F4. Menu Handling

Menu item events are returned from selections in menus that are either auto-installed at launch time, dynamically added by your program (typically using the utility command SetItm), or are added via menu controls in ViewIt windows. The type of event (if any) that is returned when an item is selected is determined by the menu item type.

Menu Item Types

Menu item types are differentiated on the basis of whether they are labeled or not, and, if labeled, whether they are one of 3 standard item types. This produces 5 distinct types:

1. Unlabeled Program Items ("Do It")
2. Labeled Program Items ("Do It#121")
3. Program-Wide Standard Items ("Hide#108")
4. Shared Standard Items ("Cut#13")
5. Module-Specific Standard Items ("Combine#1575")

where a "labeled" item is one that contains "#n" in its item text in the MENU resource (n is an integer), and "standard" items are ones that have both their appearance and behavior automatically controlled by a FaceWare module.

Most of the standard items described below are already present in our demo program MENU resources to illustrate their operation. ResEdit (or other resource editor) can be used to modify these menus and/or add new menus containing any mixture of the five menu item types.

• Unlabeled Program Items

All menu items that do not contain the "#" character in their titles (are not labeled) return simple menu events to the main program when selected:

uMenuID = menu ID of selected menu
uMenuItem = selected menu item number
uString = selected menu item text
uResult = 0

The main program is responsible for maintaining both the appearance and behavior of such items. This is the most common type of new menu item added to program menus since these items give the user access to program-specific functionality.

• Labeled Program Items

Menu items with labels in the range of #121 to #1000 also return control with menu events when selected, but return with uResult equal to the label ID:

uMenuID = menu ID of selected menu
uMenuItem = selected menu item number
uString = selected menu item text
uResult = label ID (121-1000)

A menu item with the title "Do It#121", for example, is processed by ViewIt when the menu is loaded. ViewIt removes the "#121" and stores this information in a private record that can later be used to identify and manipulate the item based on its associated label ID. (See the "Menu Utilities" topic for further information.)

The advantages of using such labeled program items versus unlabeled items are that (1) code can be written to respond to the value of uResult, making that code independent of the position of the item in MENU resources, (2) more than one item with the same label ID can be present in multiple MENU resources, and (3) the utility command SetItm can be used to manipulate all instances of the item at once.

NOTE: There is a slight dependence between memory use and the magnitude of the label ID, so it pays to use label IDs beginning with #121.

• Program-Wide Standard Items

Standard menu items are labeled items whose appearance and behavior are handled by a FaceWare module (i.e., menu events are not returned by such items). FaceIt supports several "program-wide" standard items that have label IDs between 101 and 120:

About#101 - (see description below)
Delete...#104 - supports deleting a file

Transfer...#105 - supports transferring to another app
Quit#106 - quits program
Select#107 - first item in list of open windows
Hide#108 - hides active window
Send Behind#109 - sends active window behind next
Send to Back#110 - sends active window to back
Hide Others#111 - hides all windows except active
Show All#112 - shows all hidden windows

These program-wide standard items can be added to any menu in the main menu bar, but should not be used in menu controls in ViewIt windows.

The Select standard item defines the position in a menu where FaceIt will build a windows list. Unlike the other standard items, it can only appear in one place, and must be the last item in its parent menu (since FaceIt will clobber all items below it when building the windows list).

The About standard item is reset to contain the name of the program file, "About [ProgramName]...", when DoInit is executed. Selection of this item returns a labeled menu event to the program (with uString = "About") so that the program can display information about itself. This item differs from other standard items in that it is the only one without built-in functionality.

• Shared Standard Items

Another class of standard menu items are those that are shared by different modules as the current context changes (i.e., as the identity of the active window or selected control is changed). These shared standard items have label IDs between 1 and 100:

Open...#2 - typically opens a file
Close#4 - typically closes or hides a window†
Save#5 - typically saves something to disk
Save As...#6 - typically saves something with new name
Save Special#7 - typically another way to save
Revert#8 - typically reverts something from disk
Page Setup...#9 - typically opens Page Setup dialog
Print...#10 - typically prints window/control contents
Print Special#11 - typically another way to print
Undo#12 - typically undoes last action
Cut#13 - typically cuts selection (copy + clear)
Copy#14 - typically copies selection to clipboard
Copy Special#15 - typically another way to copy
Paste#16 - typically pastes clipboard to window/control
Paste Special#17 - typically another way to paste
Clear#18 - typically clears selection
Select All#19 - typically selects all in window/control
Find...#20 - typically supports searching for something
Next Case#21 - typically finds next case of something
Go To...#22 - typically jumps to designated place

†see "Closing Windows" in "Windows" topic in ViewIt guide

A HelpCt editable control and a BaseCt editable control in the same ViewIt window, for example, "share" the Copy item in the sense that they take turns controlling this item as the user selects one control or the other.

Which of the shared standard menu items are supported by a module is defined by the content of an STR# resource that has the same ID number as the baseID of the module. BaseCt (the basic ViewIt control driver), for example, includes an STR# 1310 resource with the following strings:

1. [empty]
...
12. [empty]
13. Cut
14. Copy
15. [empty]
16. Paste

17. [empty]
18. Clear
19. Select All

which informs FaceIt that BaseCt only supports the Cut, Copy, Paste, Clear, and Select All standard items. This STR# list also defines the default menu text that FaceIt is to set these items to when a BaseCt editable control becomes the selected control in a ViewIt window (meaning that you would need to translate these strings as well as the MENU resources if converting the program to another language).

Any number of instances of the same shared standard item can be put in any number of menus of any type (i.e., you can have the standard Copy item in more than one menu). All such instances of the same standard item will have the same appearance and behavior.

• Module-Specific Standard Items

These items are standard labeled items that are supported by specific window- or control-driving modules. The label number will be equal to the baseID of the module + n where n is greater than zero. GrafCt (baseID 1570), for example, supports labels #1571, #1572, etc. These standard items are disabled when the current program context is not being managed by the associated module. Otherwise the module will control both the appearance and behavior of the item. The documentation accompanying each module will describe any support the module may have for such standard items.

Font/Size/Style/Color Items

When DoInit is called, ViewIt initializes Font, Size, Style, & Color (FSSC) menus from MENU resources 1216-1219 that have menu IDs 196-199, respectively. These menus are loaded by ViewIt as non-main menus and can be attached to hierarchical menu items in any other menu. The items within these menus are similar to standard items since both the appearance and behavior of the items in FSSC menus is automatically handled by FaceWare modules.

The style menu in this help window, for example, contains hierarchical menu items that are attached to ViewIt's FSSC menus. In general, you should provide access to the FSSC menus whenever editable controls are used that have the "Supports FSSC" option checked in the Control dialog.

Managing Menu Items

The utility commands GetItm (get menu item info) and SetItm (set menu item info) can be used to get and set menu item characteristics. These commands replace a large number of Menu Manager toolbox calls and have the added advantage of recognizing label IDs. To disable all instances of the menu item with label ID #125, for example, you can simply write,

```
FaceIt(nil,SetItm,0,125,2,0);
```

SetItm can also be used to dynamically add/delete entire menus, automatically processing any label IDs found in such menus. See "Menu Utilities" topic for a complete description of these important commands.

Custom Standard Items (Advanced)

The behavior and appearance of standard items is handled automatically by the module associated with the current program context. The standard items in this window, for example, are being controlled by the HelpCt control driver.

In some cases you may need to modify the behavior of an existing standard item. Several options are available.

• Negative Label IDs

One way to modify the behavior of a standard item is by giving the item a negative label ID in the MENU resource. This causes FaceIt to return control with a menu item event when the item is chosen (instead of executing its default behavior). Changing "Quit#106" to "Quit#-106", for example, causes FaceIt to return control with a menu item event when the Quit item is chosen:

uMenuID = menu ID of menu containing Quit
uMenuItem = item number of the Quit item
uString = item text of the Quit item (usually "Quit")
uResult = label ID = 106

If desired, the standard item can then be executed by passing its label ID as a command (as described under "Commands" in the "Program Commands" topic):

```
FaceIt(nil,106,0,0,0,0); execute Quit
```

This gives the program a chance to do additional processing before and/or after executing the standard item, or to simply replace the standard response with its own.

Note that the use of a negative label ID has no affect on the item's appearance (i.e., its title and enabled state) which is still controlled by the current program context (i.e., by the currently selected control). If you need to control both the item's behavior and its appearance, then simply make the item a program menu item.

• Changing Label IDs

Another way of modifying the behavior of a standard item is to change its label ID as a function of the current program context. You might, for example, be interested in using the "Select All" standard item to select the cells in a ListCt control under some circumstances but want the standard behavior in others. The following call would change all instances of label ID #19 (= standard "Select All") to #125 (a labeled program item), and thereby give control of the item to your program:

```
FaceIt(nil,SetItm,0,19,11,125);
```

Your program would then respond to selection of this item by selecting all of the cells in the list. To later regain the standard item behavior, simply change label ID #125 back to #19:

```
FaceIt(nil,SetItm,0,125,11,19);
```

• New Standard Items

In other cases you may be interested in adding support for standard items that are not directly supported by a module. Suppose, for example, that you wished to add support for a standard Print item to print the text in BaseCt's editable text controls. The first step is to make a copy of BaseCt's STR# 1310 and modify it to include a "Print" item:

```
   #        old STR# --> new STR#
 ...        ...              ...
  9        [empty]          [empty]
 10        [empty]           Print
 11        [empty]          [empty]
 12        [empty]          [empty]
 13          Cut              Cut
 ...        ...              ...
```

where this modified copy of STR#1310 is best placed in the program's resource file so that it does not affect programs that are sharing the FaceWare file.

The presence of the modified STR# will cause FaceIt to enable the Print standard item when a BaseCt editable text control is selected. Selection of the Print item will then send a menu event to the BaseCt driver which, since it does not know how to handle this item, will then post it back to the main program as a program menu event:

uMenuID = menu ID of menu containing Print
uMenuItem = item number of the Print item
uString = item text of the Print item ("Print")
uResult = label ID = 10

The program can then print the editable text in the BaseCt control (or do whatever else it thinks "Print" should do).

Balloon Help (System 7)

System 7 balloon help can be made available for each item in all main and non-main menus (including control menus in windows). The "normal" way of doing this is to use a special program called "BalloonWriter" that can be purchased from APDA. The use of BalloonWriter is thought by many to be required due to the complex nature of the special hmnu resources that must be created for each MENU that has balloon help associated with it. The truth is that you can use ResEdit to do this if you keep things simple by sticking to a single type of help resource text.

The file "hmnu TMPL" contains hmnu 102, STR# 1002, and TMPL 1000 (named "hmnu"). The presence of the TMPL resource allows ResEdit to edit hmnu resources that are associated with help text in STR#

resources. The hmnu's resource ID of 102 means that it will be used with the MENU that has menuID 102 (not its resource ID!). To associate the hmnu (or a copy of it) with a different MENU, simply reset its resource ID equal to the menuID of the new MENU. (WARNING: Do not try to add the TMPL resource to ResEdit itself since it cannot be used to edit all types of hmnu resources. Keep this TMPL in your own files for use in editing your own hmnu resources.)

When balloon help is enabled, hmnu resources are used to locate the help text displayed as the user moves the cursor over menu items. This help text can be stored in a variety of ways, but, to make it possible for you to use ResEdit, our TMPL only supports the use of STR#-based help text (this happens, however, to be one of the best ways to store help text). This restriction affects the way in which entries in the hmnu template are used.

The entries in an hmnu resource consist of miscellaneous header info followed by repeating blocks that correspond to the menu items:
1. Header info containing "Version", "Options", "ProcID", and "VarCode" - Use the settings in our example hmnu. For more info (don't bother!), see the Help Manager chapter of Inside Macintosh Volume 6.
2. Menu Item Blocks - Following the header are repeating blocks of info corresponding to "missing" items, the menu title, and each menu item. The first block is used for any info that's missing from other menu item blocks ("default" help text). The second block is used for the menu's title, and successive blocks are used for the menu's items. Each block defines help text for four different item states: enabled ("Enb"), disabled ("Dis"), checked ("Chk"), or marked ("Mrk"). The entries in each block are:

"Help Size" - must set to 20 (= byte size of this block)
"Help Type" - must set to 3 (= STR#-based help)
"Enb STR# ID" = ID of STR# containing help text
"Enb Index" = index into STR# resource
"Dis STR# ID" = ID of STR# containing help text
"Dis Index" = index into STR# resource
"Chk STR# ID" = ID of STR# containing help text
"Chk Index" = index into STR# resource
"Mrk STR# ID" = ID of STR# containing help text
"Mrk Index" = index into STR# resource

For example, hmnu 102 contains just one block (the default or "missing items" block) with the help entries 1002, 1, 1002, 2, 1002, 3, 1002, 4. This means that all items in any menu with menuID 102 will have help text from STR# 1002 since only the default block is defined. If the menu item is enabled, then the first string from STR# 1002 is used. If disabled, then the second string is used. If checked, the third is used. If marked, the fourth is used. To define unique help text for each menu item, a title block and a block for each item would have to be added to the hmnu (you can copy and paste entire blocks in ResEdit). Not all entries in a block need to be defined. To skip an entry, set both its STR# ID and index to zero. If help is ever needed for such an item, the default help will be displayed. (If you're reading the Help Manager chapter in IM6 note that you can't really "skip" an item block in the way described in IM6, but setting the STR# ID and index to zero has the same effect.)

The title block operates a little differently from the blocks for other items. First, its "checked" entry is used as help text for the title when the menu has been disabled due to the presence of a modal window. Second, its "marked" entry is used for all items in the menu when the menu has been disabled due to a modal window. Third, if STR# ID and index entries are zeroed in the title block, then the default block is not used and the balloon doesn't appear.

What To Do: Copy the resources from the "hmnu TMPL" file into the program file or other resource file opened by the program. Duplicate and renumber the hmnu resource so that one hmnu is present for each MENU that will have balloon help. Expand each hmnu with entries that define the source of help text for menu items that will have balloon help. Create the associated help text as one or more STR# resources. Run program under System 7 to test.