Inside the Mercutio MDEF

The **Mercutio MDEF** is a Menu DEFinition resource that allow developers to easily and elegantly extend the power of their application menus. Mercutio allow menus to have multiple-modifier key-equivalents (e.g. shift-command-C), custom icons, item callbacks, and other goodies. The Mercutio MDEF works under System 6.0.4 or later, with or without Color QuickDraw.

The Mercutio MDEF Package

The Mercutio MDEF package contains the following:

- **Mercutio MDEF**: A fully functional version of the MDEF which contains the resources needed to incorporate Mercutio into your applications.
- **Mercutio Demo App**: a demonstration application that allows you to see Mercutio in action and try out its features.
- **Sample Code*f***: Sample code in THINK Pascal and Metrowerks CodeWarrior C demonstrating how to use the MDEF, including the source code for the Mercutio DemoApp.
- **Inside Mercutio**: This document, which explains the features of Mercutio in detail.

Conventions Used in this Manual

Inside the Mercutio MDEF uses various conventions to present information. Words that require special treatment appear in specific fonts or font styles. Certain information, such as parameter blocks, use special formats so that you can scan them quickly.

Special Fonts

All code listings, reserved words, and the names of actual data structures, constants, fields, parameters, and routines are shown in Courier (`this is Courier`).

Types of Notes

There are several types of notes used in this book.

Note

A note like this contains information that is interesting but possibly not essential to an understanding of the main text. (An example appears on page 3-19). ◆

IMPORTANT

A note like this contains information that is essential for an understanding of the main text. (An example appears on page 3-19.) ▲

▲ **WARNING**

Warnings like this indicate potential severe problems that you should be aware of as you design your application. Failure to heed these warnings could result in system crashes and loss of data. (An example appears on page 4-28). ▲

Technical Support

Digital Alchemy is a small design and consulting firm based in Palo Alto, California. We specialize in graphic design, custom software development, and exotic human interfaces.

Other shareware products available from Digital Alchemy include:

- CIconButton CDEF: a simple “icon button” CDEF that allows you to use arbitrarily-sized color icon buttons in your applications.
- VersionEdit: a drag-and-drop ‘Vers’ resource editor.

All technical support for Mercutio is provided through electronic mail. For fastest response, please use the Internet e-mail address provided below.

U.S.Mail Digital Alchemy
c/o Ramon M. Felciano
P.O.Box 9632
Stanford, CA 94309-9632

Internet felciano@camis.stanford.edu

Applelink ALCHEMY

Please feel free to contact us with comments, feature requests, or (gasp!) bug reports.

Quickstart!

For those of you that want to get started right away, here are the five basic steps to integrate Mercutio into your application:

1. Copy the resources in the "Mercutio 1.2.MDEF" file into your application's resource file. There are 3 resources that are needed: the "Mercutio" `MDEF` resource, and the ".MDEF Font" `NFNT` and `FOND` resources.
2. Change the `MDEF` field in your `MENU` resources to 19999, the resource ID of the Mercutio MDEF. *Don't change this resource number.*
3. If you are programming in Pascal, add the "Mercutio API.p" file to your project. If you are programming in C, add the "Mercutio API.c" and "Mercutio API.h" files to your project.
4. Replace any calls to the toolbox `MenuKey` routine with the with new `MDEF_MenuKey` call. This replacement is needed in order to parse the additional modifier keys. Most likely you will have an event loop routine that looks something like this:

```
CASE event.what OF
   keyDown, AutoKey:
      BEGIN
      theKey := BitAnd(Event.message, charCodeMask);
      IF (BitAnd(Event.modifiers, CmdKey) = CmdKey) THEN BEGIN
         menuResult := menuKey(theKey);
         if HiWord(menuResult) <> 0 THEN BEGIN
            ProcessMenu(menuResult);
            HiliteMenu(0);
            END;
         END;
      ELSE BEGIN { cmd not down; handle typing if needed }
         ...
```

which should be changed to look like this:

```
CASE event.what OF
   mousedown:
      ...
   keyDown, AutoKey:
      BEGIN
      theKey := BitAnd(Event.message, charCodeMask);
      menuResult := MDEF_MenuKey(Event.message,
                                 Event.modifiers, hAnMDEFMenu);
      if HiWord(menuResult) <> 0 THEN BEGIN
```



```
      ProcessMenu(menuResult);
      HiliteMenu(0);
      END;
   ELSE BEGIN { wasn't caught by the menus }
      ...
```

Note
The third parameter to the `MDEF_MenuKey` routine must be a handle to a menu that is using the Mercutio MDEF. ◆

5. Recompile your application.

That's it! Run your program. Any menu item in condense style with show an option key modifier in addition to the command key; any menu item in extend style with show an shift key modifier in addition to the command key. If you want to use additional modifier keys or Mercutio features, read on…

Basic Architecture

Mercutio provides a number of features that redefine the appearance and behavior of Macintosh menus. Mercutio uses a technique called **style-bit remapping** to control these features on a menu item by menu item basis. The features that are available through style-bit remapping for a given menu are controlled **feature templates**.

Style-bit remapping

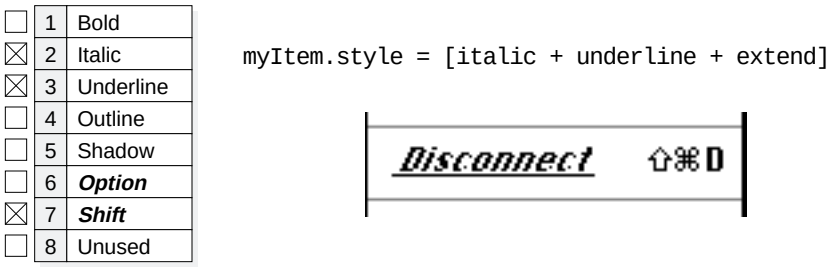
Menu items will typically differ in which Mercutio features they use: one will have a command-shift key equivalent, the next has command-option, etc. Mercutio uses the style field the standard item record to specify which features are used by a menu item. Recall that the style field is a sequence of 8 bit flags, each of which turns a particular text style on or off. The Macintosh toolbox provides a set of constants to manipulate this field (`normal = 0, bold = 1, italic = 2, underline = 4`, etc.)

Figure 2-1 Sample menu item with text styles

The 8 bit codes of the style field are normally used as **style flags** to indicate the text style of the menu item. Mercutio interprets certain style bits as **feature flags** instead of style flags. This is called **style-bit remapping**, because a style bit is linked, or remapped, to control something other than a text style. For example, by default, Mercutio interprets

the Condense style bit as a flag for the Option key, and the Extend style bit as a flag for the Shift key. Thus an item that is formatted using the Extend text style will have a Shift modifier key instead of a wider text style (Figure 2-2).

Figure 2-2 Sample menu item with Mercutio's style-bit remapping



When a style bit is used to indicate whether or not a particular Mercutio feature is to be used, that bit is called a **feature flag**. Mercutio's feature flags let you control the following six item-specific features:

- The four **modifier keys**: command, shift, option, and control. See "Extended key equivalents" on page 13.
- Whether or not an item is a **dynamic item**. See "Dynamic items" on page 17.
- Whether or not an item is a **callback item**. See "Item Callbacks" on page 19.

Feature Templates

There are several potential problems with the style-bit remapping scheme:

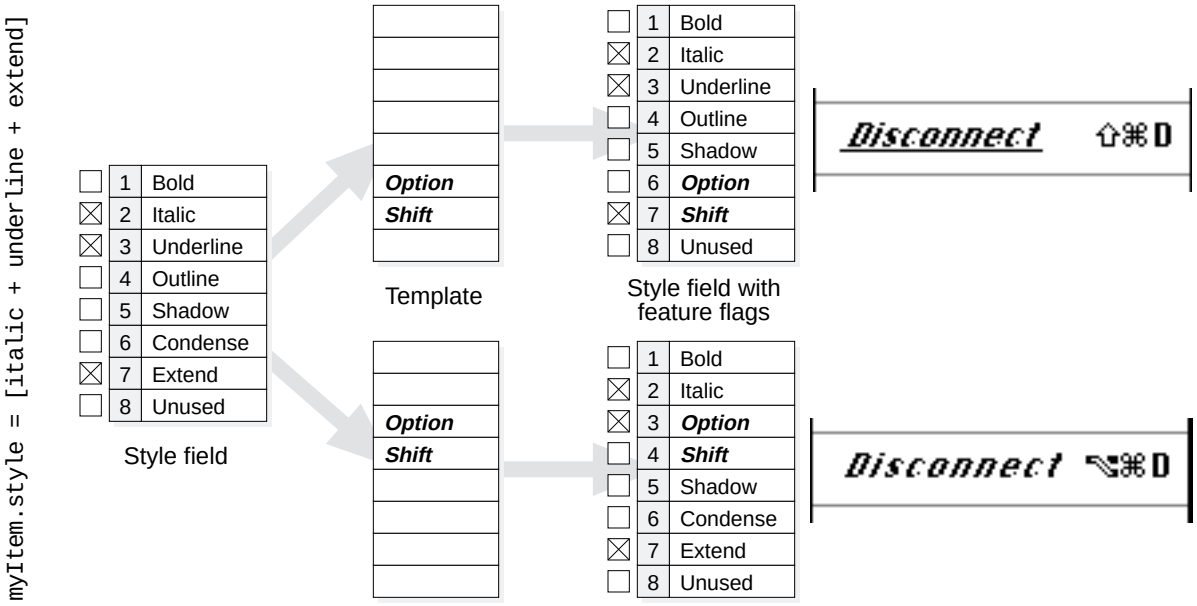
- What if you want an item to be drawn in extend format?
- What if you want the item to be drawn in extend format and have the shift-key as a modifier for the key equivalent?
- If there are six features to flag, does that mean there are only 2 text styles left to use?

Mercutio addresses these problems by letting you select, on a menu-by-menu basis, which Mercutio style bits to use as feature flags and which ones to use as style flags. For example, if one menu uses the Condense and Extend text styles for several menu items, but doesn't use Outline or Shadow styles, you can use Outline and Shadow as feature flags and restore the Extend and Condense bits to their normal function as style flags. This is done through **feature templates** that tell Mercutio which style bits to map to which features.

Every menu that uses Mercutio has its own mapping template. Thus, one menu might use the Extend style to flag the Shift modifier, whereas the next uses the Italics style to flag Shift but uses the Extend style to flag the Control modifier. Figure 2-1 shows how

the same set of style bits can be interpreted differently depending on what feature template is used.

Figure 2-1 Feature selection using a preference template



Note
You do not need to allocate a style bit to every Mercutio feature; use only those features you need. In the examples in Figure 2-1, only one modifier key is supported in each example (the Shift-key in the first example, the Option-key in the second one). ◆

You can set the feature template for your menus programmatically using the `MenuPrefsRec` data structure and the `MDEF_SetMenuPrefs` call. Listing 0-1 shows how to set the feature flags for a menu using the first template in Figure 2-1.

Listing 0-1 Setting the feature flags for a menu

```
PROCEDURE SetMenuPrefs(theMenu : MenuHandle);
VAR
   myPrefs:       MenuPrefsRec;
BEGIN
   WITH myPrefs DO BEGIN
```

```
      isDynamicFlag := [];
      forceNewGroupFlag := [];
      useCallbackFlag := [];
      controlKeyFlag := [];
      optionKeyFlag: [condense];
      shiftKeyFlag: [extended];
      cmdKeyFlag := [];
      requiredModifiers := cmdKey;
   END;
   MDEF_SetMenuPrefs(theMenu, @myPrefs);
END;
```

IMPORTANT

For compatibility reasons, the default preferences are set to make Mercutio behave the same way it did in version 1.1. In particular, the Condense style bit flags the Option key, the Extend style bit flags the Shift key, and the Command key is the default modifier. ▲

‘Xmnu’ resource

Instead of setting the features for your menus programmatically every time your application launches, you can store the feature templates for your menus in Mercutio’s `Xmnu` resources. As with the `MenuPrefsRec` record, the `Xmnu` resource stores settings for individual menus, but not for menu items.

During menu initialization, Mercutio looks for an `Xmnu` resource with the same resource ID as the menu’s `menuID`. If no `Xmnu` resource with a matching ID is found, Mercutio looks for an `Xmnu` resource with ID 0. If no Xmnu resources are found, Mercutio uses the default settings. Thus, you can set a default for all your application menus by including an `Xmnu` resource with ID 0 in your resource fork.

`TMPL` resources for ResEdit and Resorcerer are included with Mercutio to help you fill out these `Xmnu` resources.

Data Structures

This section describes the `MenuPrefsRec` record in detail as well as the `MenuResPrefs` record, which defines the format of the `Xmnu` resource.

MenuPrefsRec

The `MenuPrefsRec` record is the data structure that represents the feature template. Every Mercutio feature is represented as a style field in the record; by setting this style you indicate which style bit will be used to flag that feature. The only exception is the `requiredModifiers` field, which is of type `integer`.

```
TYPE  MenuPrefsPtr = ^MenuPrefsRec;
      MenuPrefsRec = RECORD
         isDynamicFlag: style;
         forceNewGroupFlag: style;
         useCallbackFlag: style;
         controlKeyFlag: style;
         optionKeyFlag: style;
         shiftKeyFlag: style;
         cmdKeyFlag: style;
         requiredModifiers: integer;
      END;
```

Field descriptions

`isDynamicFlag` Specifies which style bit will be used to flag whether the menu item is part of a group of menu items defining a dynamic item.

`forceNewGroupFlag` Specifies which style bit will be used to flag whether the item starts a new dynamic item.

`useCallbackFlag` Specifies which style bit will be used to flag whether the MDEF should call the callback procedure before drawing the item.

`controlKeyFlag` Specifies which style bit will be used to flag the Control modifier key.

`optionKeyFlag` Specifies which style bit will be used to flag the Option modifier key.

`shiftKeyFlag` Specifies which style bit will be used to flag the Shift modifier key.

`cmdKeyFlag` Specifies which style bit will be used to flag the Command modifier key.

`requiredModifiers` Specifies which modifiers will be used by default.

DESCRIPTION

The `controlKeyFlag`, `optionKeyFlag`, `shiftKeyFlag`, and `cmdKeyFlag` fields set style-bits used to flag the corresponding modifier keys. For example, if the `controlKeyFlag` field in the `MenuPrefsRec` is set to `[bold]`, any menu items with the bold style-bit set will be drawn with an option key equivalent (but not drawn in boldface.)

You should only use single styles to control these features. For example, don't set `controlKeyFlag` to `[bold, italic, underline]`.

If you don't want to use a particular feature for a given menu, set the style field for that feature to `[]`.

The field `requiredModifiers` can be used enforce consistency across menu item key equivalents. For example, if `requiredModifers` is set to `cmdKey + optionKey`, *every*

menu item with a key equivalent will require at least the Option- and Command-keys to be held down (additional modifiers may be required if the appropriate style bits are set).

MenuResPrefs

The `MenuResPrefs` record is very similar to the `MenuPrefsRec` except that it stores a version number as well. This is the format used to store data in the `Xmnu` resource.

```
TYPE  MenuResPrefs =
      RECORD
         version: integer;
         thePrefs: MenuPrefsRec;
      END;

MenuResPrefsPtr = ^MenuResPrefs;
MenuResPrefsHandle = ^MenuResPrefsPtr;
```

Field descriptions

`version`	Version number for the `Xmnu` resource data structure.
`thePrefs`	A Mercutio menu preference record.

DESCRIPTION

You should rarely need to use this data structure, since you can edit the `Xmnu` resources directly using ResEdit or Resorcerer and one of the supplied `TMPL` resources.

The current version number for `Xmnu` resources is 1.

Using Mercutio

The following is a comprehensive list of the features supported by Mercutio and instructions on how to use them in your software. Except as noted below, the Mercutio MDEF behaves identically to the System 7 MDEF.

Extended Icon Support

Mercutio supports a wide variety of icon types and sizes, including color icons (`'cicn'`) and System 7 icon suites (`'icl8', 'ICN#', etc.`).

Figure 3-1 Icon support in Mercutio

When determining which icon resource to use to display an icon, the Mercutio MDEF follows a particular search order to find out exactly which icon is going to be displayed (Table 3-2).

Table 3-2 Mercutio icon search order

System	Desired Size	Search Order
7	32 x 32	System 7 Icon Suites 'cicn' 'ICON' 'ICN#'
7	16 x 16	System 7 Icon Suites 'cicn' (shrunk) 'ICON' (shrunk) 'ICN#' (shrunk)
6.0.4	32 x 32	'cicn' 'icl8' 'ICON' 'ICN#'
6.0.4	16 x 16	'cicn' (shrunk) 'icl8' (shrunk) 'SICN' 'ics#' 'ICON' (shrunk) 'ICN#' (shrunk)
6.0.4, B/W	32 x 32	'ICON' 'ICN#'
6.0.4, B/W	16 x 16	'SICN' 'ics#' 'ICON' (shrunk) 'ICN#' (shrunk)

There is limited system software support for icon suites under System 6. However, Mercutio tries to use whatever icon resources are available. If a `'icl8'` is found for a given menu item, it will be converted to a `'CICN'` and used as the item's icon. Similarly, if an `'ICON'` resource is not found, Mercutio will look for a `'ICN#'` resource with the same ID. This means you can develop one set of icons that will work for both System 6.0.4 and System 7.

Support for small icons

The System MDEF lets you include small icons (16 by 16 pixels, usually stored as `sicn` resources) into menu items by putting `$1E` into the `cmdChar` field of the menu item record. Unfortunately, this doesn't allow you to use small icons in menu items with key equivalents or hierarchical submenus, since they also use the `cmdChar` field. To address this issue, Mercutio draws all icons with IDs of 500 and above as small icons.

Extended key equivalents

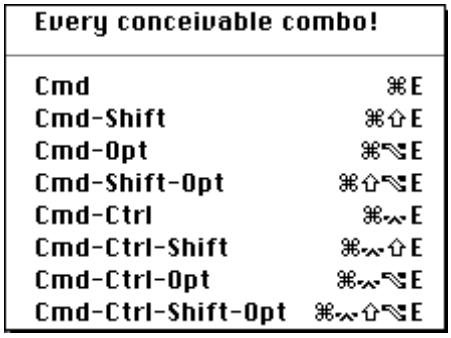
Mercutio supports an extended set of key equivalents for menu items. This includes additional **modifier keys** as well as the **non-printing keys** on the keyboard (e.g. function keys).

Support for all modifier keys

The Mercutio MDEF supports Option, Shift and Control as modifiers in addition to the Command-key.

Caps-lock is not supported.

Figure 3-1



MDEF_MenuKey

In order to take advantage of Mercutio's features, you must use the `MDEF_MenuKey` routine instead of the standard Menu Manager `MenuKey` routine. The `MDEF_MenuKey` routine checks against the various combinations of modifier keys that Mercutio allows; the `MenuKey` routine only checks for the Command key.

```
FUNCTION MDEF_MenuKey (theMessage: longint; theModifiers: integer;
hMenu: menuHandle): longint;
```

The `theMessage` and `theModifiers` parameters can be taken directly from an event record. The `hMenu` parameter must by a handle to a Mercutio menu; the `MDEF_MenuKey` routine uses this handle to get at Mercutio's private data. See "MDEF_MenuKey" on page 29.

Modifier-defaults

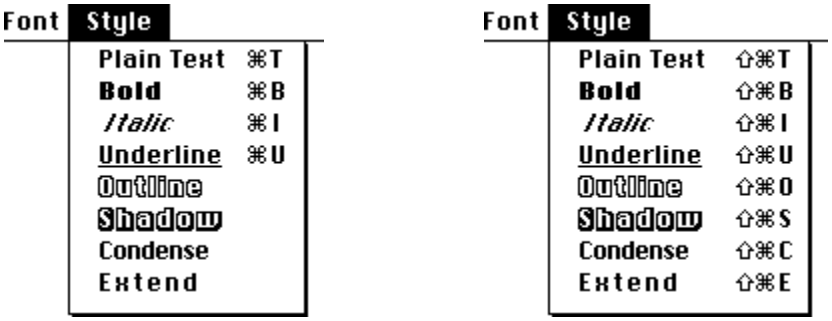
By default, all key equivalent combinations include the command key. That is, if you don't set any additional feature flags but do fill in the `cmdChar` field for the menu item, Mercutio assumes you require the Command key to be down for the key equivalent to trigger. The `defaultModifiers` field in the `MenuPrefs` record lets you select which modifier keys to use as the default modifiers.

Using modifier-defaults to add key equivalents to a Style menu

The modifier-defaults feature turns out to be particularly useful for Style menus. The Macintosh Human Interface Guidelines suggest you use the styles to indicate the effects of choosing an item from the style menu (See "Style menu normal and with Command-Shift defaults" on page 14.) Since Mercutio uses the style bits to flag features, it would seem that certain items can't be drawn in their styles if we want the Style menu to include additional modifiers.

In particular, at least one style can't be drawn as a text style because it's being used as a feature flag. For example, if we use the Italic bit to flag the Shift-key, the third item in the left menu in Figure 3-1 wouldn't be drawn italics.

Figure 3-1 Style menu normal and with Command-Shift defaults



Conventional Style Menu **Desired Style Menu**

You can use the `defaultModifiers` field to address this problem. Listing 0-2 shows how to set the default modifiers to be Command-Shift and clear the feature flags (by setting the style fields in the `MenuPrefs` record to `[]`). Using these settings, any menu item with data in its `cmdChar` field will assume that the Command- and Shift-keys to be held down for the key equivalent to trigger (the right menu in Figure 3-1).

Listing 0-2 Setting the feature flags for a menu

```
PROCEDURE SetStyleMenuPrefs(theStyleMenu : MenuHandle);
VAR
   myPrefs:       MenuPrefsRec;
BEGIN
   WITH myPrefs DO BEGIN
      isDynamicFlag := [];
      forceNewGroupFlag := [];
      useCallbackFlag := [];
      controlKeyFlag := [];
      optionKeyFlag: [];
      shiftKeyFlag: [];
      cmdKeyFlag := [];
      requiredModifiers := cmdKey + shiftKey;
   END;
   MDEF_SetMenuPrefs(theStyleMenu, @myPrefs);
END;
```

Note

This `defaultModifiers` feature was added to Mercutio as a direct result of user requests for a method of using additional modifiers for Style menu items. Bear in mind that the Apple Human Interface Guidelines have specific recommendations for key equivalents in the Style menu (as shown in the image above); if you use other key equivalents, you run the risk of interrupting the continuity of the user experience. ◆

Support for non-printing key-equivalents

The Mercutio MDEF supports non-printing keys such as the function keys, page up, page down, arrow keys, tab and delete. Note that certain non-printing keys have symbolic representations in the menu (e.g. the arrow keys are represented by iconic arrows) and others are spelled out in full-text (Figure 3-1).

Figure 3-1 Example of non-printing characters as key equivalents

The Mercutio MDEF interprets *lowercase characters* in the menu item's `cmdChar` field as these non-printing keys. For example, a lowercase A (`'a'`) in the `cmdChar` field will appear as Enter in the menu. Table 4-1 describes how the ASCII lowercase characters are mapped to new values in order to support non-printing characters.

Note

Note that the arrow keys are the only non-printing keys that don't have lower-case equivalents (we ran out with only 26 lowercase letters to choose from). You'll need to enter these values ($80-$83) by hand using ResEdit or another resource editor. ◆

Table 3-2 Mercutio ASCII character mapping

Char	ASCII	Becomes	Key-Code
a	97 ($61)	Enter	$4C
b	98 ($62)	Return	$24
c	99 ($63)	Tab	$30
d	100 ($64)	Num Lock	$47

Table 3-2 Mercutio ASCII character mapping

Char	ASCII	Becomes	Key-Code
e	101 ($65)	F1	$7A
f	102 ($66)	F2	$78
g	103 ($67)	F3	$63
h	104 ($68)	F4	$76
i	105 ($69)	F5	$60
j	106 ($6A)	F6	$61
k	107 ($6B)	F7	$62
l	108 ($6C)	F8	$64
m	109 ($6D)	F9	$65
n	110 ($6E)	F10	$6D
o	111 ($6F)	F11	$67
p	112 ($70)	F12	$6F
q	113 ($71)	F13	$69
r	114 ($72)	F14	$6B
s	115 ($73)	F15	$71
t	116 ($74)	Help	$72
u	117 ($75)	Del	$33
v	118 ($76)	Forward Del	$75
w	119 ($77)	Home	$73
x	120 ($78)	End	$77
y	121 ($79)	Page Up	$74
z	122 ($7A)	Page Down	$79
	128 ($80)	Up Arrow	$7E
	129 ($81)	Down Arrow	$7D
	130 ($82)	Left Arrow	$7B
	131 ($83)	Right Arrow	$7C

Use ‘.MDEF Font’ for menu symbols

All keyboard symbols (modifiers and key equivalent characters) are stored in a custom font called `.MDEF Font`. To use the Mercutio MDEF, you will need to copy this font information (a NFNT/FOND resource pair) into the resource fork of your application.

Do not rename or renumber this font.

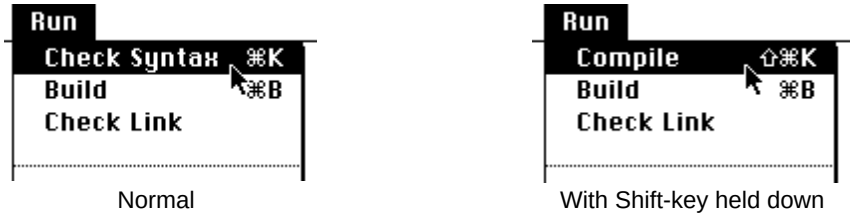
Dynamic items

Dynamic menu items are items that change appearance and behavior depending on what modifier keys are being held down. This is useful for closely-related commands, rarely-used commands, power-user commands, or other situations where you want to provide functionality without cluttering your menus. Several commercial applications use this kind of menu. For example, the THINK Pascal® “Run” menu changes its appearance if the shift-key is held down (Figure 3-1)

Figure 3-1 Menu in THINK Pascal®

You can accomplish the same thing using Mercutio (Figure 3-2). The main difference is that Mercutio will also display the modifier keys being held down.

Figure 3-2 Menu with Mercutio



Mercutio does this by grouping sets of menu items, called **item alternates**, that occupy the same location in the menu—the actual item from this set that is displayed depends on the modifier keys held down by the user. Obviously, not every menu item will have alternates, so not all items will change when modifier keys are held down. For example, the “Build” command in Figure 3-2 above doesn’t change.

Note
For a dynamic menu item to switch to its alternate, the user must hold down the alternate's modifier combination exactly. In Figure 3-2, if the user holds down Option- and Shift-, the menu would display "Check Syntax", not "Compile". ◆

Using Dynamic Items in Mercutio

You establish a set of item alternates by grouping them sequentially in the `MENU` resource. Mercutio considers a sequence of menu items as a group if they:

1. Have the `isDynamic` flag set.
2. Share the same key equivalent.

The first item in the set is the one that is initially displayed when the menu appears; the subsequent ones are alternates that will appear if their particular combination of modifiers is held down.

You may want to have two dynamic items next to each other that share the same key equivalent character. You can force a separation between two groups of items that share the same key equivalent character with the `ForceNewGroup` flag.

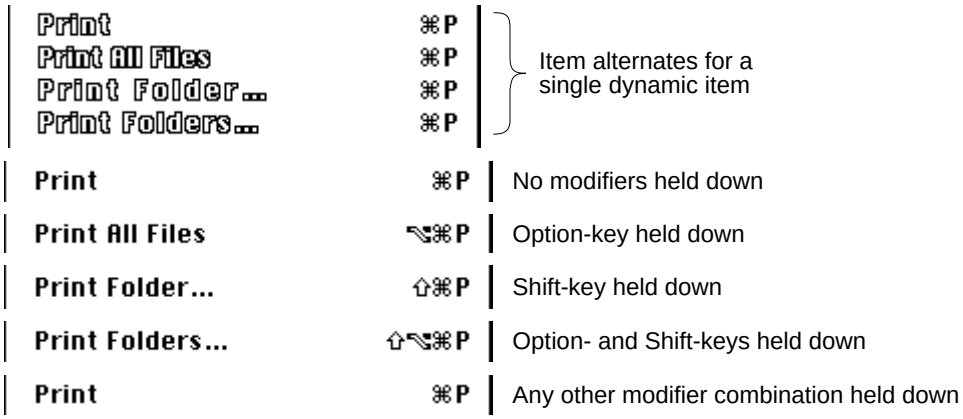
Listing 0-3 shows how to set the feature template for a menu that uses the Condense style bit to flag the Option key, the Extend style bit to flag the Shift key, and the Outline style bit to flag dynamic items (i.e. the default behavior plus support for dynamic items).

Listing 0-3 Setting the feature flags for a menu

```
PROCEDURE SetMyMenuPrefs(theStyleMenu : MenuHandle);
VAR
   myPrefs:       MenuPrefsRec;
BEGIN
   WITH myPrefs DO BEGIN
      isDynamicFlag := [outline];
      forceNewGroupFlag := [];
      useCallbackFlag := [];
      controlKeyFlag := [];
      optionKeyFlag: [condense];
      shiftKeyFlag: [extended];
      cmdKeyFlag := [];
      requiredModifiers := cmdKey;
   END;
   MDEF_SetMenuPrefs(theStyleMenu, @myPrefs);
END;
```

Using this feature template, any sequence of menu items with their Outline bits set and with the same key equivalent will be considered item alternates for a single dynamic item. Figure 3-1 shows a “Print” menu item with four item alternates that will toggle depending on what menu items are held down.

Figure 3-1 Dynamic “Print” menu item with 4 alternates



A more complete example of how to design and build Dynamic Items for your application is shown in Figure 3-2 on page 3-21.

Note

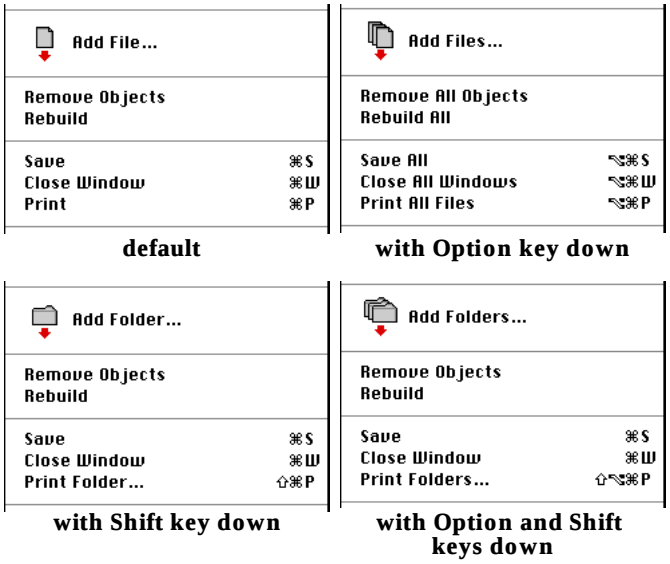
Mercutio searches sequentially through all the alternates for a given item, and selects the first match; if there are several alternates that have the same modifier sequences and key equivalent, only the first one will be available to the user. ◆

Figure 3-2 Scenario demonstrating Dynamic Items

We want to create a menu that supports three key equivalents (the default Command-, plus Option- and Shift-), and uses Dynamic items. *

First we decide how we'd like the menu to look and behave. In our example, we try to enforce a simple interface rule when appropriate: the Option-key toggles between single and multiple objects (e.g. “Add File…” vs. “Add Files…”).

Now that we know what items will be in the menu, and where and when they will appear, we need to decide how to indicate flag the different features on each menu item. In particular, we decide which bits in the menu item's style field will be used as feature flags.

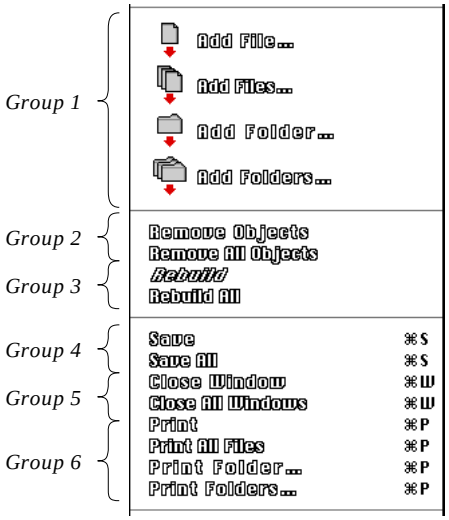


1	Bold	
2	***newGroup***	***newGroup***
3	Underline	
4	***Dynamic***	***Dynamic***
5	Shadow	
6	***Option***	***Option***
7	***Shift***	***Shift***
8	Unused	

We decide we want to use the Italic, Outline, Condense and Extend styles to control the various feature settings. This gives us a style field and feature template as represented on the left. We can now assign these preference settings to the menu programmatically using `MDEF_SetMenuPrefs`, or via an `'Xmnu'` resource.

In this example, we will set the preferences programmatically. We fill out a `MenuPrefsRec` record to represent our feature template, and call `MDEF_SetMenuPrefs`. Note that we set the `requiredModifiers` field as well (that is, in our menu, the default modifier key for key equivalents is Command-).

```
WITH prefs DO BEGIN
    optionKeyFlag := [condense];
    shiftKeyFlag := [extend];
    cmdKeyFlag := [];
    controlKeyFlag := [];
    isDynamicFlag := [outline];
    forceNewGroupFlag := [italic];
    useCallbackFlag := [];
    requiredModifiers := cmdKey;
END;
MDEF_SetMenuPrefs(hMenu, @prefs);
```



Finally, we build our menu using our favorite resource editor. Note that all the items are outlined (since they all belong to one of the groups of dynamic items), and that, with one exception, we don’t need to indicate where one group ends and another starts, since Mercutio implicitly uses a change of key equivalent character as an end-of-group marker.

← explicit end-of-group (italic style)

← implicit end-of-group (separator, not in outline style)

← implicit end-of-group (new key equiv.)

← implicit end-of-group (new key equiv.)

* This example is based on the sample code in the Mercutio package; a notable difference is that the sample code also uses the Underline style to flag Callback items.

Callback items

Callback items allow you to determine the contents of a menu item at runtime. This is for situations where you don't know the contents of the item ahead of time. For example, a "File Type" menu might include icons from applications that the user has on the hard drive; you could use dynamic items to pull the icons out of the Desktop Database at runtime.

Mercutio allows you to associate a **callback procedure** with a Mercutio menu, and flag certain items as callback items. Before drawing any callback item, Mercutio will load in the item data (item text, icon handle, item modifiers, enabled state, etc.) from the `MENU` resource, then call the callback procedure in your application to give you the opportunity to modify the data before the item is displayed.

Note
Style bits are interpreted before your callback procedure is called. Since the item text is drawn in whatever style is returned from the callback procedure, your callback procedure can set the item's text style safely without affecting the modifier keys or other features. ✥

The Callback Procedure

The callback procedure is called for each menu item whose `useCallback` flag is set. The callback procedure receives a record with the item's data, as well as a message field which indicates what fields the procedure may change. The callback procedure has the following header:

```
PROCEDURE MyCallbackProc (menuID: integer; previousModifiers:
integer; VAR itemData: RichItemData);
```

The `previousModifiers` field is supplied in case you want to compare the current modifiers against those held down the last time this item was referenced.

A callback procedure gets called several times for each menu item because Mercutio requires different information at different times. The `cbMsg` field indicates what fields Mercutio wants filled in. There are three values it can take:

```
cbBasicDataOnlyMsg = 1;
cbIconOnlyMsg = 2;
cbGetLongestItemMsg = 3;
```

Depending on the value of cbMsg, different fields are filled in and available for changing by the callback procedure (See "RichItemData" on page 21.)

Listing 0-4 shows the structure of a typical callback procedure.

Listing 0-4 Sample callback procedure

```
PROCEDURE MyCallbackProc (menuID: integer;
                          previousModifiers: integer;
                          VAR itemData: RichItemData);
BEGIN
   itemData.changedByCallback := false;
   CASE itemData.itemID OF

      3 :
         BEGIN
            CASE itemData.cbMsg OF
               cbBasicDataOnlyMsg : BEGIN
                  … fill in data for item 3
                  END;
               cbIconOnlyMsg : BEGIN
                  … fill in icon data for item 3
                  END;
               cbGetLongestItemMsg:  BEGIN
               … fill in longest data for item 3
               END;
            itemData.changedByCallback := true;
         END;

      7 :
         BEGIN
            CASE itemData.cbMsg OF
               cbBasicDataOnlyMsg : BEGIN
                  … fill in data for item 7
                  END;
                  cbIconOnlyMsg : BEGIN
               … fill in icon data for item 7
                  END;
               cbGetLongestItemMsg:  BEGIN
                  … fill in longest data for item 7
                  END;
               END;
            itemData.changedByCallback := true;
         END;

      … et al for other callback menu items …
      END; { CASE itemData.itemID }
END;
```

RichItemData

This is record stores information about the contents, behavior, and visual appearance of a single menu item. The first four fields are the same as the four-byte header for each menu item in a MENU resource.

```
TYPE  richItemData = PACKED RECORD
         iconID: Byte;
         keyEq: char;
         mark: char;
         textStyle: Style;
         itemID: integer;
         itemRect: rect;
         flags: itemFlagsRec;
         iconType: ResType;
         hIcon: Handle;
         pString: stringPtr;
         itemStr: str255;
         cbMsg: integer;
      END;
      richItemPtr = ^richItemData;
```

Field descriptions

iconID	Resource ID of the item's icon.
keyEq	ASCII value of the key equivalent.
mark	ASCII value of the mark symbol.
textStyle	Font face to use when displaying the menu item.
itemID	Position of the item in the menu.
flags	Mercutio feature flags (interpreted from the item style).
iconType	4-character resource type for the icon.
hIcon	Handle to the icon data.
pString	Pointer to the item string.
itemStr	Storage for the item string.

DESCRIPTION

If you supply a handle to a new menu icon in hIcon, be sure to set the iconType field as well.

The iconType field indicates to what type of icon data hIcon points. Usually this is one of the standard resource types ('ICON','sicn', 'cicn'). It can also be 'suit' if you use System 7 icon suites.

The pString field points to the menu item's text string. By default, it points to the beginning of the itemStr field which holds the item text from the menuHandle. You can change it to point to another string, or change the itemStr field directly.

The cbMsg field takes one of 3 values:

- `cbBasicDataOnlyMsg = 1`: fill in the non-icon data fields only.
- `cbIconOnlyMsg = 2`: fill in the icon data fields only.
- `cbGetLongestItemMsg = 3`: fill in the longest possible menu item. Mercutio uses this item to determine the maximum width of the menu. You should determine what the longest item would be (i.e. longest item text, biggest icon, most modifiers, etc.), and return it.

The following sections explain what fields are available to you for changing.

- □ →: Mercutio fills this field with data, but you should not change it.
- □ ↔: Mercutio fills this field with data; you may change it.
- □ ←: Mercutio needs this field; you should fill it in.
- □ ⊗: This field contains invalid data; you should not change it.

Table 3-1 explains what fields are available to you for changing depending on the value of `cbMsg`.

Table 3-1 Field permissions for callback messages in `cbMsg`

Field	cbBasicDataOnlyMsg	cbIconOnlyMsg	cbGetLongestItemMsg
iconID	↔	↔	↔
keyEq	↔	⊗	↔
mark	↔	⊗	↔
textStyle	↔	⊗	↔
itemID	→	→	→
itemRect	→	→	→
flags	↔	⊗	↔
iconType	⊗	↔	↔
hIcon	⊗	←	↔
pString	↔	⊗	↔
itemStr	↔	⊗	↔
cbMsg	→	→	→

If your callback procedure does change any of the fields, you must set the `changedByCallback` flag in the `itemFlagsRec`.

IMPORTANT

Mercutio relies on the setting of the `changedByCallback` flag to determine whether any information has changed. If the flag is not set, Mercutio will ignore any changes you made to the record. ◆

ItemFlagsRec

This is a record structure used to flag the features of a given menu item. It is derived by the MDEF from the style field and the MenuPrefsRec as described above.

```
TYPE  ItemFlagsPtr = ^ItemFlagsRec;
      ItemFlagsRec = PACKED RECORD
         { high byte }
         forceNewGroup: boolean;
         isDynamic: boolean;
         useCallback: boolean;
         controlKey: boolean;
         optionKey: boolean;
         unused10: boolean;
         shiftKey: boolean;
         cmdKey: boolean;

         { low byte }
         isHier: boolean;
         changedByCallback: boolean;
         enabled: boolean;
         hilited: boolean;
         smallIcon: boolean;
         hasIcon: boolean;
         unused1: boolean;
         unused0: boolean;
      END;
```

Field descriptions

`forceNewGroup`	A Boolean value indicating whether or not the menu item forces the start of a new group of item alternates for a dynamic item. Mercutio ignores this field if the `isDynamic` field is not set to `TRUE`.
`isDynamic`	A Boolean value indicating whether or not the menu item is part of a group of item alternates for a dynamic item.
`useCallback`	A Boolean value indicating whether or not the menu item will call a callback procedure before being drawn. Mercutio ignores this field if the menu has no callback procedure associated with it (See “MDEF_SetCallbackProc” on page 28.)
`controlKey`	A Boolean value indicating whether or not the menu item uses the Control key as a modifier for the key equivalent. Ignored if the `keyEq` field of the `RichItemData` record is empty.
`optionKey`	A Boolean value indicating whether or not the menu item uses the Option key as a modifier for the key equivalent. Ignored if the `keyEq` field of the `RichItemData` record is empty.
`unused10`	Reserved for future use.

`shiftKey`	A Boolean value indicating whether or not the menu item uses the Shift key as a modifier for the key equivalent. Ignored if the `keyEq` field of the `RichItemData` record is empty.
`cmdKey`	A Boolean value indicating whether or not the menu item uses the Command key as a modifier for the key equivalent. Ignored if the `keyEq` field of the `RichItemData` record is empty.
`isHier`	A Boolean value indicating whether or not the menu item is a hierarchical menu item. Mercutio sets this to true if the data from the `MENU` resource shows a value of `hMenuCmd` (`$1B`) in the item's `keyEq` field.
`changedByCallback`	A Boolean value indicating whether or not any of the fields in this record or in the `RichItemData` record have been changed by the callback procedure.
`enabled`	A Boolean value indicating whether the menu item will be drawn enabled or disabled.
`hilited`	A Boolean value indicating whether the menu item will be drawn hilited or not.
`smallIcon`	A Boolean value indicating whether or not the menu item is a hierarchical menu item. Mercutio sets this to true if the data from the `MENU` resource shows a value of (`$1E`) in the item's `keyEq` field, or if the item's `icon` field shows a value of 500 or greater (See "Support for small icons" on page 12.)
`hasIcon`	A Boolean value indicating whether or not the menu item has an icon associated with it. Mercutio sets this to true if the data from the `MENU` resource shows a non-zero value in the item's `icon` field.
`unused1`	Reserved for future use.
`unused0`	Reserved for future use.

DESCRIPTION

This set of flags control most of Mercutio's features, including whether or not an item is a callback item or not.

The MDEF Messages

MDEFs are code resources that function by responding to a series of messages sent by the Menu Manager. There are 7 messages that all System 7 compatible MDEFs should recognize; Mercutio recognizes these 7 plus several others needed to control the various features and settings in Mercutio menus.

MDEFs are procedures with the following header:

Listing 0-5 Menu definition procedure header

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

The parameters are interpreted differently depending on the value of the `message` parameter. Table 4-1 describes all of the messages understood by the Mercutio MDEF.

Table 4-1 Mercutio MDEF Messages (italics indicate non-standard messages)

Message	Call
areYouCustomMsg	Used to determine whether the MDEF is one of Digital Alchemy's custom MDEFs or not. If so, `menuRect.topLeft` will contain the 4 character resType `'CUST'`.
getVersionMsg	Returns the MDEF version number in `menuRect.topLeft` (typecast to a `longint`)
getCopyrightMsg	Returns a `StringHandle` to the MDEF copyright information in `menuRect.topLeft`.
setCallbackMsg	Set the callback procedure for the MDEF to the procedure pointed to in `hitPt`.
stripCustomDataMsg	Dispose any custom data structures setup by the MDEF.

Table 4-1 Mercutio MDEF Messages (italics indicate non-standard messages)

Message	Call
setPrefsMsg	Set menu preferences (feature template) to the `MenuPrefsRec` record pointed to in `hitPt`.
mMenuKeyMsg	Given a keyboard event, determine whether it maps to a key equivalent in one of the currently installed menus (both custom and standard menus are checked).
mDrawItemStateMsg	Draw menu item `itemID` in rectangle `menuRect`. If `hitPt.h` = 1, the item is drawn in hilited state. If `hitPt.v` = 0, the item is drawn disabled.
mDrawMsg	Draw the entire menu within `menuRect`
mChooseMsg	Unhilite menu item `itemID`, hilite the item under point `hitPt`, and return the ID of that item in `itemID`.
mSizeMsg	Calculate the size of the menu and store values in the `menuHeight` and `menuWidth` fields of `theMenu`
mPopUpMsg	Calculate the rectangle in which the popup should appear, and return it in `menuRect`.
mDrawItemMsg	Draw menu item `itemID` in rectangle `menuRect`.
mCalcItemMsg	Calculate the rectangle of menu item `itemID`. The `top` and `left` fields of `menuRect` should be filled in; the MDEF will calculate the `right` and `bottom` fields

Note

Table 4-1 is provided for informational purposes only; in order to encourage compatibility with future versions of Mercutio, you should use the wrapper routines provided in the C and Pascal API files, described in Chapter 5, “API Routines.” ◆

Mercutio API Routines

The following routines are provided in the C and Pascal application programming interface (API). At the very least, you will need to use the MDEF_MenuKey routine to trap key equivalents using the Mercutio MDEF.

MDEF_MenuKey

The `MDEF_MenuKey` function finds the menu and item associated with a keypress.

```
FUNCTION MDEF_MenuKey (theMessage: longint; theModifiers: integer;
hMenu: menuHandle): longint;
```

`theMessage`	The message field from an Event record
`theModifiers`	The modifiers field from an Event record
`hMenu`	A handle to a menu that uses the Mercutio MDEF.

DESCRIPTION

`MDEF_MenuKey` is a replacement for the standard toolbox call `MenuKey` for use with the Mercutio MDEF. Given the keypress message and modifiers parameters from a standard event record, it checks to see if the keypress is a key-equivalent for a particular menu item.

If you are currently using custom menus (i.e. menus using a Mercutio MDEF), pass the handle to one of these menus in hMenu. If you are not using custom menus, pass in NIL or another menu, and `MDEF_MenuKey` will use the standard `MenuKey` function to interpret the keypress.

As with `MenuKey`, `MDEF_MenuKey` returns the menu ID in high word of the result, and the menu item in the low word.

MDEF_SetCallbackProc

The MDEF_SetCallbackProc procedure sets the callback procedure.

```
PROCEDURE MDEF_SetCallbackProc (menu: MenuHandle; theProc:
procPtr);
```

menu	A handle to the specified menu record.
theProc	A pointer to the callback procedure.

DESCRIPTION

The MDEF_SetCallbackProc procedure sets the procedure that will be called for each item before it is drawn. The procedure is also called during the SizeMenu message call. The callback routine should have the following header:

```
PROCEDURE MyGetItemInfo (menuID: integer;
                         previousModifiers: integer;
                         VAR itemData: RichItemData);
```

menuID	The ID of the menu that is being referenced. Note: the itemData record contains the ID of the item itself.
previousModifier	The state of the modifiers the last time the callback procedure was called. You may want to use this to set the dirty flag (see below). Only the high-byte of this parameter is valid.
itemData	A parameter block data structures. The validity and read-write state of the fields is defined by the cbMsg field as described below.

For more information on how to use callback procedures with Mercutio, see “Callback items” on page 3-19, or check the sample code that came with Mercutio.

MDEF_CalcItemSize

The MDEF_CalcItemSize procedure calculates and returns the coordinates of the rectangle in which the menu item will be displayed.

```
PROCEDURE MDEF_CalcItemSize (menu: MenuHandle; item: integer;
VAR theRect: rect);
```

menu	A handle to the specified menu record.
item	The number of the menu item.
theRect	Holds the top and left coordinates of the desired rectangle. When the procedure returns, the bottom and right coordinates are filled in.

DESCRIPTION

`MDEF_CalcItemSize` will calculate the height and width for a given menu item. It assumes that `theRect.top` and `theRect.left` of `theRect` are filled in; Mercutio will fill in `theRect.bottom` and `theRect.right`.

MDEF_DrawItem

The `MDEF_DrawItem` procedure draws a menu item in the specified location.

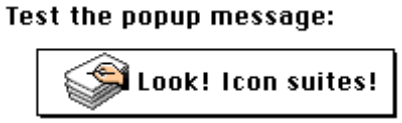
```
PROCEDURE MDEF_DrawItem (menu: MenuHandle; item: integer;
destRect: rect);
```

`menu`	A handle to the specified menu record.
`item`	The number of the menu item.
`destRect`	The rectangle where the item should be drawn.

DESCRIPTION

The `MDEF_DrawItem` procedure will draw a given item in the rectangle you specify. This is useful for drawing popup menus that should show the current menu item as it would be drawn by Mercutio (Figure 4-1). The sample code provided with Mercutio demonstrates how to do this.

Figure 5-1 Popup menu drawn with `MDEF_DrawItem`



MDEF_DrawItemState

The `MDEF_DrawItemState` procedure draws a menu item at a specified location and in the specified enabled and hilited states.

```
PROCEDURE MDEF_DrawItemState (menu: MenuHandle; item: integer;
destRect: rect; hilited, enabled: boolean);
```

`menu`	A handle to the specified menu record.
`item`	The number of the menu item.
`destRect`	The rectangle where the item should be drawn.
`hilited`	Whether the item should be drawn hilited or not.

`enabled` Whether the item should be drawn enabled or not.

DESCRIPTION

The `MDEF_DrawItemState` procedure allows you to override the menu item's `hilited` and `enabled` settings.

MDEF_StripCustomMenuData

The `MDEF_StripCustomMenuData` procedure will remove any custom data allocated by the Mercutio MDEF for the specified menu.

```
PROCEDURE MDEF_StripCustomMenuData (menu: MenuHandle);
```

`menu` A handle to the specified menu record.

DESCRIPTION

`MDEF_StripCustomMenuData` will remove any custom data allocated by the Mercutio MDEF. Use this before writing a Mercutio menu to disk as a `MENU` resource.

▲ **WARNING**

If you write your `MENU` resource to disk *before* calling `MDEF_StripCustomMenuData`, it may not initialize correctly the next time you load the menu and have Mercutio display it. ▲

MDEF_SetMenuPrefs

The `MDEF_SetMenuPrefs` procedure sets the feature preferences for the specified menu.

```
PROCEDURE MDEF_SetMenuPrefs (menu: MenuHandle; pPrefs:
MenuPrefsPtr);
```

`menu` A handle to the specified menu record.
`pPrefs` A pointer to the Mercutio preferences data structure.

DESCRIPTION

The `MDEF_SetMenuPrefs` procedure lets you determine which style bits for the menu items will be interpreted as feature flags for the MDEF.

The `pPrefs` pointer should point to a `MenuPrefs` record which holds the settings for the current menu. Mercutio makes a copy of these preferences and stores them internally, so the data structure may be disposed of after the call to `MDEF_SetMenuPrefs` returns.

You can call `MDEF_SetMenuPrefs` repeatedly to change the preferences of a menu.

If the preferences you are setting can impact the height or width of the menu (e.g. allowing new modifier keys which could make certain menu items wider), you must call `CalcMenuSize` on the menu after the `MDEF_SetMenuPrefs` call.

MDEF_IsCustomDef

The `MDEF_IsCustomDef` function returns `TRUE` if the menu is being controlled by a Digital Alchemy MDEF.

```
FUNCTION MDEF_IsCustomDef (menu: MenuHandle): boolean;
```

`menu` A handle to the specified menu record.

DESCRIPTION

This function checks an "author code" embedded in the MDEF.

Note

`MDEF_IsCustomDef` will return `false` if another MDEF is controlling the menu. ◆

MDEF_GetCopyright

The `MDEF_GetCopyright` function returns a Pascal string containing the copyright notice for Mercutio.

```
FUNCTION MDEF_GetCopyright (menu: MenuHandle): str255;
```

`menu` A handle to the specified menu record.

DESCRIPTION

The string returned by `MDEF_GetCopyright` is the appropriate notice to use in manuals and About boxes. For more information on when to use this, see "Licensing and Distribution" on page 4-33.

MDEF_GetVersion

The `MDEF_GetVersion` function returns a version number for Mercutio.

```
FUNCTION MDEF_GetVersion (menu: MenuHandle): longint;
```

`menu` A handle to the specified menu record.

DESCRIPTION

The version number returned by `MDEF_GetVersion` is in the same format as the “short version” information as normally stored in a `'vers'` resource.

Compatibility

Mercutio attempts to be compatible with the System 7 MDEF default behavior whenever possible. This chapter summarizes the differences in appearance and behavior between Mercutio and the System 7 MDEF (aside from the obvious additional features that Mercutio provides.)

Saving MENU resources

Mercutio stores custom data at the end of the menu's handle. This data must be initialized at application startup. Because of this, it is important that you don't save your MENU resources back to your application's resource fork. If you do, the next time you launch the program, Mercutio will use the old outdated data.

Note that any such UI changes that need to persist from one session to the next should be stored in the Preferences folder, not to the application itself. This is particularly important in order to let your application run off of CD-ROMs and other locked volumes.

Hierarchical menu glitch

When moving back and forth between two adjacent hierarchical menus, the hierarchical menu sometimes appears near the top of the parent menu item, and sometimes near the bottom. According to Apple's Developer Technical Support, this is a bug in the Menu Manager that the system MDEF hacks to get around. Because this is a relatively minor cosmetic glitch, we decided against making illegal forays into the internal Menu Manager structures in order to fix it.

Better scroll positioning

The system MDEF occasionally leaves a gap at the bottom of a scrolling menu instead of extending to the bottom of the screen; Mercutio doesn't do this. This is probably a side-effect of some assumptions they make for speed optimizations.

MenuKey compatibility

The standard toolbox `MenuKey` routine ignores the shift key; `MDEF_MenuKey` doesn't. That is, if you hit shift-command-T and there is only a command-T item, `MDEF_MenuKey` returns 0, whereas `MenuKey` would have returned the command-T item.

The standard `MenuKey` routine allows you to select (via command keys) from a hierarchical menu item whose parent menu item or parent menu is disabled. This is a bug that `MDEF_MenuKey` fixes.

Font information for popup menus

Using the System MDEF, you can get popup menus to appear in different fonts and sizes by changing the text-related fields of the `grafPort` just before calling `PopupMenuSelect`. Mercutio does not support this behavior, primarily because there is no way for the MDEF to tell whether it is being called from a popup menu or from a standard pull-down menu in the menubar. It also isn't a very elegant solution to the problem of having menus in different fonts and sizes.

We hope to address this issue in future a version of Mercutio.

Licensing and Distribution

This chapter describes the terms under which you may use Mercutio in your applications or distribute the Mercutio package. We've tried to come up with a licensing scheme that is fair and allows shareware and freeware developers to use Mercutio without paying a licensing fee.

Overview

Normally, to use Mercutio in your application, you license it from Digital Alchemy and pay a licensing fee. A more attractive alternative is our **Poor Man's License**, which allows you to use Mercutio provide you credit us in your About box and manual, and send us a copy of the software.

If you are interested in licensing Mercutio for fee, or according to other terms, contact Digital Alchemy directly.

Use and Distribution of the Mercutio MDEF

Usage of the Mercutio MDEF is subject to a software license and licensing fee. The license lets you use and distribute the Mercutio MDEF in a single Macintosh application. You agree not distribute it except as part of the licensed application.

You must include the following text in your manual or on-line documentation:

> **Mercutio MDEF from Digital Alchemy**
> **Copyright © Ramon M. Felciano 1992-1995, All Rights Reserved**

All licenses are non-transferable, single-application licenses, and include free upgrades to future versions of Mercutio. Contact Digital Alchemy for details about licensing fees.

Licensing fees and terms are subject to change without notice. All technical support will be provided via e-mail.

Poor Man's License

The licensing fee is waived through the Poor Man's License. Under the Poor Man's License, you can use Mercutio for free in your application if you do the following:

1. Include the following text in your About box:

   Mercutio MDEF by Ramon M. Felciano, © 1992-1995, All Rights Reserved

2. Include the following in your User manual or on-line documentation:

   Mercutio MDEF from Digital Alchemy
   Copyright © Ramon M. Felciano 1992-1995, All Rights Reserved

3. Send us a copy of your product (including free upgrades to any future versions that still use the MDEF). The address is provided in the front of this manual. If the software includes full on-line documentation, you can send it via e-mail.

If you are developing shareware or freeware that will be distributed across the Internet, you don't need to explicitly send me a copy. Just notify me when the software is released, tell me where I can find it, and include me in your database of registered users.

As mentioned above, other licensing terms are available—contact Digital Alchemy for details.

Use and Distribution of the Mercutio MDEF Package

The Mercutio MDEF Distribution Package contains a fully functional version of the MDEF. You may copy, share or give the package to whomever you wish provide you always distribute the package in its entirety and no modifications are made to its contents; you may not sell, trade it, or otherwise charge for it.

The Mercutio MDEF Distribution Package may be included as part of a CD-ROM or other collection of developer or on-line materials provided you notify Digital Alchemy of this fact.

Legal Stuff

For those of you that don't want to read the boring legal version, here's the gist of what follows (paraphrased from the EasyFlow license & disclaimer as quoted in Wired 2.01):

> **If Mercutio doesn't work, tough. If you lose millions because Mercutio messes up, it's you that's out millions, not us. If you don't like this disclaimer: tough. We reserve the right to do the absolute minimum provided by law, up to and including nothing. This is basically the same disclaimer that comes with all software packages, but this one is in plain English and the official one is in legalese. I didn't want to include any disclaimer at all, but our lawsuit-happy society makes it a requirement.**

The following sections essentially contain the same thing, but in painful legalese.

Disclaimer of Warranty

In using this software, you understand and agree that this software is provided “as is” without warranty of any kind. The entire risk as to the results and performance of using this software lies entirely with you, the user. DIGITAL ALCHEMY AND RAMON M. FELCIANO MAKE NO WARRANTIES, EXPRESS OR IMPLIED, INCLUDING WITHOUT LIMITATION THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE, REGARDING THE SOFTWARE OR ITS USE AND OPERATION ALONE OR IN COMBINATION WITH LICENSEE PROGRAM. Neither Licensee, its employees, agents, or Distributors have any right to make any other representation, warranty or promise with respect to the Software.

Limitation of Liability

In no event shall Digital Alchemy or Ramon M. Felciano be liable for any consequential, indirect, incidental, or special damages whatsoever (including without limitation damages for loss of critical data, loss of profits, interruption of business, and the like) arising out of the use or inability to use, distribution, or licensing of this software by licensee or any third party, whether under theory of contract, tort (including negligence), product liability or otherwise. Because some states do not allow the exclusion or limitation of liability for consequential or incidental damages, the above limitations may not apply to you.

Although the author would appreciate any feedback and bug reports, the author shall not be responsible for correcting any problems which you discover or otherwise help you maintain and use this software. Furthermore, the author may at any time replace, modify, alter, improve, enhance or change this software.

Troubleshooting

This chapter will help you solve any problems you might run into or answer question you may have about Mercutio. Feel free to send us e-mail if you have any other questions.

Basic checklist

If you are having trouble getting Mercutio to work, the following checklist may help narrow down the problem.

- You must have the ".MDEF Font" `FONT` and `NFNT` resources installed in your resource fork.
- The Mercutio MDEF must be installed in the resource fork of your application. It's resource ID must be 19999.
- You must assign tell your menus to use the Mercutio MDEF rather than the System MDEF by setting the `MDEF` field of the `MENU` resource to 19999.
- If you want to use any Mercutio features other than the Shift- and Option- key support, you must set your preferences with an `Xmnu` resource or programmatically using the `MDEF_SetMenuPrefs` call.
- `MDEF_SetMenuPrefs` affects a single menu, so you must issue a separate call for every menu.
- To be safe, call `CalcMenuSize` after every call to `MDEF_SetMenuPrefs`. This isn't required, but it will make sure that Mercutio keeps the height and width of menus correct.
- You must replace every call to `MenuKey` with a call to `MDEF_MenuKey`.
- The last parameter to `MDEF_MenuKey` *must* be a handle to a menu that uses Mercutio.

Common symptoms and remedies

If you run into difficulties getting up and running with Mercutio, these tips may help you overcome them.

Some of my key equivalents have become page up/down, function keys, etc.!

- Mercutio maps lowercase key equivalent characters to non-printing keys such as page up / down (See “Support for non-printing key-equivalents” on page 15.) If you installed Mercutio and are getting these key equivalents unexpectedly, make sure the characters in the `keyEq` field of the menu items are uppercase, not lowercase.

The width/height of my menu is all screwed up!

- Make sure you call `CalcMenuSize` after you set the preferences for a given menu.

My callback routine isn’t being called!

- Check the `MenuPrefsRec` or `Xmnu` resource for that menu to make sure that the `useCallbackFlag` field has a style associated with it.
- Make sure the menu item(s) in question have that style bit set.
- Make sure you’ve called `MDEF_SetCallbackProc` to link your procedure to that menu. Remember that you need to do this for every menu that uses that procedure.

My menu item isn’t appearing in the correct text style.

- If that style is missing, you are probably using it as a feature flag. Check the `MenuPrefsRec` or `Xmnu` resource for that menu.

All I get is the Shift- and Option- modifiers.

- Remember that the only features that work “out of the box” are Shift- and Option-. If you want your menus to support the Control-key, dynamic items, or item callbacks, you must set those preferences with `MDEF_SetMenuPrefs`. See “MenuPrefsRec” on page 8.

My menu items have extra modifier keys.

- You have some extra feature flags set. Check the `MenuPrefsRec` or `Xmnu` resource for that menu, as wells as the style bit settings for the menu items in question.

How can I have different key equivalent characters in the alternates for a given dynamic item?

- You can’t.

The modifier keys show up correctly, but the menu doesn't respond to keypresses

- Make sure you are calling `MDEF_MenuKey`, not the Toolbox `MenuKey` routine.
- Make sure the last parameter to `MDEF_MenuKey` is a menuHandle for a menu that uses Mercutio.

ResEdit complains about illegal fields in the Xmnu TMPL resource.

- You're probably using the TMPL resource designed for Resorcerer. Copy a fresh on from the "Xmnu Template for Resedit" file.

My icons are being drawn smaller than normal.

- Check the resource ID of the icons in question. Remember that Mercutio draws any icon with a resource ID between 500 and 512 as a small icon—See "Support for small icons" on page 12.

My popup menu isn't appearing in Geneva 9 point anymore

- This is a limitation of the current version of Mercutio. See "Font information for popup menus" on page 36.

Frequently Asked Questions

If you have other questions about Mercutio, these may answer them for you. If not, please feel free to contact Digital Alchemy directly.

Our product includes three applications that use Mercutio. How many licenses do we need?

Three—a single license covers a single Macintosh application. However, to take advantage of the Poor Man's License, you only need to send one copy of the product, assuming it includes all three applications.

The credit information should be in all three About boxes and user manuals.

Does the license include upgrades to Mercutio?

Yes. As long as you abide by the terms of the license, you can use subsequent versions of Mercutio.

What about upgrades to my product?

If you are taking advantage of the Poor Man's License, you should send me a copy whenever you release a product upgrade.

What text needs to be displayed in the About box?

Use `MDEF_GetCopyright` to get the correct copyright notice.

Why is this thing called “Mercutio” anyway?

When I first started programming Mercutio, I was taking a Shakespeare class at Stanford. Part of the class included playing out selected scenes from the plays we were reading; I played Mercutio, Romeo’s hothead brother, and was pretty immersed in the character at the time.

In other words, it seemed like a good idea at the time. :)

What was the “Shakespeare MDEF Collection” and what happened to it?

Mercutio 1.0 was first released in early 1992. Many developers requested features such as additional modifier keys and support for non alphanumeric key-equivalents. Including all of the features in a single MDEF would have used up all of the style bits; the Shakespeare MDEF Collection was to address these requests by providing a suite of MDEFs with a variety of feature combinations. We subsequently came upon the idea of storing the style-bit–to–feature mapping external to the MDEF, which allowed us to provide all the features and still let developers choose which style-bits to give up. This obviated the need for a collection of MDEFs.

The Shakespeare MDEF Collection may still make an appearance sometime in the future once it becomes unrealistic to add more features to Mercutio. Until then, however, we’ll leave it to the Halls of Vaporware fame!

Glossary

Callback item A menu item that has its `useCallbackFlag` set so that Mercutio calls a callback procedure before displaying the item. See also **callback procedure**.

Callback procedure An application procedure that can perform modifications on menu item data before the menu item is displayed. Mercutio calls this routine for items that have their `useCallbackFlag` set. See also **callback item**.

Dynamic item An item that changes appearance and behavior depending on what modifier keys are being held down. A dynamic item is controlled by a set of item alternates, each of which represents a different appearance and behavior for the item. For example, a dynamic item might be controlled by two alternates: the default one, and one that shows up if the user holds down the Option-key. See also **item alternates**.

Feature flag A style bit used to turn a Mercutio feature on or off for a given menu item. This is the way Mercutio determines which modifier keys a menu item uses, whether it is a dynamic item, etc.

Feature template A record that determines which style bits in the menu item's style field will be used as style flags and which ones will be used as feature flags. See also **feature flag**, **style flag**.

Item alternates A set of menu items, grouped sequentially in `MENU` resource, that can appear at the same location in a menu depending on what modifiers are being held down. See also **dynamic items**.

Mercutio menu A menu controlled by the Mercutio MDEF.

Modifier keys The Shift, Option, Command, Control, and Caps Lock keys. For the purposes of use with Mercutio, the Caps Lock key is ignored.

Non-printing keys Keys on the Macintosh keyboard that don't typically have ASCII representations and don't appear in printed documents. Examples are the function keys, arrow keys, page up and page down.

Poor Man's License The no-fee license through which you can use Mercutio by including certain copyright credits and sending us a copy of your program.

Style-bit remapping The mechanism by which Mercutio interprets certain style bits as feature flags instead of style flags. See also **feature flag**, **style flag**.

Style flag A style bit used to turn a text style on or off for a given menu item. This is the way style bits are normally used by the toolbox and the System MDEF.