ViewIt Demo Source Code Notes

1. Minimum Code
2. "About Program" Item
3. Modeless Window
4. Modal Window
5. Nested Modal Window
6. Window Structure
7. Window Data Links
8. Control Overrides

One of the best features of FaceWare programming is the small amount of code required to do even complex operations. Only two pages of code were needed to create this "vDemoXY" program!

The following notes describe the steps taken to build vDemoXY from the "MinimumXY" program, and assume that you have at least read the "Startup" topics in the main ViewIt Help window. These notes provide a detailed description of why each line of code appears in vDemoXY, and will help you understand how to add ViewIt windows to your programs. Print the source from one of the vDemoXY programs and examine it while reading the notes.

Although the source appears here in Pascal, source in other languages is included with ViewIt, and has a line-by-line correspondence to the Pascal. If using the HyperFace interface to HyperCard®, the "vDemoHF" stack contains HyperTalk® scripts that also have a line-by-line correspondence to the Pascal.

1. Minimum Code

The lines from vDemoXY corresponding to the MinimumXY program (described in "Minimum Code" in "Startup" topics) are

```
fRec.uName := 'vDemo.Rsrc';
FaceIt(nil,DoInit,0,0,0,0);
...
repeat
  FaceIt(nil,DoLoop,0,0,0,0);
  ...
until false;
```

which, when executed, open the supplemental resource file named "vDemo.Rsrc", initialize FaceWare (DoInit), and handle events (DoLoop). (If using HyperFace, the stack file contains all program resources, and uName is not used to pass file name - see HyperFace guide.)

DoInit is responsible for loading the main menus (Apple, File, Edit, Window) and opening the "ViewIt On-Line Help" window, so no code is needed to accomplish these tasks. DoLoop handles all events from standard menu items and the help window, so, again, almost no code is needed to support these since nearly all of the menu items in the main menus are standard items. (If using HyperFace, then DoLoop is replaced by HyperCard's own event loop, and events needing handling are returned to a stack via the "MainProc" message.)

2. "About Program" Item (not applicable to HyperFace)

The first menu item in the Apple menu is the "About Program" item. Unlike the "Open", "Save", "Cut" and other standard items in the main menus, the "About" item must be handled by the program (since there is no obvious default behavior that FaceIt could execute). When chosen, this item produces an event that is returned from DoLoop: uMenuID = 101 = menu ID of Apple menu, and uMenuItem = 1 = menu item number. To handle this event, we added a little code after DoLoop:

```
...
repeat
  FaceIt(nil,DoLoop,0,0,0,0);
  if (uMenuID = 101) and (uMenuItem = 1) then
    begin
      uString := concat('Demonstration of...
      FaceIt(nil,ShoStr,3,12,...
```

```
    end
  ...
until false;
```
which uses the ViewIt ShoStr command to open a simple alert that describes the program. After responding to the user event, the code loops back to DoLoop again to handle the next event.

3. Modeless Window

We wanted to illustrate several different window types, the first of which was a modeless window (described in "Windows" section of ViewIt guide). Modeless windows can be opened and closed at any point within a program, and events from such windows are returned from DoLoop. In this case, we wanted the modeless window to be opened once at launch time and remain open while the program was running. To accomplish this we added a call to NewWnd after DoInit but before DoLoop:

```
...
FaceIt(nil,DoInit,0,0,0,0);
FaceIt(nil,NewWnd,1000,1,0,0);
repeat
  FaceIt(nil,DoLoop,0,0,0,0);
  ...
```
where a = 1000 indicates that resource FWND 1000 should be used to open the window, and b = 1 indicates that the window is a modeless window. As described in the "Windows" topic of the ViewIt guide, the call to NewWnd can be made even if the FWND resource does not exist since ViewIt can add a new FWND to the program's resource file when the code is first executed.

The modeless window opened is the "Modeless ViewIt Window". Events from this window that need to be handled by the program are returned from DoLoop with uMenuID = FWND ID = 1000. Two of the controls in the window generate such events when hit: the "?" button and the "Open Modal ViewIt Window" button. Both of these controls have the "Return On Hit" option checked in the Control dialog which is why they generate events when hit. To handle these events, the following code was added after DoLoop:

```
  FaceIt(nil,DoLoop,0,0,0,0);
  ...
  else if (uMenuID = 1000) and (wcHit = 2) then
    ...
  else if (uMenuID = 1000) and (wcHit = 3) then
    ...
```
where wcHit is being used to distiguish which button was hit.

4. Modal Window

The "?" and "Open Modal ViewIt Window" buttons are used to illustrate the opening and management of modal ViewIt windows (described in "Windows" section of ViewIt guide). Modal windows are always opened and closed within isolated sections of program code, and the user is forced to deal with the options presented in such windows before continuing.

As described above, the "Open Modal ViewIt Window" button generates an event returned from DoLoop with uMenuID = 1000 and wcHit = 2. In response to this event, we open a new modal window (this window) based on FWND 1001:

```
  FaceIt(nil,DoLoop,0,0,0,0);
  ...
  else if (uMenuID = 1000) and (wcHit = 2) then
    begin
      FaceIt(nil,NewWnd,1001,0,0,0);
      repeat
        FaceIt(nil,MdlWnd,1001,0,0,0);
        ...
      until false;
      FaceIt(nil,EndWnd,1001,0,0,0);
```

```
    end
  ...
```
where NewWnd opens the window, MdlWnd handles the events generated by the modal window, and EndWnd closes the window. The loop around MdlWnd is similar to the main loop around DoLoop, and the events returned from MdlWnd are similar to those returned by DoLoop, but only involve events generated by the modal window.

The modal window opened is the "Modal ViewIt Window". Events from this window that need to be handled by the program are returned from MdlWnd with uMenuID = FWND ID = 1001. The only items which generate such events in this window are the "Open Nested Modal Window" button and the close box at the top, left of the window. Since both of these items return 1001 in uMenuID, we can simply respond to wcHit to distinguish the button hit (wcHit = 1) from the close box hit (wcHit = -1):

```
FaceIt(nil,NewWnd,1001,0,0,0);
repeat
 FaceIt(nil,MdlWnd,1001,0,0,0);
 if (wcHit = -1) then
  leave
 else if (wcHit = 1) then
  ...
 end if
until false
FaceIt(nil,EndWnd,1001,0,0,0);
```

where "leave" exits the modal event loop and causes EndWnd to close the window (i.e., hitting the close box closes the window).

5. Nested Modal Window

The "Open Nested Modal Window" button is used to illustrate the opening of a nested modal window from inside the event loop of an existing modal window. The trick here is to keep all of the code that is used to open and close the second modal window within the event loop of the first modal window (i.e., to "nest" event loops).

As described above, the "Open Nested Modal Window" button generates an event returned from MdlWnd with uMenuID = 1001 and wcHit = 1. In response to this event, we open another modal window based on FWND 1002:

```
FaceIt(nil,MdlWnd,1001,0,0,0);
...
else if (wcHit = 1) then
 begin
  FaceIt(nil,NewWnd,1002,0,0,0);
  ...
  repeat
   FaceIt(nil,MdlWnd,1002,0,0,0);
   ...
  until false;
  ...
  FaceIt(nil,EndWnd,1002,0,0,0);
 end
...
```

which is the same set of commands used to open, manage, and close FWND 1001.

The code added to open the nested modal window completes the overall structure of the vDemoXY program. This structure can be seen more clearly by highlighting just the commands used to open and manage the modeless and modal windows:

```
DoInit
NewWnd,1000,1
repeat
  DoLoop
  if ["About" selected] then
    ShoStr
  else if [button hit] then
```

```
      NewWnd,1001
      repeat
        MdlWnd,1001
        if [close box hit] then
          leave
        else if [button hit] then
          NewWnd,1002
          repeat
            MdlWnd,1002
            ...
          until [done]
          EndWnd,1002
        end if
      until [done]
      EndWnd,1001
    end if
 until [done]
```

In practice, modal event loops are often executed within isolated procedures (or subroutines or functions), but the sequence of execution is the same: events from the main DoLoop loop cause the program to enter isolated modal MdlWnd loops to support modal windows, which in turn can contain other modal loops.

6. Window Structure

In addition to illustrating how to open and close modeless and modal windows, about half of the vDemoXY program is devoted to initializing and managing the contents of the window opened with FWND 1002, the "Nested Modal ViewIt Window". This window is more complex than the other windows, requires extra code to initialize its contents, plus more code to handle events, plus code to return information from the window to the program (i.e., it's more like the "real" windows you hope to create!).

The first thing to understand about the nested window is that it contains three views (enter edit mode to see these):

view #1 - at bottom with "OK" and "Show/Hide" buttons
view #2 - initially shown at top with 4 linked controls
view #3 - initially at top but hidden with a list control

The presence of multiple views means that we will need to pay attention to the view number returned in wvHit when handling events from the window (you can alternatively assign unique ID numbers that get returned in wiHit, but we did not do this here since we wanted to emphasize the view hierarchy).

Four of the controls in the nested window return events when hit: the 2 arrow controls plus the "OK" and "Show/Hide" buttons. In addition to these events, we illustrate the use of "idle" time to animate the icon at the top, right of the window. This is done by calling MdlWnd with b = -2 to inform ViewIt that it should return control to the program with a null event (uMenuID = 0) whenever no events need handling. In all other cases uMenuID returns with 1002 (the FWND ID), so we were able to write the following code to handle all events from the modal window:

```
      ...
      repeat
       FaceIt(nil,MdlWnd,1002,0,0,0);
       if (uMenuID = 0) then
         ...
       else if (wvHit = 1) then
         if (wcHit = 1) then
           leave
         else if (wcHit = 2) then
           ...
       else if (wvHit = 2) then
         if (wcHit = 6) or (wcHit = 7) then
           ...
```

```
    until false;
    ...
```

At this point, the exact actions taken in response to each of these events is not as important as is understanding the logic of the event processing. A brief discussion of each action follows, with references to documentation where further info can be found.

In response to the null event (uMenuID = 0), the current tick count is checked and, if enough time has elapsed, the icon at the top, right is changed (see "Icons, Picts, ..." discussion in BaseCt on-line help for more info about multi-resource icons).

In response to a hit in the "OK" button (wvHit = 1, wcHit = 1), the loop is exited and the window gets closed with EndWnd. If the "Show/Hide" button is hit (wvHit = 1, wcHit = 2), then ShoCtl is called to switch between views 2 and 3 by hiding one and showing the other (see "Views" topic in the ViewIt guide for more info).

In response to a hit in one of the arrow controls (wvHit = 2, wcHit = 6 or 7), the numerical value in the adjacent static text item is increased or decreased by 0.001 units. The value in the static text control was linked to a program variable (discussed below), and can therefore be updated with the command SetVal (see "Data Links" in ViewIt guide for more info).

7. Window Data Links

Windows like the nested modal window which contain controls that display numbers or strings are often linked to program variables. This "data linking" (described in "Data Links" topic of the ViewIt guide) requires a little work to set up, but makes it much easier to move information to and from the window when compared to traditional approaches.

There are 2 different ways to set up data linking, and the nested modal window illustrates both:

Method 1. Four controls in view #2 are linked to elements of the "myRec" program record by passing the memory address of myRec as the last parameter in the call to NewWnd:

```
FaceIt(nil,NewWnd,1002,0,110,ord(@myRec));
```

In addition to passing the address of myRec, each of the linked controls must be assigned a data type and byte offset into the record so that ViewIt can locate the linked variable. These types and offsets are set in ViewIt's Control or Links dialog.

Method 2. The list control in view #3 is linked to the "myList" program variable by passing the control's ControlHandle, the variable's address, and the variable's data type to the LnkCtl command:

```
FaceIt(nil,GetCtl,1002,0,3,3);
FaceIt(nil,LnkCtl,ord(cControl),ord(@myList),2,0);
```

When getting started, you may prefer to use LnkCtl to do your data linking, but many advanced programmers also make use of linking records when such records form a natural part of their program.

Once linked, values in variables can be moved to control values with the ViewIt SetVal command, and copied from control values back to program variables with GetVal. The demo program does this before and after the nested window's modal event loop:

```
    FaceIt(nil,NewWnd,1002,0,0,0);
    ...
    FaceIt(nil,SetVal,1002,0,0,0);
    repeat
     FaceIt(nil,MdlWnd,1002,0,0,0);
     ...
    until false;
    FaceIt(nil,GetVal,1002,0,0,0);
    FaceIt(nil,EndWnd,1002,0,0,0);
```

where passing c = d = 0 to SetVal and GetVal causes them to update all linked controls or variables at once. (In practice, you will usually only call GetVal if the user indicates in some way that their changes are to be saved.)

8. Control Overrides

The final feature illustrated by this demo is that of modifying control behavior with

"override" procedures. An override procedure (or C function or Fortran subroutine or HyperTalk message handler) is a procedure that ViewIt will send control messages to instead of the control's driver. This means that you can intercept any message sent to a control, giving you the power to modify the control's appearance and/or behavior.

The nested modal window contains one editable text control whose behavior is affected by the demo's override procedure "OverProc". OvrCtl is used to link this control to "OverProc" after calling GetCtl to get the control's ControlHandle:

```
FaceIt(nil,GetCtl,1002,0,2,3);
FaceIt(nil,OvrCtl,ord(cControl),ord(@OverProc)...
```

When "OverProc" is called by ViewIt with a message for the editable text control, the procedure first checks whether the message is a key down event (uCommand = 264), and, if so, converts any space characters (uParam[1] = 32) being passed to underline characters (uParam[1] = 95) before passing the message on to the driver. See the "Override" topic in the ViewIt guide for more information about overriding controls.