Carbon Menu Manager API Preliminary Documentation

For CarbonLib 1.0

Preliminary Draft

Technical Publications

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

Simultaneously published in the United States and Canada.

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

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

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

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

In the following sections, this document discusses the changes to the Menu Manager that are part of the Carbon library which is included with Mac OS 9.0:

IMPORTANT

This is a preliminary document. Although it has been reviewed for technical accuracy, it is not final. Apple Computer, Inc. is supplying this information to help you plan for the adoption of the technologies and programming interfaces described herein. This information is subject to change, and software implemented according to this document should be tested with final operating system

You can check <http://developer.apple.com/techpubs/macos8/SiteInfo/whatsnew.html> for information about updates to this and other developer documents. To receive notification of documentation updates, you can sign up for ADC's free Online Program and receive their weekly Apple Developer Connection News e-mail newsletter. (See <http://developer.apple.com/membership/index.html> for more details about the Online Program.) ▲

Changes to Your Application and the Menu Manager With Carbon

Tips for Working With the Carbon Menu Manager

- Calling `GetMenu` twice on the same resource ID will create two independent, unique menus. Previously, the second call to GetMenu would return the same MenuHandle as the first call.
- You should never call `ReleaseResource` on a menu handle; it's no longer a handle, let alone a resource handle. Always use `DisposeMenu`, which works correctly on Mac OS 8.x and Carbon.
- Use menu accessor functions to access menu data.
- Modify code that uses `EnableItem` or `DisableItem` to use the 8.5 APIs `EnableMenuItem` and `DisableMenuItem`.
- Modify code that sets the `enableFlags` field of the menu directly to use `EnableMenuItem` or `DisableMenuItem` or the Carbon Menu Manager APIs `EnableAllMenuItems` and `DisableAllMenuItems`.
- Modify code that sets low memory in order to hide the menu bar to instead use the 8.5 APIs `HideMenuBar` and `ShowMenuBar`.
- Modify code that uses `GetMenuItemRefCon2` or `SetMenuItemRefCon2` to use the 8.5 APIs `GetMenuItemProperty` and `SetMenuItemProperty`.
- Modify code that creates a menu with a custom menu defproc at runtime (rather than from a resource) to use the new Carbon Menu Manager API `CreateCustomMenu`.

Obsolete APIs

- Remove calls to `AppendResMenu( appleMenu, 'DRVR' )`. Carbon automatically adds items to the Apple menu.
- Remove calls to `OpenDeskAcc` in response to selections from the Apple Menu. Carbon automatically handles selections from the Apple menu.

- Remove calls to `SystemEdit` and `SystemMenu`. Desk accessories are not part of Mac OS X, and applications do not need to support them.
- Remove calls to `InitProcMenu`. Carbon does not support custom menu bar defprocs.
- `EnableItem` and `DisableItem` are removed in favor of `EnableMenuItem` and `DisableMenuItem`.
- `GetMenuItemRefCon2` and `SetMenuItemRefCon2` are removed in favor of the menu item properties.
- The `kMenuDrawItemMsg` message is obsolete.

Obsolete Low-Memory Globals

- `MBarHeight`: Use `GetMBarHeight` or `GetThemeMenuBarHeight`, as appropriate. Also, applications that are setting the menu bar height are most likely doing so to hide the menu bar, and we also now have `HideMenuBar` and `ShowMenuBar`. Therefore, `MBarHeight` will not be supported under Carbon.
- `MBarEnable`: Not supported under Carbon.
- `MenuFlash`: Will be obsolete in future releases of Carbon.
- `MenuHook`: Not supported under Carbon. Removed in favor of a much richer API based on Carbon Events.
- `MBarHook`: Removed in favor of a much richer API based on Carbon Events. Use `kEventMenuOpening` for notification that a menu will be displayed.
- `MenuCInfo`: Use `GetMCInfo` and `SetMCInfo`.
- `MenuList`: This is widely used by applications to examine what's currently in the menu bar. However, it's also a data structure that is shared with the Menu Manager, and will therefore be difficult to support in a multi-threaded world. It will probably be removed and replaced with a new API for accessing the menu bar contents.

Updated Types

The following types have been added to Menus.i and other interface files.

- `UInt16 MenuItemIndex`
- `SInt16 MenuID`

- `UInt32 MenuCommand`

The following types have been changed in Menus.i and other interface files.

- All `MenuHandle` types are now `MenuRef` types.
- All "`UInt16 item`" parameters are now "`MenuItemIndex item`" parameters.

New Apple Menu Behavior

Carbon provides automatic support for the Apple Menu. The Menu Manager will automatically insert an Apple Menu into the menu bar at launch, and will populate it with the contents of the Apple Menu folder. Carbon applications may remove all calls to `AppendResMenu( appleMenu, 'DRVR' )`. The Menu Manager also automatically opens selections from the Apple Menu. Carbon applications should remove all calls to `OpenDescAcc` (which is not exported from CarbonLib).

The default Apple Menu does not have an About item. A Carbon application can customize the section of the Apple menu before the DAs, just as in the past, by calling `InsertMenu` on a menu whose title is the Apple character. The contents of that menu will be inserted as the first items in the Apple Menu. When these items are selected, `MenuSelect` will return the menu ID of the application's Apple menu, as well as an item index based on the position of the item in the application's menu.

The menu list returned by `GetMenuBar` will contain the application's Apple menu, if any. If the application has not supplied a custom Apple menu, no Apple menu will be returned (the default menu is not available to applications).

Accessing Opaque Menu Data

The `MenuHandle` type is now opaque. You may no longer access the `MenuInfo` data structure directly. The `MenuHandle` value returned by `NewMenu` and `GetMenu` is modified so that it is not a valid handle.

Because of opacity, menu handles are no longer resource handles. Therefore:

Calling `GetMenu` twice on the same resource ID will create two independent, unique menus. You should never call `ReleaseResource` on a menu handle; it's no

longer a handle, let alone a resource handle. Always use `DisposeMenu`, which always works correctly.

The following accessor APIs are new:

```
SInt16    GetMenuID( MenuRef menu );

SInt16    GetMenuWidth( MenuRef menu );

SInt16    GetMenuHeight( MenuRef menu );

StringPtr GetMenuTitle( MenuRef menu, StringPtr title );

OSStatus  GetMenuDefinition( MenuRef menu, MenuDefSpecPtr outDefSpec);
```

`GetMenuDefinition` will only return `noErr` if a custom menu definition has already been associated with the menu. If the menu is using the system defproc, it will return `menuUsesSystemDefErr`. There's no way to get a pointer to the system menu definition function.

```
void      SetMenuID( MenuRef menu, SInt16 menuID );

void      SetMenuWidth( MenuRef menu, SInt16 width );

void      SetMenuHeight( MenuRef menu, SInt16 height );

OSStatus  SetMenuTitle( MenuRef menu, ConstStr255Param title );
```

`SetMenuTitle` can return an error if it runs out of memory while installing the new title. After setting the title, you should call `DrawMenuBar` if the menu is currently inserted in the menu bar.

```
OSStatus  SetMenuDefinition( MenuRef menu, const MenuDefSpec* defSpec );
```

`SetMenuDefinition` will send a dispose message to the current menu definition and an init message to the new definition. If the new definition returns an error, `SetMenuDefinition` will return the same error. It also invalidates the menu size so the menu size is recalculated.

Debugging Utilities

Since the Menu Manager data structures are now opaque, we must provide functional access to the contents of those data structures for use while debugging.

APIs added:

```
void GDBShowMenuList( void );

void GDBShowMenuInfo( MenuRef inMenu );

void GDBShowMenuItemInfo( MenuRef inMenu, SInt32 inItem );
```

These are callable from the GNU Debugger (“GDB”), the debugger used with Mach-O applications on Mac OS X.

Accessing Menu Item Data Using Universal Command IDs

You should never have to use a menu item index. If you’ve supplied command IDs for all of your menu items, you should be able to do anything you want with the Menu Manager simply by specifying a command ID; perhaps not even a menu handle.

These APIs allow conversion from a command ID to a `MenuRef` value and item index.

```
ItemCount  CountMenuItemsWithCommandID( MenuRef menu,
      MenuCommand commandID );

OSStatus   GetIndMenuItemWithCommandID( MenuRef menu,
      MenuCommand commandID,
      UInt32 itemIndex,
      MenuRef* outMenu,
      MenuItemIndex* outIndex );
```

`CountMenuItemsWithCommandID` returns the number of menu items with a given command ID found in a menu or its submenus.

`GetIndMenuItemWithCommandID` returns the `MenuRef` and item index for the `itemIndexth` menu item with a given command ID. The `itemIndex` parameter ranges from 1 to `CountMenuItemsWithCommand( menu, cmdID )`.

`CountMenuItemsWithCommandID` and `GetIndMenuItemWithCommandID` are limited in functionality in the first (Sonata) release of CarbonLib. In this release, these APIs only return the first item found with the given command ID. In other words, `CountMenuItemsWithCommandID` always returns either zero or one, and the `itemIndex` parameter to `GetIndMenuItemWithCommandID` must be one.

These APIs allow enabling and disabling menu items by command ID.

```
void    EnableMenuCommand( MenuRef theMenu, MenuCommand commandID );

void    DisableMenuCommand( MenuRef theMenu, MenuCommand commandID );

Boolean IsMenuCommandEnabled( MenuRef menu, MenuCommand commandID );
```

These API complement the existing `GetMenuItemProperty` and `SetMenuItemProperty` APIs.

```
OSStatus  GetMenuCommandProperty(     MenuRef menu,
                                        UInt32 commandID,
                                        OSType propertyCreator,
                                        OSType propertyTag,
                                        UInt32 bufferSize,
                                        UInt32* actualSize,
                                        void* propertyBuffer);


OSStatus  GetMenuCommandPropertySize( MenuRef menu,
                                         UInt32 commandID,
                                        OSType propertyCreator,
                                         OSType propertyTag,
                                        UInt32* size );


OSStatus  SetMenuCommandProperty(     MenuRef menu,
                                        UInt32 commandID,
                                        OSType propertyCreator,
```

```
                                                OSType propertyTag,
                                                UInt32 propertySize,
                                                void* propertyData );


OSStatus  RemoveMenuCommandProperty(  MenuRef menu,
                                                UInt32 commandID,
                                                OSType propertyCreator,
                                                OSType propertyTag );
```

Writing a Carbon MDEF

Carbon MDEF Architecture

We are providing easier pointer-based MDEF usage with the Carbon Menu Manager.

Carbon will not load MDEFs from resources. Rather, we will provide a registry for mapping MDEF ID to defproc function pointer. The `RegisterMenuDefinition` API implements this registry:

```
OSStatus RegisterMenuDefinition ( SInt16 inResID, MenuDefSpecPtr
inDefSpec );
```

`inResID` should be the MDEF ID that's stored in a MENU resource. `GetMenu` will look up the MDEF ID in a MENU resource in the registry and use the associated defproc to implement the menu.

There are two new API for creating a menu:

```
enum
{
   kMenuDefProcPtr   = 0 // raw proc-ptr access based on old MDEF
};
typedef UInt32 MenuDefType;

typedef struct OpaqueMenuLayout* MenuLayoutRef;
```

```
struct MenuDefSpec
{
   MenuDefType defType;
   union
   {
       MenuDefUPP   defProc;
}u;
};

OSStatus   CreateNewMenu(   MenuID menuID,
                                    MenuAttributes attributes,
                                     MenuRef* outMenu );
```

`CreateNewMenu` creates a new menu with the given menu ID and attributes. The new menu has no title. By removing the title parameter, we get closer to the day when you can write a Unicode-aware Mac application without dealing with `Str255`s.

```
OSStatus   CreateCustomMenu( MenuDefSpec* defSpec,
                                    MenuID menuID,
                                    MenuAttributes attributes,
                                    MenuRef* outMenu );
```

`CreateCustomMenu` similarly creates a new menu and installs a pointer-based defproc into the menu.

New MDEF messages:

```
enum
{
   kMenuInitMsg       = 8,
   kMenuDisposeMsg    = 9,
   kMenuFindItemMsg   = 10,
   kMenuHiliteItemMsg = 11
};
```

The `kMenuChooseMsg` message will be removed and replaced by the `kMenuFindItemMsg`.

The `kMenuDrawItemMsg` message will be removed; it had been used by the popup menu control, but isn't anymore since 8.5.

kMenuInitMsg will be sent to the MDEF after the menu has been created. The MenuRef parameter to the defproc is valid; the menuRect and hitPt parameters are unused; the whichItem parameter points to an OSErr error code which the defproc may set if an error occurs during initialization. If the defproc reports an error, the menu will be destroyed and the API which created it will return NULL and an error code. The defproc will not receive a dispose message if it returns an error from the init message.

kMenuDisposeMsg will be sent to the MDEF before the menu is destroyed. The MenuRef parameter to the defproc is valid; all other parameters are unused.

kMenuFindItemMsg will use this new structure:

```
struct MenuTrackingData
{
    MenuRef       menu;
    MenuItemIndex itemSelected;
    MenuItemIndex itemUnderMouse;
    Rect          itemRect;
    SInt32        virtualMenuTop;
    SInt32        virtualMenuBottom;
};
```

The Menu Manager will now send a kMenuFindItemMsg instead of a kMenuChooseMsg. The MenuRef, menuRect, and hitPt parameters are as usual. The whichItem parameter is actually a pointer to a MenuTrackingData structure. On entry, the virtualMenuTop and virtualMenuBottom fields contain the values previously returned by the MDEF, and itemSelected contains the previously selected item. The MDEF should hit-test the given point, scrolling if necessary, and return:

in itemSelected, the item that would actually be selected if the mouse were released here. If the mouse is over a disabled item, return 0.

in itemUnderMouse, the item that's actually under the mouse, regardless of whether the item is enabled.

in itemRect, the rectangle bounding the item under the mouse.

in virtualMenuTop and virtualMenuBottom, updated item coordinates for scrolling.

The MDEF should not hilite the item that is under the mouse during a `kMenuFindItemMsg`. The Menu Manager will send a separate `kMenuHiliteItemMsg` to request the item hiliting.

`kMenuHiliteItemMsg` will use this new structure:

```
struct HiliteMenuItemData
{
   MenuItemIndex   previousItem;
   MenuItemIndex   newItem;
};
```

After sending the `kMenuFindItemMsg`, the Menu Manager will send the `kMenuHiliteItemMsg` to hilite the newly selected item. The `MenuRef`, `menuRect`, and `hitPt` parameters are as usual. The `whichItem` parameter is actually a pointer to a `HiliteMenuItemData` structure. The MDEF should unhilite the previously selected item, if non-zero, and hilite the newly selected item.

The `mDrawMsg` is slightly modified. Previously, the `whichItem` parameter was unused. It will now point to a read/write `MenuTrackingData` structure. The MDEF may update the fields of the data structure if it wants.

This new API will also be added:

```
OSStatus GetMenuTrackingData( MenuRef menu, MenuTrackingData* outData );
```

`GetMenuTrackingData` returns information about the menu currently being tracked. It can be used by applications and MDEFs to replace direct access to lowmem. The `menu` parameter may be `NULL` to get info about the deepest menu in the tracking hierarchy, or a `MenuRef` for any menu that's currently open. On output, all fields are initialized except for the `itemRect` field. That field is only for use by the MDEF while handling `kFindItemMsg`.

The `InitProcMenu` API is not implemented on Carbon, which ends application customizability of the MBDF.

Carbon MDEF Tips

- MDEFs must be compiled as PowerPC code into your application or library.
- Use the new API `RegisterMenuDefinition` to register a mapping from MDEF ID to defproc function pointer.

- MDEFs should ignore any unknown messages. This has always been the case for writing defprocs, but not all defprocs have followed the rules. Please do.
- Modify code that implements `mDrawMsg` to get information about the menu virtual top and bottom from the `MenuTrackingData` parameter instead of from low memory.
- Modify code that implements `mChooseMsg` so that it implements `kMenuFindItemMsg` and `kMenuHiliteItemMsg` instead.
- Modify code that reads or sets the low memory globals `TopMenuItem`, `AtMenuBottom`, `MenuDisable`, and `MBSaveLoc` to use the new `MenuTrackingData` structure instead. This structure is passed to `mDrawMsg` and `kMenuFindItemMsg`, and may be read at any time using the new API `GetMenuTrackingData`.
- MDEFs have previously needed to modify the `mbUglyScroll` field of the `MBSaveLoc` low memory global (at offset 16) in order to support correct scrolling in combination with submenus. CarbonLib now sets this global automatically. However, if your MDEF implements scrolling, it is still necessary to avoid scrolling if your menu is not the frontmost displayed menu. The easiest way to determine this is to call `GetMenuTrackingData(NULL, &trackingData)` and compare `trackingData.menu` with the menu passed to your MDEF. If they are the same, the menu passed to the MDEF may be safely scrolled. If they are different, your MDEF should not scroll.
- Optionally, support the new messages `kMenuInitMsg` and `kMenuDisposeMsg`.

Working With Menu and Menu Item Attributes

The `GetMenuAttributes`, `ChangeMenuAttributes`, `GetMenuItemAttributes`, and `ChangeMenuItemAttributes` APIs add attributes for the menu as a whole and for each menu item:

```
enum
{
    kMenuAttrExcludesMarkColumn,
    kMenuAttrAutoDisable
};
typedef OptionBits MenuAttributes;
```

```
OSStatus ChangeMenuAttributes( MenuRef menu,
      MenuAttributes attributesToSet,
      MenuAttributes attributesToClear );

OSStatus GetMenuAttributes( MenuRef menu,
      MenuAttributes* attributes );

enum
{
   kMenuItemAttrSubmenuParentChoosable
};
typedef OptionBits MenuItemAttributes;

OSStatus ChangeMenuItemAttributes( MenuRef menu,
      MenuItemIndex item,
      MenuItemAttributes attributesToSet,
      MenuItemAttributes attributesToClear );

OSStatus GetMenuItemAttributes( MenuRef menu,
      MenuItemIndex item,
      MenuItemAttributes* attributes );
```

In general, menu and menu item attributes are things that you would set statically in a resource file. If an aspect of a menu will change dynamically many times at runtime, its probably not an attribute.

The following functions are similar to Mac OS 8.5 APIs in the Control Manager and Window Manager.

```
OSStatus GetMenuItemPropertyAttributes( MenuRef menu,
      MenuItemIndex item,
      OSType  propertyCreator,
      OSType  propertyTag,
      UInt32* attributes );



OSStatus ChangeMenuItemPropertyAttributes( MenuRef menu,
      MenuItemIndex item,
      OSType  propertyCreator,
```

```
    OSType  propertyTag,
    UInt32  attributesToSet,
    UInt32   attributesToClear );
```

These will be provided in CarbonLib. However, CarbonLib will not support all of the attributes that Carbon/X does. Attribute status:

Attribute: On 8.1?/On 8.5?

`kMenuAttrExcludesMarkColumn`: no/yes

`kMenuAttrAutoDisable`: yes/yes

`kMenuItemAttrSubmenuParentChoosable`: no/yes

Enabling and Disabling Menu Items

These APIs supply a more general way of getting information about the enable state of the menu items.

```
// menu attributes
enum
{
   kMenuAttrAutoDisable // menu is auto-disabled when all items are
};

void    DisableAllMenuItems( MenuRef theMenu );

void    EnableAllMenuItems( MenuRef theMenu );

Boolean MenuHasEnabledItems( MenuRef theMenu );
```