V3. Windows

ViewIt eliminates the old distinctions between "alerts" , "dialogs", and "windows". Any ViewIt window can be made to look and behave like a typical alert, dialog, or document window. A more important distinction supported by ViewIt is between modal and modeless windows. Modal windows require the presence of the ViewIt and UtilIt modules, whereas modeless windows also require the FaceIt or FaceSt modules to be present.

Modal vs. Modeless

Modeless windows behave in a way that is similar to your program's main menus. The items in such windows, like the items in main menus, are always available to users. This "modeless" behavior is powerful but can be difficult to manage since your program must be able to respond at any time to events in modeless windows, and manage the state of any context-sensitive items in such windows.

Modal windows, on the other hand, are opened and closed as needed, making their management more straightforward. The type of window that you choose to use will depend upon the nature of the task involved. Some have suggested that all windows should be modeless, but we do not share this view. It often makes sense to force a user to deal with the options presented in a modal window before continuing.

Adding Windows

All ViewIt windows are opened from resource templates of type "FWND". These FWNDs can be edited within running programs by entering edit mode (Option-âŒ˜-Shift). New FWNDs are typically created in one of three ways:

1. Use ResEdit to make a copy of an existing FWND, and renumber and rename it for use with your program. Then call NewWnd to open a window based on it, and enter edit mode to edit the FWND.

OR 2. Run a program that already opens a ViewIt window and use the "Edit Another" option from the File menu when in edit mode to open a new ViewIt window. ViewIt will add a new FWND for this window to the program's resource file and the FWND can be immediately edited on-line.

OR 3. Execute a call to NewWnd that refers to an FWND not yet created. ViewIt will then ask you whether you wish to have it create the FWND before continuing. This new FWND will be added to the program's resource file and can then be immediately edited on-line.

Modal Windows

Modal window management by your program occurs within isolated sections of your code in which the modal window is opened, responded to, and closed. These steps correspond to three ViewIt commands (see "Commands" for more info): NewWnd opens a new window, MdlWnd responds to events in the window, and EndWnd closes the window.

Most event processing in modal windows is automatically handled by ViewIt, meaning that very little code needs to be written to open simple modal windows. The following code fragments illustrate this point, and can be copied and pasted into an existing program to open a new modal window:

```
FaceIt(nil,NewWnd,1000,0,0,0);    Pascal
FaceIt(nil,MdlWnd,1000,0,0,0);
FaceIt(nil,EndWnd,1000,0,0,0);

FaceIt(0L,NewWnd,1000L,0L,0L,0L); /* C */
FaceIt(0L,MdlWnd,1000L,0L,0L,0L);
FaceIt(0L,EndWnd,1000L,0L,0L,0L);

call FaceIt(0,NewWnd,1000,0,0,0)   !Fortran
call FaceIt(0,MdlWnd,1000,0,0,0)
```

```
call FaceIt(0,EndWnd,1000,0,0,0)
```

The above code will cause ViewIt to attempt to create a new modal window based on FWND 1000. If FWND 1000 is not present, then ViewIt displays an alert which allows you to add a new FWND to your resource file (as noted above). A default window is then opened and can be immediately edited by entering ViewIt's editing mode.

TIP: If the contents of a window need to be adjusted by the program just after opening it, then keep the window hidden when calling NewWnd to minimize any unsightly screen drawing. One way of doing this is to use -FWND ID:

```
FaceIt(nil,NewWnd,-1000,0,0,0);
```

which keeps the window hidden until MdlWnd is called.

• Item Events

Many modal windows contain at least one control which performs some program-specific operation when pressed. In order to give the main program a chance to respond to clicks in such controls, any ViewIt control can be marked as "enabled" by checking the "Returns On Hit" option in the Control dialog. Clicks in such controls then cause ViewIt to exit from the MdlWnd call and return control to the main program with a menu or "pseudo-menu" event (a message that uses uMenuID, uMenuItem, and other variables).

To handle events from enabled items, a simple event loop can be placed around the call to MdlWnd:

```
FaceIt(nil,NewWnd,1000,0,0,0);
repeat
 FaceIt(nil,MdlWnd,1000,0,0,0);
 if (wcHit = 1) then
  [do first thing]
 else if (wcHit = 2) then
  [do second thing]
until [done];
FaceIt(nil,EndWnd,1000,0,0,0);
```

where in most cases the enabled items will be button-type controls. When returning from MdlWnd, the following fRec variables indicate which control was hit:

uMenuID = FWND ID of parent window
uMenuItem = wiHit = item number
uResult = undefined
uString = undefined
wiHit = item number (w/o regard to view hierarchy)
wvHit = view number (that contains hit control)
wcHit = control number (within parent view)
wClick = click type (1 = single, 2 = double, 3 = triple)
wEvent = raw event

You can think of wiHit as the old, non-hierarchical dialog item number, and wvHit/wcHit as the new, hierarchical way to identify controls. For example, if a window contains 2 views that each contain 5 controls, and the fourth control in the second view is hit, then ViewIt returns wvHit = 2, wcHit = 4, and wiHit = 11 (= first view + 5 controls in first view + second view + 4 controls in second view).

WARNING: Do not assume that the content of any "u", "w", or "c" prefixed fRec variables is preserved across calls to the Control Manager or the FaceIt dispatching procedure. Values that need to be used in multiple Control Manager or FaceIt calls should be preserved in local variables. One exception: Most UtilIt commands preserve "w" & "c" var.s.

• Editable Items

Enabled editable-type controls work a little differently than other types of enabled

controls. In this case a message is posted whenever the enabled editable control is selected or deselected (such as when tabbing between items). The message posted simply informs the main program that the selected state of an enabled editable control has changed:

uMenuID = 1200
uMenuItem = +2 (selected) or -2 (deselected)
uResult = control handle of selected/deselected control
& other event-related variables are undefined

To learn more about the affected control, pass the control handle returned in uResult to GetCtl:

```
FaceIt(nil,GetCtl,0,0,0,uResult);
```

which, for example, would return the item, view, & control numbers in ciIndex, cvIndex, & ccIndex, respectively. Information about the currently selected control can also be found in vSelectCtl, vSelectRec, and vSelectID.

In the case, for example, of tabbing between two controls that are both enabled, ViewIt would post two messages: one for the deselection of the first control (uMenuItem = -2) and a second message for the selection of the second control (uMenuItem = +2). These messages are returned from DoLoop in this order, and your program can choose to deal with one, both, or neither message.

Posting a special message in response to changes in the selected control complements FaceIt's optional support for notifying the program when the active window is changed via a similar message (uMenuID = 1100, uMenuItem = 2). The active window and the currently selected control together define the current program "context". A typical use of these messages is to change the state of menu items when the active window is changed, or to change the options presented in a ViewIt window as the user jumps (by click, TAB, or Command Key) from one editable item to another.

NOTE: The "1200, -2" message is not posted for controls that are deselected before being disposed of (i.e., when their parent window is being closed or when DspCtl is called) or when their parent window is being deactivated.

• Menu Events

The selection of program menu items from menu controls (controls associated with MENU resources) generate menu events that can be distinguished from hits in control items by the fact that uMenuID returns with the menuID of the selected menu (rather than the FWND ID). In this case,

uMenuID = menuID of selected menu
uMenuItem = selected menu item number
uResult = selected menu item label (or zero)
uString = selected menu item text
& other event-related variables are undefined

If the menu control is also enabled, then the program will see both an item hit event and a menu event (in that order). See the FaceIt Guide for a further discussion of "program", "labeled", and "standard" menu items.

Note that menu events are not generated by standard menu items. This means, for example, that the code which runs this help window does not need to deal with any menu events since all of the menu items in the controls at the top of this window are standard items.

• Module Messages

Any FaceWare module can post a message back to the main program at any time. These messages are characterized by uMenuID = baseID of the module, with other event-related variables defining the type of message. The editable item event posted by ViewIt (uMenuID = baseID = 1200, and uMenuItem = 2) is an example of such a message.

• Other Modal Events

ViewIt's Window dialog contains the option "Return Modal Events". If this option is checked, then ViewIt also returns all unprocessed events to the main program (such as disk and AppleEvents). Control returns from MdlWnd with,

uMenuID = 0
wEvent = raw event
& other event-related variables are undefined

where the program is again able to distinguish this event by examining uMenuID.

In the case of disk events, ViewIt will mount uninitialized disks inserted in a floppy disk drive before returning the disk event, so there is no need to have disk events returned for this purpose.

• Event Summary

A modal event loop (a loop around MdlWnd) that responds to both item hits, menu selections, and other events would need to use uMenuID to distinguish these event types:

0 -> unprocessed raw event
menuID -> menu event
FWND ID -> item hit
baseID -> module message

Items hit can then be identified by item number, or view and control numbers.

In practice, most modal event loops are simpler than this since the programmer knows what type of events will be returned. If, for example, "Return Modal Events" is not checked, and the window does not contain any menu controls with program menu items, and there is only one control in the window that returns when hit, and the window is to be closed when that control is hit, then the event loop would be reduced to the original three lines (NewWnd, MdlWnd, and EndWnd).

• Loop Options

Parameter b of the MdlWnd command can be used to force ViewIt to return control to your program. If b = -1, then control is returned without processing any pending events. This might be useful in a situation where an initially hidden modal window needs to be made visible for a time before entering a modal event loop:

```
FaceIt(nil,NewWnd,-1000,0,0,0); modal hidden
[finish hidden window setup]
FaceIt(nil,MdlWnd,1000,-1,0,0); modal shown
[continue visible window setup]
repeat
 FaceIt(nil,MdlWnd,1000,0,0,0); modal in use
 ...
```

If b = -2, then control is returned from MdlWnd after processing any pending events. This can be useful in cases where your program must monitor something or perform some periodic action while a modal window is open. In this case, if no event was the cause of control being returned, then uMenuID (and wEvent.what) will be zero:

```
repeat
 FaceIt(nil,MdlWnd,1000,-2,0,0);
 if (uMenuID = 0) then
  [do your own thing]
 else
  [respond to events]
until [done];
```

Modeless Windows

Modeless windows require use of the FaceIt or FaceSt event handling modules (described

in the FaceIt Guide). The simplest approach is to use FaceIt to get it to do most event handling by calling DoLoop at the top of your program's main event loop. The following discussion assumes that you are using FaceIt, although the menu or pseudo-menu events returned from DoLoop are also seen when using FaceSt.

Modeless windows can be opened or closed at any point in your main program code. As with modals, NewWnd is used to open a modeless window and EndWnd to close it, but, unlike modals, these commands need not be used together in isolated sections of your program code. Modeless windows can also be "auto-initialized" by FaceIt when the program is launched. The auto-initialization string added to STR# 1000 will look something like "1200,20,1000" which, in this case, would be asking ViewIt (baseID 1200, version 2) to open a new modeless window using FWND 1000.

• Modeless Events

Modeless window events are handled within a program's main event loop. One way to think about this is to consider FaceIt's DoLoop command as a replacement for MdlWnd. This is illustrated by the following code fragment:

```
FaceIt(nil,NewWnd,1000,1,0,0); open window
...
repeat
 FaceIt(nil,DoLoop,0,0,0,0);    get events
 if (uMenuID = 1000) then          FWND 1000
  if (wvHit = 2) then              view #2
   if (wcHit = 4) then             control #4
     ...
 else if (uMenuID = 101) then     menuID 101
  if (uMenuItem = 2) then         item #2
    ...
until [done];
...
FaceIt(nil,EndWnd,1000,0,0,0); close window
```

where "done" refers to some condition that causes the loop to be exited, and the final EndWnd command was included to illustrate the relationship between this example and the modal window code presented above. In practice, when and whether modeless windows get opened or closed is under your program's control.

As with modal windows, uMenuID indicates the type of event being returned by FaceIt, and the same variables are used to indicate which control or menu item was chosen. Unlike modal events, however, a modeless event could be from any modeless window, or any menu control, or any main program menu. This means that the program's main event loop around DoLoop is likely to have more cases to consider than typical modal event loops, and that you'll be paying more attention to uMenuID.

As with modal windows, unprocessed events are returned with uMenuID = 0, but the raw event is found in fEvent rather than wEvent, and you cannot stop such unprocessed events from being returned (the "Modal Events" flag set in the Window dialog is ignored).

Enabled editable items in modeless ViewIt windows behave the same as in modal windows, with the same message being posted (uMenuID = 1200, uMenuItem = ±2, uResult = control handle) when the identity of the selected control is changed.

• Modeless-to-Modal

ViewIt supports the temporary conversion of modeless windows to modal windows. A modeless window can be made modal by calling MdlWnd with a = 0 (see the "Commands" topic for a complete description of this command) which causes the window to be brought to the front above all other windows. This call is usually part of a modal event loop in an isolated section of program code:

```
repeat
```

```
    FaceIt(nil,MdlWnd,1000,0,0,0);
    ...
  until [done];
```
which keeps control of the window until some condition or event occurs. The window can then be made modeless again by calling MdlWnd with a = 1:
```
  FaceIt(nil,MdlWnd,1000,1,0,0);
```
which puts the window back in its original position in the window list and marks it as being modeless.

Closing Windows

ViewIt modal or modeless windows that have a close box also support the "Close" standard menu item. A hit in this close box, or selection of the "Close" item, returns with uMenuID = FWND ID and uMenuItem = -1. For modal windows, the variables wiHit, wcHit, and wvHit are also set equal to -1.

A typical response to such a close message will be for the program to close the window by calling EndWnd. Before doing this, however, programs often enquire whether the user wishes to save the contents of the window. If working with controls that support the "Save" message (such as a TextCt text-editing control), these controls can be notified of the pending window closing by calling SavCtl, thereby giving users a chance to save their contents:

```
  ...
  if (uMenuID = 1000) then      FWND 1000 event
    if (uMenuItem = -1) then    close box/item
      begin
        FaceIt(nil,SavCtl,1000,0,0,0);
        if (uResult = 0) then
          FaceIt(nil,EndWnd,1000,0,0,0);
      end;
  ...
```
where the window closing should be aborted if uResult returns a value from SavCtl other than zero, and passing c = d = 0 to SavCtl notifies all controls in the window (see description of SavCtl in "Commands" topic).

Miscellaneous Notes

• If successive modal windows are opened, then their event loops must be "nested" so that each new modal window's NewWnd, MdlWnd, and EndWnd commands are contained within the previous modal window's event loop.
• Windows originally opened as modal cannot be later made modeless.
• Don't open ViewIt modal windows and DialIt modals at the same time.
• Modal ViewIt windows can be shown or hidden at any time using ShoWnd and HidWnd. Note that this will not affect the ordering of existing modal ViewIt windows. You might use HidWnd, for example, to hide a modal window that the user is finished with but that you need to keep around for a time before calling EndWnd.
• Any attempt to close (or switch to modeless state) a modal window that is not the topmost modal window will result in closing (or switching to modeless) all windows above it. This provides a quick way to close multiple modal windows.
• Returning to your main event loop (i.e. where DoLoop is called) when modal windows are still open will cause FaceIt to close (or switch back to modeless) all modal windows.
• The toolbox call "FrontWindow" will not return the top modal window's window pointer, and fRec's fActiveWnd variable is set to nil when a ViewIt modal window is open. Use ViewIt's GetWnd command to get information about modal and modeless ViewIt windows. The window pointer of a new ViewIt window can also be obtained from wWindow after a successful call to NewWnd.