Creating Carbon Menus

Carbon > User Experience

2004-02-23

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

Simultaneously published in the United States and Canada.

Contents

Figures, Tables, and Listings

Carbon Menus Concepts

This chapter gives conceptual information about menus that is useful for developers of Carbon applications.

Note that this chapter does not describe general menu appearance, usage, or behavior. For that information, see *Apple Human Interface Guidelines*.

Components of a Carbon Menu

All menus are displayed in the menu bar, which runs across the top of the main display screen. The menu bar is also called the root menu.

A menu reference (type `MenuRef`) identifies an instance of a menu. This opaque structure contains information about the menu such as its size, position, menu items, and so on.

Each menu can have an associated menu ID, which must be a positive integer that uniquely identifies the menu within the application. Many Menu Manager functions require a menu ID to specify the menu to be acted upon.

Each menu has one or more menu items associated with it.

Menu items have a menu item index associated with them that specifies its position within a given menu. That is, a menu item with index 2 is the second item down in the menu. You can then specify any given menu item in an application by its parent menu and its menu item index.

A menu item may also be a submenu. Sometimes called hierarchical menus, a submenu opens an additional menu and displays another set of menu items. In most cases, you cannot select the submenu itself, but only one of its menu items.

Each menu item can have a command key associated with it. Better known as the keyboard equivalent or command key equivalent, the user can enter this key combination as an alternate way to select the menu item.

Note that the command key equivalent can be either a character code or a virtual keycode. A character code specifies a typeable character (for example, “k”, “K”, or “3”), while a virtual keycode identifies the physical key on the keyboard, some of which may not have a corresponding character (such as F10 or the Delete key). To display such “characterless” keys, the Menu Manager uses special keyboard glyphs.

Note: Using the Keyboard and Mouse pane in System Preferences, users can specify their own keyboard shortcuts for applications.

Each menu item (even those with submenus) can have a command ID associated with it. This ID uniquely identifies the menu item within the application. When the user selects a menu item, the Carbon Event Manager can send an event containing the command ID to your application, where you can take appropriate action. You can also use the command ID internally to find a menu item even if you don't know the item's parent menu.

In Mac OS X v10.3 and later, all standard menu content is drawn using the HIView drawing model. This object-oriented view system for drawing user interface elements improves performance and reduces the amount of code needed for custom objects. If you are using standard menus and menu items, you do not need to worry about adopting HIView as that is done for you. However, if you want to use custom menus, you should learn how to do so using the HIView model. For more details, see *HIView Programming Guide*.

Carbon Menu Tasks

This chapter describes how to create Carbon menus in Interface Builder and load them into your application.

Creating a Menu Using Nibs

While you can create menus by calling various Carbon Menu Manager functions, it is much easier to create them using the Interface Builder tool included with Xcode.

Interface Builder is Apple's graphical user interface layout tool. In true WYSIWYG fashion, you simply drag user interface elements onto windows, menus, and controls to create your interfaces. This information is stored in a nib file, which your application can access using a few simple function calls.

Interface Builder has many advantages over other layout methods:

- The WYSIWYG interface makes it easy to visualize your interface objects.
- Its ease of use allows for experimenting and rapid prototyping.
- Special guides makes it easy to conform to Aqua's layout guidelines.
- Simple APIs make it easy to create interface objects from nib files.

Important: While InterfaceBuilder is associated with Xcode, Apple's development environment, you do not need to use Xcode to take advantage of Interface Builder's nib files.

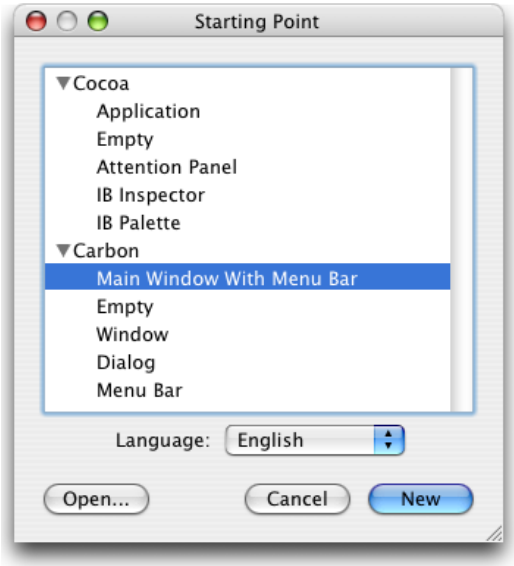
You can use Interface Builder's nib files even if you are working with legacy code. Applications can support both nib-based and older resource-based windows and controls at the same time, so you can make the transition as gradual as you like. Nib file support is available back to Mac OS 8.6 using CarbonLib.

Interface Builder is included on the Xcode CD available with Mac OS X.

The Nib File

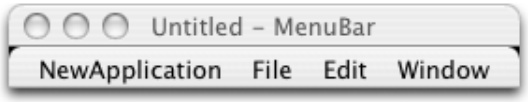
Interface Builder stores all the information about your application's windows, menus, and controls in a nib file (typically named *filename.nib*). When creating a new file, Interface Builder gives you the option of selecting what type of nib file you want to create. When creating interfaces for Carbon applications, you should always select one of the Carbon options, as shown in Figure 2-1 (page 10).

Figure 2-1 Opening dialog for new nib files



When you select Menu Bar or Main Window With Menu Bar, Interface Builder brings up a default menu bar as shown in Figure 2-2 (page 10), which you can then populate with your menus and menu items.

Figure 2-2 The default menu bar

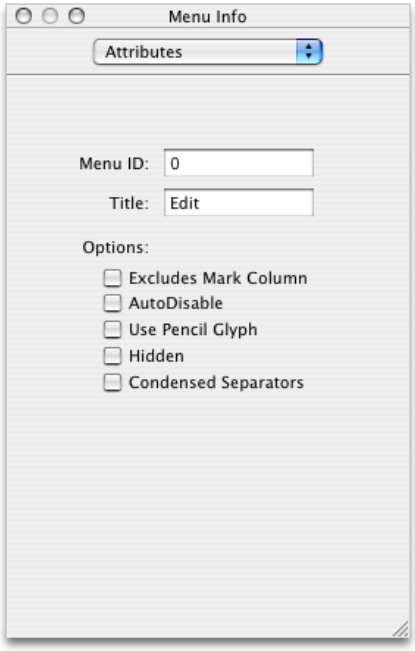


Note that this menu bar already contains the standard menus required for applications (File, Edit, Window, and so on). Clicking on any menu opens it, displaying the standard menu items for each menu.

If your application does not require certain menu items (if your application does not require printing, for example), you can simply select the item and hit Delete to remove it. Similarly, if you do not need an entire menu, you can select its title and delete it.

If you select a menu and then select the Show Info menu item in Interface Builder's Tools menu, you bring up the Info Palette for that menu, as shown in Figure 2-3 (page 11).

Figure 2-3 The Info panel for a menu



The Info panel allows you to set various menu attributes, such as the title, the menu ID, and so on.

Note: The popup menu in the Info window lets you select four other panes: Control, Size, Layout, and Help. However, none of these other panes apply to menus.

The Menu title is self-explanatory. The Menu ID is used to identify the menu in certain Menu Manager calls. You can also set these fields programmatically by calling the Menu Manager functions in Table 2-1 (page 11).

Table 2-1 Menu features

Panel Item	Menu Manager Function Equivalent
Menu ID	`SetMenuID`
Title	`SetMenuTitleWithCFString`

The checkboxes correspond to menu attributes that you can set or unset using the `ChangeMenuAttributes` function, as shown in Table 2-2 (page 11).

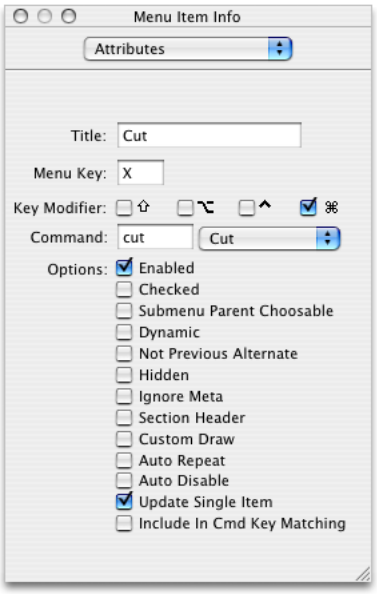
Table 2-2 Menu Info panel attributes

Panel Checkbox	When Checked	Menu Attribute to Pass to ChangeMenuAttributes
Excludes Mark Column	Specifies that the menu shouldn't allocate any column space for mark characters	`kMenuAttrExcludesMarkColumn`

Panel Checkbox	When Checked	Menu Attribute to Pass to ChangeMenuAttributes
AutoDisable	Disables the menu if all its menu items are disabled.	`kMenuAttrAutoDisable`
Use Pencil Glyph	Use the Pencil glyph (Japanese input method menus only)	`kMenuAttrUsePencilGlyph`
Hidden	Menu is hidden	`kMenuAttrHidden`
Use Condensed Separators	Hide extra separators to eliminate blank spaces in the menu.	`kMenuAttrUse-CondensedSeparator`

If you select a menu item, the Show Info palette displays appropriate menu item attributes, as shown in Figure 2-4 (page 12). As with the menu information, the pop up menu lets you select other panes: Control, Size, Layout, and Help. However, except for Help, which lets you assign a help tag (sometimes called a tool tip) to a menu item, none of the other panes apply to menu items.

Figure 2-4 The Info panel for a menu item



The Menu item title is, again, self-explanatory. The Menu Key is the letter or number in the menu item's keyboard equivalent (for example, the C in Command-C). You can then check the appropriate checkbox or checkboxes to select the accompanying combination of Shift, Option, Control, or Command keys required to activate the keyboard equivalent.

The Command field is the four-character code that specifies the menu item's command ID. This is the ID that the Carbon Event Manager sends to your application in a `kEventCommandProcess` event when the user selects this menu item. The popup menu lists a number of predefined command IDs for common menu tasks (such as Cut, Copy, New, and so on).

You can set these fields programmatically by calling the appropriate function in Table 2-3 (page 13).

Table 2-3 Menu item features

Panel Item	Menu Manager Function Equivalent
Title	`SetMenuItemTitleWithCFString`
Menu Key and Keyboard Modifier	`SetMenuItemCommandKey`
Command	`SetMenuItemCommandID`

The checkboxes correspond to various menu item attributes, most of which you can set or unset using the `ChangeMenuItemAttributes` function, as shown in Table 2-4 (page 13).

Table 2-4 Menu Item Attributes

Panel Checkbox	When Checked	Menu Item Attribute to Set
Enabled	The menu item is enabled	Unset `kMenuItemAttr-Disabled`
Checked	The menu item has a check mark	No attribute. Use the `SetItemMark` function to set.
Submenu Parent Choosable	The user can select this parent of a submenu	`kMenuItemAttrSubmenu-ParentChoosable`
Dynamic	Indicates that this menu item is part of a dynamic group. See "Dynamic Menu Items" (page 16) for more information.	`kMenuItemAttrDynamic`
Not Previous Alternate	Indicates that this menu item is not part of the previous dynamic group. See "Dynamic Menu Items" (page 16) for more information.	`kMenuItemAttrNot-PreviousAlternate`
Hidden	Menu item is hidden.	`kMenuItemAttrHidden`
IgnoreMeta	Ignore the dash (-) meta character when drawing this item. Only required if you want to display a dash in your menu item. See Menu Manager Reference for more details.	`kMenuItemAttrIgnoreMeta`
Section Header	Item is a section header. This item is disabled and cannot be selected.	`kMenuItemAttrSection-Header`
Custom Draw	Indicates that this is a custom menu item. The Menu Manager sends the appropriate menu drawing events to your application. See Menu Manager Reference for more details.	`kMenuItemAttrCustomDraw`
Auto Repeat	Indicates that `IsMenuKeyEvent` recognizes this item when an autorepeating keyboard event occurs.	`kMenuItemAttrAutoRepeat`

Panel Checkbox	When Checked	Menu Item Attribute to Set
Auto Disable	Indicates that this item is automatically disabled if the item does not respond to the `kEventCommandUpdateStatus` event. See Menu Manager Reference for more details.	`kMenuItemAttrAutoDisable`
Update Single Item	Update only the menu item with the matching command key when calling `IsMenuKeyEvent`. See Menu Manager Reference for more details.	`kMenuItemAttrUpdateSingleItem`
Use in Cmd Key Matching	Consider this item when using `IsMenuKeyEvent` to match a command key to a menu item.	`kMenuItemAttrIncludeInCmdKeyMatching`

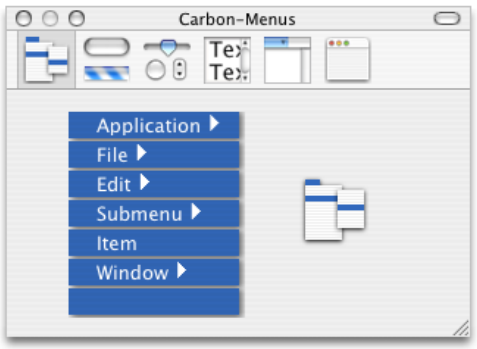
Use `SetMenuItemIconHandle` to set an icon in a menu item.

The Menus Palette

For Carbon applications, Interface Builder provides five different layout palettes, which are displayed in the Carbon palettes window. If the window is not already open, you can do so by choosing Palettes from Interface Builder's Tools menu. This document focuses on the Menu palette, which you can display by selecting the leftmost item in the palette window toolbar. Figure 2-5 (page 14) shows the Menus palette

Note: The Carbon layout palettes differ from those used for Cocoa applications.. Make sure that you select a Carbon-based nib when creating a new nib file.

Figure 2-5 The Menus palette



You add menus or menu items to the menu bar by simply dragging the appropriate blue box from the menu palette.

- The Application element corresponds to the menu containing the application name immediately to the left of the Apple menu. You should rename the menu title to be your application's name. By default, this menu contains only the About *AppName* menu item, but often you will want to add more.

  Note that the system automatically adds additional menu items to the end of the Application menu at run time (Services, Hide, Hide Others, Show All, and Quit).

The system also adds a Preferences menu item at runtime (with the standard Command-, keyboard shortcut), but this is hidden by default. To access the menu item, you need to call `GetIndMenuItemWithCommandID`, searching for the `kHICommandPreferences` command ID. After obtaining the menu item, you can make it visible and otherwise manipulate it.

- The File, Edit, and Window elements correspond to those particular menu types, and they contain the default items for those menu types. For example, the Edit menu contains the Copy, Cut, and Paste items, among others.
- The Submenu element can be used for any nonstandard menus you want to add to the menu bar. If you drag the Submenu item to the menu bar, it becomes a new menu, which you can then rename and populate as necessary. If you drag the Submenu item into an existing menu, it becomes a submenu. You an position the menu as desired by simply dragging it to the desired location.
- The Item element corresponds to a menu item. You can drag an item into any menu or submenu, and position it as desired.
- The empty element corresponds to a separator item. This special menu item appears as a gray line, and is used to group menu items into logical categories.

At run time, the system automatically adds an Apple menu to the left of the Application menu. You should not add any application-specific menu items to the Apple menu.

Important: Interface Builder does not provide many built-in guides for how to arrange menus and menu items, which means that it is your responsibility to make sure your menus correspond to the Aqua specification. Be sure to follow the guidelines in *Apple Human Interface Guidelines* for naming and arranging menus and menu items.

Creating a Simple Menu

This section gives a step-by step example of adding a simple menu to the menu bar. The ideas and methods used apply to any menu you may want to create.

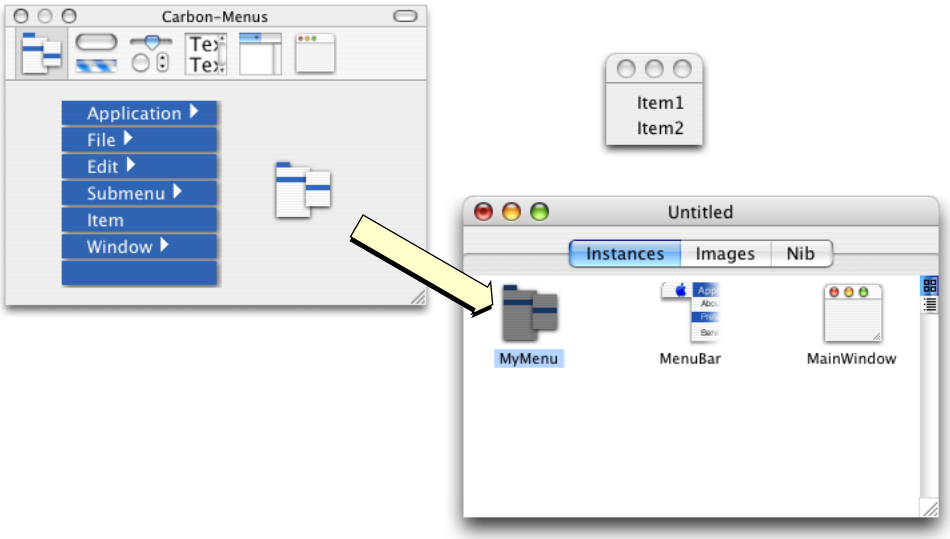
- First, begin with a menu bar. The default menu bar is already populated with the standard Application, File, Edit, and Window menus. If your application uses multiple menu bars, you should assign each a unique name by selecting the title in the main nib window, as shown in figure. You specify this name in your application when it comes time to create a menu bar from the nib.
- To add a new menu to the menu bar, drag a Submenu item from the Menu palette. While dragging, a faint red line appears in the menu bar to indicate where the menu will appear. If you don't like where it ended up, you can simply drag it to a new location, or simply remove it by selecting it and hitting Delete.
- You can rename the menu by double-clicking on it or by changing the title from the Info palette. You can also assign a menu ID if you like.
- Clicking on the menu in the menu bar opens it, displaying a single menu item. If you want to add additional menu items, you can drag them from the Menu palette.
- For each item, you assign a name by double-clicking the item or by changing the title from the Info panel. If desired you can also assign a keyboard equivalent and check any desired attributes. You should also assign a command ID to each item, which will be used in event handling. See "Simple Event Handling" (page 19) for more details.

Standalone Menus

Interface Builder also allows you to create standalone menus that are not part of the menu bar. You may want to create such menus if you want to insert them into the menu bar at some later time. For example, you can use these menus in popup or contextual menus.

To create a standalone menu, drag the menu icon in the lower right of the Menus palette to the Instances pane of the main nib window. A menu then appears in the Instances pane, and a small menu window appears containing two items. By selecting the menu in the instance pane, you can change its title and other attributes in the Info window. You can modify the menu items in a similar fashion, just as if they were in a menu installed in the menu bar.

Figure 2-6 A Standalone Menu



Dynamic Menu Items

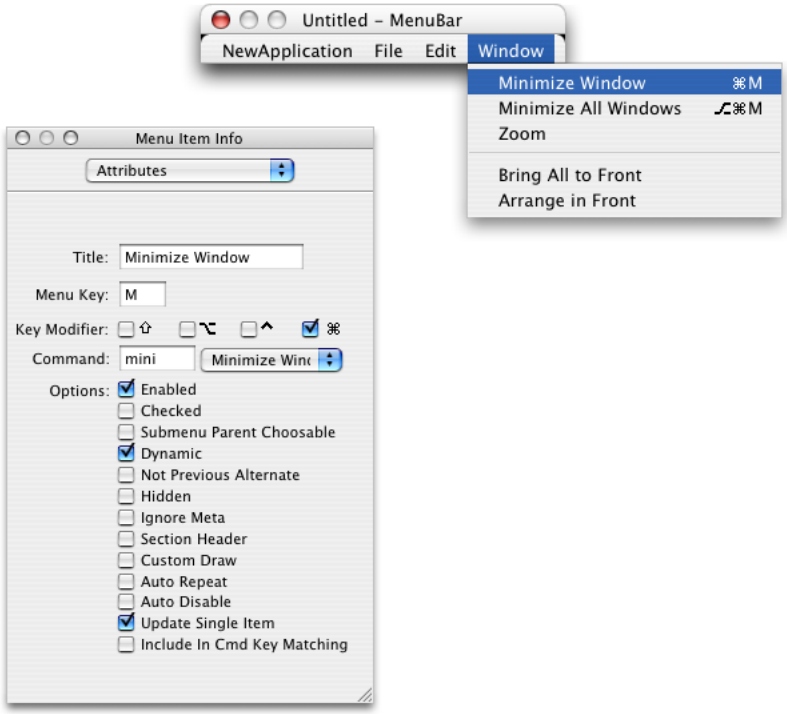
In some cases, you may want to create menu items that change depending on the state of the modifier keys held down by the user. For example, in the default Window menu, a menu item named “Minimize Window” changes to “Minimize All Windows” when the user depresses the Option key while the menu is open. Similarly, if the keyboard equivalent Command-M activates “Minimize Window,” Command-Option-M activates “Minimize All Windows.” Such items are called dynamic menu items.

Creating a dynamic menu item requires you to create two or more menu items in Interface Builder, one for each possible state.

Figure 2-7 (page 17) shows two menu items that make up the Minimize dynamic item. To indicate that they are dynamic and should work together, you must check the Dynamic attribute for each item. In addition, each item must have the same letter or number for its keyboard equivalent. The modifier keys for the keyboard equivalent do not have to “stack.” For example, you can specify that Command-K activate “Remove Mark” while Option-K activates “Remove All Marks.” You can specify several different modifier keys for a single dynamic menu.

Note: You do not have to specify a keyboard equivalent to get dynamic behavior. If you leave the Menu Key blank, depressing only the appropriate modifer key causes the menu item to change.

Figure 2-7 Dynamic menu items



The Menu Manager assumes that if multiple sequential menu items have the same menu key, the are part of the same dynamic group. However, if there is a case where a menu item with the same command key (or no command key at all) should not be considered part of the dynamic group, you can flag it by checking the NotPreviousAlternate attribute for that item. Any menu items that follow are also not considered to be part of the dynamic group.

Creating Menus From a Nib file

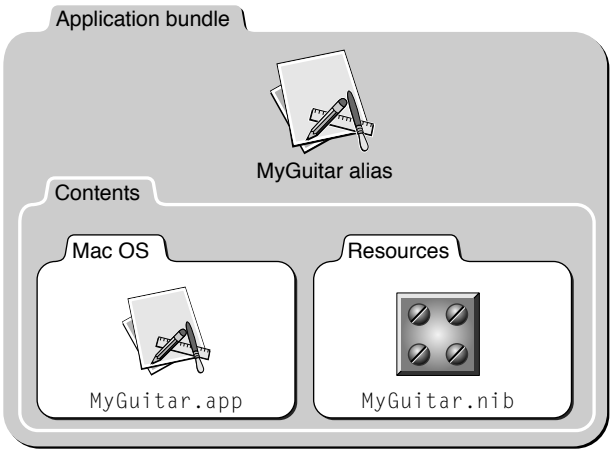
After you have created a nib file containing your menus, you can access them from your application.

Note that while a nib file can contain multiple windows, menus, and so on, to make the best use of resources, you may want to break up your user interface elements among several nib files. For example, you can put your main menu bar and menus in one nib file, windows in another, and so on.

To make sure your application can find the nib file, you should place it in the Resources folder of your application's bundle hierarchy, as shown in Figure 2-8 (page 18). For information about creating application bundles, see *Mac OS X Technology Overview* .

Note: If your nib files contain localizable text, you should create separate nib files for each language you want to support. Each nib should be placed in the appropriate `.lproj` folder within the Resources folder.

Figure 2-8 The nib file in an application bundle



When retrieving menus from a nib, you can either load the entire menu bar with its associated windows, or load a particular menu.

Listing 2-1 (page 18) shows how you to load a menu bar from a nib.

Listing 2-1 Creating a menu bar from a nib file

```
OSStatus err;
IBNibRef theNib;

err = CreateNibReference (CFSTR("MyGuitar"), &theNib);                    // 1
if (!err)
    SetMenuBarFromNib (theNib, CFSTR("GuitarMenu"));                      // 2
```

Here is what the code does:

1. The Interface Builder Services function `CreateNibReference` simply creates a nib reference that points to the specified file. In this case, the file is `MyGuitar.nib` (you don't need to specify the `.nib` extension when calling this function). The `CFSTR` function converts the string into a Core Foundation string, which is the format that `CreateNibReference` expects.

2. The Interface Builder Services function `SetMenuBarFromNib` uses the nib reference to access a menu bar within the nib file. The name of the menu bar (`GuitarMenus` in this example) is the name you assigned to it in the Instances pane of the nib file window. As with the `CreateNibReference` function, `SetMenuBarFromNib` expects a Core Foundation string for the window name, so it must first be converted using `CFSTR`. The created window is stored as a window reference in `theWindow`.

   Note that `SetMenuBarFromNib` automatically sets the menu bar you specified to be visible. If for some reason you want to create a menu bar but don't want it to be immediately visible, you can call `CreateMenuBarFromNib`.

The complete menu bar can now appear in your application. However, while the facade is there (and the menus and menu items are functional), this menu bar does not do anything useful. To make the menu items do useful work, you must attach Carbon event handlers, which are described in detail in "Simple Event Handling" (page 19).

Simple Event Handling

After you have populated the menu bar with your menus and menu items, you need to make them functional, which means they must be able to respond to events. To do so, you must install Carbon event handlers. To get the most out of this section, you should be familiar the Carbon Event Manager, as described in Handling Carbon Events in Carbon Events and Other Input documentation.

For most applications, you should assign a command ID to each menu item. The command ID is a four-character code that uniquely identifies a particular action. When the user selects a menu item, the Carbon Event Manager sends a `kEventCommandProcess` event containing the menu item's command ID to your application. You application can then filter the event to determine the command ID and take the appropriate action.

Note: Command IDs are also used in simple controls, such as push buttons, which initiate a single action when activated. If you have a menu item that performs the same action as a control, you can assign the same command ID to both; doing so means you only need one event handler to handle both cases.

If you assigned a command ID to your menu item, your application is sent command events whenever the menu item is activated. Command events are of the class `kEventClassCommand`.

The Carbon Event Manager defines command ID's for many common commands, such as OK, Cancel, Cut, Paste, and so on. You can also define your own for application-specific commands. Your event handler for the `kEventCommandProcess` event can then determine which command ID was sent and take appropriate action.

Important: Command IDs containing all lower-case letters are defined by Apple; if you create nonstandard command IDs, they must contain at least one upper-case letter.

You assign the command ID to a menu item in the Attributes pane of Interface Builder's Info window, as shown previously in Figure 2-4 (page 12). You can also call the Menu Manager function `SetMenuItemCommandID`.

Note: You can also assign command IDs to controls by using Interface Builder or by calling the Control Manager function `SetControlCommandID`.

The `kEventCommandProcess` event indicates that your menu item was selected. The actual command ID is stored within an `HICommand` structure in the event reference, so you must call the Carbon Event Manager function `GetEventParameter` to retrieve it, as shown in Listing 2-2 (page 19).

Listing 2-2 Obtaining the command ID from the event reference

```
HICommand commandStruct;
UInt32 theCommandID;
```

```
GetEventParameter (event, kEventParamDirectObject,                          // 1
                   typeHICommand, NULL, sizeof(HICommand),
                   NULL, &commandStruct);

theCommandID = commandStruct.commandID;                                     // 2
```

Here is what the code does:

1. When calling `GetEventParameter`, you must specify which parameter you want to obtain. For command events, the direct object (`kEventParamDirectObject`) is the `HICommand` structure, which describes the command that occurred.
2. The command ID of the control (or menu) that generated the event is stored in the `commandID` field of the `HICommand` structure.

To respond to events from menus, you should install your command event handler at the window or application level. Doing so also allows you to use the same handler to catch command events coming from controls, if so desired. Also, attaching your handler at the window level makes sense if you have menu items that apply to one type of document window but not to another.

After handling a command, your application may need to change the state of a menu item. For example, after saving a document, the Save menu item should be disabled until the document changes. Whenever the status of a command item might be in question, the system makes a note of it. When the user takes an action that may require updating the status (such as pulling down a menu), your application receives a `kEventCommandUpdate` event. To make sure that the states of your menus are properly synchronized, you should install a handler for the `kEventCommandUpdate` event. This handler should check the attributes bit of the command event to determine which items may need updating. Some examples of possible updates include

- enabling or disabling menu items
- changing the text of a menu item (for example, from Show xxxx to Hide xxxx).

If the `kHICommandFromMenu` bit in the `attributes` field of the `HICommand` structure (shown in Listing 2-3 (page 20)) is set, then you should check the menu item in question to see if you need to update it.

Listing 2-3 The HICommand structure

```
struct HIComamnd
{
    UInt32  attributes;
    UInt32  commandID;
    struct
    {
        MenuRef         menuRef;
        MenuItemIndex   menuItemIndex;
    } menu;
};
```

Document Revision History

This table describes the changes to *Creating Carbon Menus*.

Date	Notes
2004-02-23	Preliminary draft.

Document Revision History

Glossary

character code A code that identifies a typeable character, as opposed to a particular physical key.

command ID A four character code that uniquely identifies a menu item or control. Note that a menu item and a control can share the same command ID.

keyboard glyph A graphical representation of a physical key that doesn't have a character equivlent (such as a function key or the Shift key).

menu A user interface element that displays a list of possible selections to the user.

menu bar The bar at the top of the main display that holds the list of available menus.

menu ID A unique ID that identifies a menu.

menu item One of the choosable options displayed in a menu.

menu item index A one-based index that identifies a particular menu item in the menu. A menu item index of 3 indicates the third item in the menu.

submenu A menu that is attached to a menu item of another menu. Also called a *hierarchical menu*.

virtual keycode A code that identifies a physical key on a keyboard. Note that virtual keycodes do not have to have a corresponding character code. A key with a virtual keycode is represented in a menu by a keyboard glyph.