item, you need to provide its text, the icon number, the keyboard equivalent or other value ($1B to indicate the menu item has a submenu, $1C to indicate a script code other than the system script for the item's text, $1D to indicate the item's icon should be reduced, or $1E to indicate that an `'SICN'` icon should be used), the marking character of the menu item or menu ID of the menu item's submenu, and the font style of the menu item's text. If an item doesn't have a particular characteristic, specify 0 for that characteristic. Figure 3-38 shows the variable-length data portion of a compiled `'MENU'` resource that uses the standard menu definition procedure.

Figure 3-38 The variable-length data that describes menu items as defined by the standard menu definition procedure


Variable-length data in 'MENU' resource
(For each menu item)
Bytes
Length (m) of menu item text
1
Text of menu item
m
Icon number, script code, or 0
1
Keyboard equivalent, $1B, $1C, $1D, $1E, or 0
1
Marking character or menu ID of submenu, or 0
1
Style of the menu item
1

The variable-length data portion of a compiled version of a `'MENU'` resource that uses the standard menu definition procedure contains the following elements:

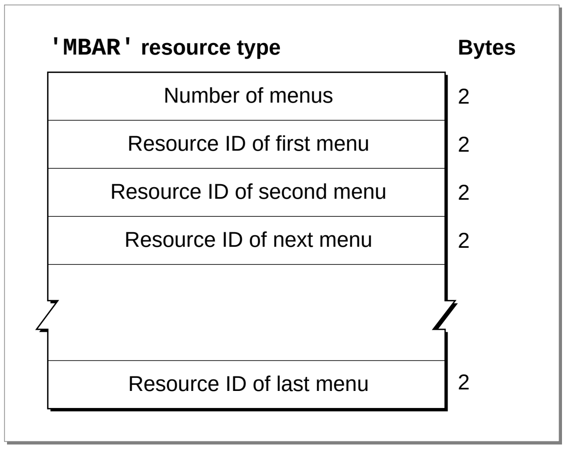
- Length (in bytes) of the menu item's text.
- Text of the menu item.
- Icon number, script code, or 0 (as specified by the `noicon` constant in a Rez input file) if the menu item doesn't contain an icon and uses the system script. The icon number is a number from 1 through 255 (or from 1 through 254 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'` 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, also provide the value $1D in the keyboard equivalent field. If you provide an `'SICN'` resource, provide $1E in the keyboard equivalent field. Otherwise, the Menu Manager looks first for a `'cicn'` resource with the calculated resource ID and uses that icon. If you want the Menu Manager to draw the item's text in a script other than the system script, specify the script code here and also provide $1C 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. An item that is drawn in a script other than the system script cannot also have an icon.
- Keyboard equivalent (specified as a 1-byte character), the value $1B (as specified by the constant `hierarchicalMenu` in a Rez input file) if the item has a submenu, the value $1C if the item uses a script other than the system script, or 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. A menu item can have a keyboard equivalent, a submenu, a small icon, a reduced icon, or a script code, but not more than one of these characteristics. For items containing icons, you can provide $1D in this field if you want the Menu Manager to reduce an `'ICON'` resource to the size of a small icon. Provide $1E if you want the Menu Manager to use an `'SICN'` resource for the item's icon. The values $01 through $1A as well as $1F and $20 are reserved for use by Apple; your application should not use any of these reserved values in this field.
- Marking character, the menu ID of the item's submenu, or 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. Submenus of an application should have menu IDs from 1 through 235; submenus of a desk accessory should have menu IDs from 236 through 255.
- 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 procedure, you should use the same format for your resource descriptions of menus as shown in Figure 3-37. You can use the same format or a format of your choosing to describe menu items. You can also use bits 1–31 of the `enableFlags` field of the menu record as you choose; however, bit 0 must still indicate whether the menu is enabled or disabled.

The Menu Bar Resource

You can describe the order and number of menus in your menu bar in an `'MBAR'` resource, and you can describe your menus in `'MENU'` resources. If you do so, you can use the `GetNewMBar` function to read in the descriptions of your menus and create a new menu list. The Menu Manager stores information about your application's menu bar in a menu list. Figure 3-39 shows the format of a compiled `'MBAR'` resource. (See Listing 3-4 on page 3-49 for a description of an `'MBAR'` resource in Rez input format.)

Figure 3-39 Structure of a compiled menu bar (`'MBAR'`) resource



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

- Number of menus described by this menu bar.
- A variable number (the amount should match the number declared in the first 2 bytes) of resource IDs; each resource ID should identify a `'MENU'` resource.

If you use the `GetNewMBar` function, the Menu Manager places the menus in the menu bar according to the order that they appear in the `'MBAR'` resource.

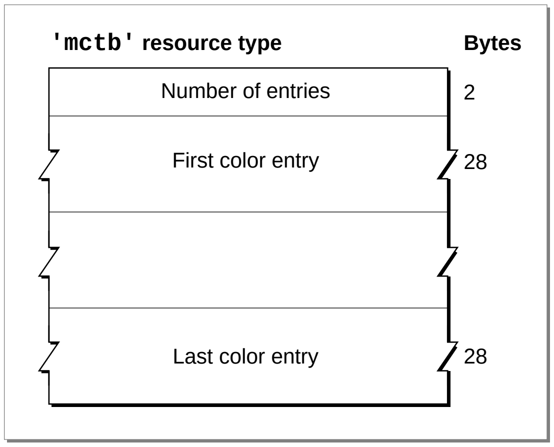
The Menu Color Information Table Resource

To use colors other than the default colors in a menu, provide a **menu color information table** (`'mctb'`) **resource** with the same resource ID as its corresponding `'MENU'` resource. You can also choose to provide an `'mctb'` resource with resource ID 0 to define colors for all your menus and your menu bar. Note that you should usually use the default colors provided by the Menu Manager.

The Menu Manager stores color information about your application's menus and menu bar in a menu color information table. If you provide an `'mctb'` resource with resource ID 0, the Menu Manager reads the resource in when your application calls `InitMenus` and stores the information in your application's menu color information table. If you provide an `'mctb'` resource with the same resource ID as a `'MENU'` resource, when you use `GetMenu` to read in the resource description of the menu (or `GetNewMBar` to read in all menus in the menu bar), the Menu Manager also reads in any associated `'mctb'` resource (if it exists). "The Menu Color Information Table Record" beginning on page 3-98 describes the format of the menu color information table.

Figure 3-40 shows the format of a compiled `'mctb'` resource.

Figure 3-40 Structure of a compiled menu color information table (`'mctb'`) resource



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

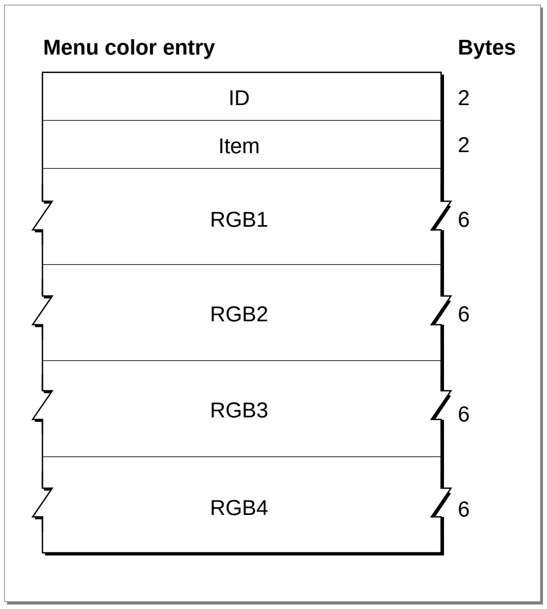
- a count of the number of menu color entry descriptions
- a variable number of menu color entries

A color entry defines colors for various parts of the menu and menu bar. Figure 3-41 on the next page shows the format of a compiled menu color entry in an `'mctb'` resource.

Each menu color entry in an `'mctb'` resource contains the following:

- A menu ID to indicate that this entry is either a menu item entry or menu title entry, 0 to indicate that this entry is a menu bar entry, or –99 to indicate that this is the last entry in this resource.
- An item number to indicate that this entry is a menu item entry, or 0 to indicate that this is either a menu title or menu bar entry. Together, the menu ID and menu item determine how the type of menu color entry is described. See Table 3-7 on page 3-100 for a complete description of how the menu ID and menu item specifications define the type of menu color entry.
- RGB1: for a menu bar entry, the default color for menu titles; for a menu title entry, the title color of a specific menu; for a menu item entry, the mark color for a specific item.
- RGB2: for a menu bar entry, the default background color of a displayed menu; for a menu title entry, the default color for the menu bar; for a menu item entry, the color for the text of a specific item.

Figure 3-41 Structure of a menu color entry in an `'mctb'` resource



- RGB3: for a menu bar entry, the default color of items in a displayed menu; for a menu title entry, the default color for items in a specific menu; for a menu item entry, the color for the keyboard equivalent of a specific item.
- RGB4: for a menu bar entry, the default color of the menu bar; for a menu title entry, the background color of a specific menu; for a menu item entry, the background color of a specific menu.

The Menu Definition Procedure Resource

If you provide your own menu definition procedure, you should store it in a resource of type `'MDEF'`. Provide as the resource data the compiled or assembled code of your menu definition procedure. The entry point of your procedure must be at the beginning of the resource data.

If you define your menus in `'MENU'` resources (and use the `GetMenu` or `GetNewMBar` function), you specify the menu definition procedure that the Menu Manager should use to manage the menu in the `'MENU'` resource. If you use the `NewMenu` function (instead of `'MENU'` resources), your application must explicitly replace the handle to the standard menu definition procedure in the `menuProc` field of the menu record with a handle to the desired menu definition procedure.

Summary of the Menu Manager

Pascal Summary

Constants

```
CONST
      noMark         = 0;  {menu item doesn't have a marking character}

      {values for the message parameter to the menu definition procedure}
      mDrawMsg       = 0;  {draw the menu items of a menu}
      mChooseMsg     = 1;  {highlight or unhighlight a menu item as }
                           { appropriate if the cursor is in a menu item}
      mSizeMsg       = 2;  {calculate the dimensions of a menu}
      mPopUpMsg      = 3;  {calculate the open pop-up box rectangle}

      textMenuProc   = 0;  {resource ID of standard menu definition }
                           { procedure}
      hMenuCmd       = 27; {constant ($1B) specified as keyboard equivalent }
                           { to indicate a menu item has a submenu}
      hierMenu       = -1; {constant used with InsertMenu routine to insert }
                           { a submenu or pop-up menu into the submenu }
                           { portion of the current menu list}
      mctAllItems    = -98;{search for all items with the given ID}
      mctLastIDIndic = -99;{last menu color table entry has this value }
                           { in the ID field of the entry}
```

Data Types

```
TYPE
      MenuInfo =                  {menu record}
      RECORD
         menuID:     Integer;     {number that identifies the menu}
         menuWidth:  Integer;     {width (in pixels) of the menu}
         menuHeight: Integer;     {height (in pixels) of the menu}
         menuProc:   Handle;      {menu definition procedure}
         enableFlags:LongInt;     {indicates whether menu and }
                                  { menu items are enabled}
```

```
   menuData:   Str255;     {title of menu}
   {itemDefinitions}       {variable-length data that }
                           { defines the menu items}
END;

MenuPtr     = ^MenuInfo;   {pointer to a menu record}
MenuHandle  = ^MenuPtr;    {handle to a menu record}

MCEntry =                  {menu color entry record}
RECORD
   mctID:      Integer;    {menu ID or 0 for menu bar}
   mctItem:    Integer;    {menu item number or 0 for }
                           { menu title}
   mctRGB1:    RGBColor;   {usage depends on mctID and }
                           { mctItem}
   mctRGB2:    RGBColor;   {usage depends on mctID and }
                           { mctItem}
   mctRGB3:    RGBColor;   {usage depends on mctID and }
                           { mctItem}
   mctRGB4:    RGBColor;   {usage depends on mctID and }
                           { mctItem}
   mctReserved:Integer;    {reserved}
END;

MCEntryPtr  = ^MCEntry;    {pointer to a menu color entry record}

MCTable     = ARRAY[0..0] OF MCEntry;  {menu color table}
MCTablePtr  = ^MCTable;       {pointer to a menu color table}
MCTableHandle = ^MCTablePtr;  {handle to a menu color table}
```

Menu Manager Routines

Initializing the Menu Manager

```
PROCEDURE InitMenus;
PROCEDURE InitProcMenu      (resID: Integer);
```

Creating Menus

```
FUNCTION NewMenu            (menuID: Integer; menuTitle: Str255)
                             : MenuHandle;
FUNCTION GetMenu            (resourceID: Integer): MenuHandle;
```

Adding Menus to and Removing Menus From the Current Menu List

```
PROCEDURE InsertMenu        (theMenu: MenuHandle; beforeID: Integer);
PROCEDURE DeleteMenu        (menuID: Integer);
PROCEDURE ClearMenuBar;
```

Getting a Menu Bar Description From an 'MBAR' Resource

```
FUNCTION GetNewMBar         (menuBarID: Integer): Handle;
```

Getting and Setting the Menu Bar

```
FUNCTION GetMenuBar: Handle;
PROCEDURE SetMenuBar        (menuList: Handle);
FUNCTION GetMBarHeight: Integer;
```

Drawing the Menu Bar

```
PROCEDURE DrawMenuBar;
PROCEDURE InvalMenuBar;
```

Responding to the User's Choice of a Menu Command

```
FUNCTION MenuSelect         (startPt: Point): LongInt;
FUNCTION MenuKey            (ch: Char): LongInt;
FUNCTION MenuChoice: LongInt;
PROCEDURE HiliteMenu        (menuID: Integer);
FUNCTION PopUpMenuSelect    (menu: MenuHandle;
                             Top: Integer; Left: Integer;
                             PopUpItem: Integer): LongInt;
PROCEDURE SystemMenu        (menuResult: LongInt);
FUNCTION SystemEdit         (editCmd: Integer): Boolean;
```

Getting a Handle to a Menu Record

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
FUNCTION GetMenuHandle      (menuID: Integer): MenuHandle;
FUNCTION HMGetHelpMenuHandle
                            (VAR mh: MenuHandle): OSErr;
```

Adding and Deleting Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
PROCEDURE AppendMenu        (menu: MenuHandle; data: Str255);
PROCEDURE InsertMenuItem    (theMenu: MenuHandle; itemString: Str255;
                             afterItem: Integer);
```

```
PROCEDURE DeleteMenuItem    (theMenu: MenuHandle; item: Integer);
PROCEDURE AppendResMenu     (theMenu: MenuHandle; theType: ResType);
PROCEDURE InsertResMenu     (theMenu: MenuHandle; theType: ResType;
                             afterItem: Integer);
```

Getting and Setting the Appearance of Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
PROCEDURE EnableItem        (theMenu: MenuHandle; item: Integer);
PROCEDURE DisableItem       (theMenu: MenuHandle; item: Integer);
PROCEDURE GetMenuItemText   (theMenu: MenuHandle; item: Integer;
                             VAR itemString: Str255);
PROCEDURE SetMenuItemText   (theMenu: MenuHandle; item: Integer;
                             itemString: Str255);
PROCEDURE GetItemStyle      (theMenu: MenuHandle; item: Integer;
                             VAR chStyle: Style);
PROCEDURE SetItemStyle      (theMenu: MenuHandle; item: Integer;
                             chStyle: Style);
PROCEDURE GetItemMark       (theMenu: MenuHandle; item: Integer;
                             VAR markChar: Char);
PROCEDURE SetItemMark       (theMenu: MenuHandle; item: Integer;
                             markChar: Char);
PROCEDURE CheckItem         (theMenu: MenuHandle; item: Integer;
                             checked: Boolean);
PROCEDURE GetItemIcon       (theMenu: MenuHandle; item: Integer;
                             VAR iconIndex: Byte);
PROCEDURE SetItemIcon       (theMenu: MenuHandle; item: Integer;
                             iconIndex: Byte);
PROCEDURE GetItemCmd        (theMenu: MenuHandle; item: Integer;
                             VAR cmdChar: CHAR);
PROCEDURE SetItemCmd        (theMenu: MenuHandle; item: Integer;
                             cmdChar: CHAR);
```

Disposing of Menus

```
PROCEDURE DisposeMenu       (theMenu: MenuHandle);
```

Counting the Items in a Menu

```
FUNCTION CountMItems        (theMenu: MenuHandle): Integer;
```

Highlighting the Menu Bar

```
PROCEDURE FlashMenuBar      (menuID: Integer);
PROCEDURE SetMenuFlash      (count: Integer);
```

Recalculating Menu Dimensions

```
PROCEDURE CalcMenuSize      (theMenu: MenuHandle);
```

Managing Entries in the Menu Color Information Table

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
FUNCTION GetMCInfo: MCTableHandle;
PROCEDURE SetMCInfo         (menuCTbl: MCTableHandle);
PROCEDURE DisposeMCInfo     (menuCTbl: MCTableHandle);
FUNCTION GetMCEntry         (menuID: Integer; menuItem: Integer)
                             : MCEntryPtr;
PROCEDURE SetMCEntries      (numEntries: Integer;
                             menuCEntries: MCTablePtr);
PROCEDURE DeleteMCEntries   (menuID: Integer; menuItem: Integer);
```

Application-Defined Routine

```
PROCEDURE MyMenuDef         (message: Integer; theMenu: MenuHandle;
                             VAR menuRect: Rect; hitPt: Point;
                             VAR whichItem: Integer);
```

C Summary

Constants

```
enum {
      #define noMark '\0'  /*menu item doesn't have a marking character*/

      /*values for the message parameter to the menu definition procedure*/
      mDrawMsg       = 0,  /*draw the menu items of a menu*/
      mChooseMsg     = 1,  /*highlight or unhighlight a menu item as */
                           /* appropriate if the cursor is in a menu item*/
      mSizeMsg       = 2,  /*calculate the dimensions of a menu*/
      mPopUpMsg      = 3,  /*calculate the open pop-up box rectangle*/

      textMenuProc   = 0,  /*resource ID of standard menu definition */
                           /* procedure*/
      hMenuCmd       = 27, /*constant ($1B) specified as keyboard */
                           /* equivalent to indicate an item has a submenu*/
      hierMenu       = -1, /*constant used with InsertMenu to insert */
                           /* a submenu or pop-up menu into the submenu */
                           /* portion of the current menu list*/
      mctAllItems    = -98,/*search for all items with the given ID*/
      mctLastIDIndic = -99 /*last menu color table entry has this value */
                           /* in the ID field of the entry*/
   };
```

Data Types

```
struct MenuInfo {          /*menu record*/
         short       menuID;        /*number that identifies the menu*/
         short       menuWidth;     /*width (in pixels) of the menu*/
         short       menuHeight;    /*height (in pixels) of the menu*/
         Handle      menuProc;      /*menu definition procedure*/
         long        enableFlags;   /*indicates whether menu and */
                                    /* menu items are enabled*/
         Str255      menuData;      /*title of menu*/
         /*itemDefinitions*/        /*variable-length data that */
                                    /* defines the menu items*/
      };

typedef struct MenuInfo MenuInfo;        /*pointer to a menu record*/
typedef MenuInfo *MenuPtr, **MenuHandle;  /*handle to a menu record*/

struct MCEntry {           /*menu color entry record*/
         short       mctID;         /*menu ID or 0 for menu bar*/
         short       mctItem;       /*menu item number or 0 for */
                                    /* menu title*/
         RGBColor    mctRGB1;       /*usage depends on mctID and */
                                    /* mctItem*/
         RGBColor    mctRGB2;       /*usage depends on mctID and */
                                    /* mctItem*/
         RGBColor    mctRGB3;       /*usage depends on mctID and */
                                    /* mctItem*/
         RGBColor    mctRGB4;       /*usage depends on mctID and */
                                    /* mctItem*/
         short       mctReserved;   /*reserved*/
      };

typedef struct MCEntry MCEntry;
typedef MCEntry *MCEntryPtr;        /*pointer to a menu color entry record*/
                                    /*menu color table*/
typedef MCEntry MCTable[1], *MCTablePtr, **MCTableHandle;
```

Menu Manager Routines

Initializing the Menu Manager

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

Creating Menus

```
pascal MenuHandle NewMenu   (short menuID, const Str255 menuTitle);
pascal MenuHandle GetMenu   (short resourceID);
```

Adding Menus to and Removing Menus From the Current Menu List

```
pascal void InsertMenu      (MenuHandle theMenu, short beforeID);
pascal void DeleteMenu      (short menuID);
pascal void ClearMenuBar    (void);
```

Getting a Menu Bar Description From an 'MBAR' Resource

```
pascal Handle GetNewMBar    (short menuBarID);
```

Getting and Setting the Menu Bar

```
pascal Handle GetMenuBar    (void);
pascal void SetMenuBar      (Handle menuList);
#define GetMBarHeight()     (* (short*) 0x0BAA)
```

Drawing the Menu Bar

```
pascal void DrawMenuBar     (void);
pascal void InvalMenuBar    (void);
```

Responding to the User's Choice of a Menu Command

```
pascal long MenuSelect      (Point startPt);
pascal long MenuKey         (short ch);
pascal long MenuChoice      (void);
pascal void HiliteMenu      (short menuID);
pascal long PopUpMenuSelect (MenuHandle menu, short top, short left,
                             short popUpItem);
pascal void SystemMenu      (long menuResult);
pascal Boolean SystemEdit   (short editCmd);
```

Getting a Handle to a Menu Record

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal MenuHandle GetMenuHandle
                             (short menuID);
pascal OSErr HMGetHelpMenuHandle
                             (MenuHandle *mh);
```

Adding and Deleting Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal void AppendMenu      (MenuHandle menu, ConstStr255Param data);
pascal void InsertMenuItem  (MenuHandle theMenu,
                             ConstStr255Param itemString,
                             short afterItem);
pascal void DeleteMenuItem  (MenuHandle theMenu, short item);
pascal void AppendResMenu   (MenuHandle theMenu, ResType theType);
pascal void InsertResMenu   (MenuHandle theMenu, ResType theType,
                             short afterItem);
```

Getting and Setting the Appearance of Menu Items

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal void EnableItem      (MenuHandle theMenu, short item);
pascal void DisableItem     (MenuHandle theMenu, short item);
pascal void GetMenuItemText (MenuHandle theMenu, short item,
                             Str255 itemString);
pascal void SetMenuItemText (MenuHandle theMenu, short item,
                             ConstStr255Param itemString);
pascal void GetItemStyle    (MenuHandle theMenu, short item,
                             Style *chStyle);
pascal void SetItemStyle    (MenuHandle theMenu, short item, short chStyle);
pascal void GetItemMark     (MenuHandle theMenu, short item,
                             short *markChar);
pascal void SetItemMark     (MenuHandle theMenu, short item,
                             short markChar);
pascal void CheckItem       (MenuHandle theMenu, short item,
                             Boolean checked);
pascal void GetItemIcon     (MenuHandle theMenu, short item,
                             short *iconIndex);
pascal void SetItemIcon     (MenuHandle theMenu, short item,
                             short iconIndex);
```

```
pascal void GetItemCmd      (MenuHandle theMenu, short item, short
                             *cmdChar);
pascal void SetItemCmd      (MenuHandle theMenu, short item, short cmdChar);
```

Disposing of Menus

```
pascal void DisposeMenu     (MenuHandle theMenu);
```

Counting the Items in a Menu

```
pascal short CountMItems    (MenuHandle theMenu);
```

Highlighting the Menu Bar

```
pascal void FlashMenuBar    (short menuID);
pascal void SetMenuFlash    (short count);
```

Recalculating Menu Dimensions

```
pascal void CalcMenuSize    (MenuHandle theMenu);
```

Managing Entries in the Menu Color Information Table

```
{some routines have two spellings, see Table 3-8 for the alternate spelling}
pascal MCTableHandle GetMCInfo(void);
pascal void SetMCInfo       (MCTableHandle menuCTbl);
pascal void DisposeMCInfo   (MCTableHandle menuCTbl);
pascal MCEntryPtr GetMCEntry(short menuID, short menuItem);
pascal void SetMCEntries    (short numEntries, MCTablePtr menuCEntries);
pascal void DeleteMCEntries (short menuID, short menuItem);
```

Application-Defined Routine

```
pascal void MyMenuDef       (short message, MenuHandle theMenu,
                             Rect *menuRect, Point hitPt,
                             short *whichItem);
```

Assembly-Language Summary

Data Structures

The Menu Information Data Structure

0	`menuID`	word	number that identifies the menu
2	`menuWidth`	word	width (in pixels) of the menu
4	`menuHeight`	word	height (in pixels) of the menu
6	`menuDefHandle`	long	menu definition procedure
10	`menuEnable`	long	enable flags
14	`menuData`	256 bytes	menu title followed by menu item information

Global Variables

`AtMenuBottom`	The pixel value at the bottom of the scrollable menu.
`MBarEnable`	Contains 0 if all menus in the current menu bar belong to an application; contains a nonzero value if all menus belong to a desk accessory.
`MBarHeight`	Contains current height of the menu bar, in pixels.
`MBarHook`	Address of routine that `MenuSelect` calls repeatedly while the mouse button is down.
`MenuCInfo`	Contains a handle to application's menu color information table.
`MenuDisable`	Contains the menu ID and item number of the last item chosen, regardless of whether the item was disabled or enabled.
`MenuFlash`	Contains the current count (number of times) a menu item blinks when chosen by the user.
`MenuHook`	Address of routine that `MenuSelect` calls after a menu title is highlighted and the menu rectangle is calculated but before the menu is drawn.
`TheMenu`	Contains the menu ID of the highlighted menu in the menu bar.
`TopMenuItem`	The pixel value at the top of the scrollable menu.

Result Codes

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