ViewIt Demo Source Code Notes

One of the best features of FaceWare programming is the small amount of code required to do even complex operations. The notes that follow describe the two pages of source code that was used to create this demo program. Print the source from a "vDemoXY" program and examine it while reading these 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.

Initialization (see "Startup" topics in ViewIt Help window)

All programs that use FaceWare modules call DoInit before any other FaceWare command. The name of a supplemental resource file can also be passed in uName when calling DoInit:

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

where the resource file named by uName is only opened if LoadIt and other resources cannot be found in the application file. (If using HyperFace, the stack file contains all program resources, and uName is not used to pass file name - see HyperFace guide.)

As a convenience, this program then calls HlpWnd (a ViewIt command) to open ViewIt's main on-line help window so that it is readily available while running the demo:

```
FaceIt(nil,HlpWnd,0,0,10,10);
```

HlpWnd is ignored if ViewIt's on-line help is not available, so this call can be left in finished programs.

This demo then opens one modeless window using FWND 1000,

```
FaceIt(nil,NewWnd,1000,1,0,0);
```

which corresponds to the "Modeless ViewIt Window" seen behind this modal window.

Main Event Loop (see "The Main Loop" in FaceIt Guide)

By making use of the FaceIt module, the demo's main event loop can be devoted to handling program-specific events, leaving the rest of the work to FaceIt:

```
repeat
  FaceIt(nil,DoLoop,0,0,0,0);
  ...
until false;
```

where control is only returned from DoLoop if an event occurs that requires handling by the program.

If using HyperFace with HyperCard, then DoLoop is replaced by HyperCard's own event loop, and events needing handling are returned to a stack via the message "MainProc":

```
on MainProc
  ...
end MainProc
```

where the events returned will be very similar to those seen in a FaceIt-based program that calls DoLoop.

Most of the code in this program's event loop (or "MainProc" message handler) is devoted to handling just 1 event: a click in the "Open Modal ViewIt Window" button at the bottom of the modeless ViewIt window. Other events such as the selection of menu items from the main menu bar are automatically handled by FaceWare modules (or by HyperCard), since these items are "standard" items with default behavior.

The type of event returned from DoLoop (or to the "MainProc" message handler) is designated by uMenuID. In the case of a hit in the "Open Modal" button, the program sees uMenuID = 1000 = FWND ID of the window, and wcHit = 2 = number of control hit in the window:

```
...
else if (uMenuID = 1000) and (wcHit = 2) then
  begin
   FaceIt(nil,NewWnd,1001,0,0,0);
```

```
    ...
  end;
```
which results in NewWnd being called to open a new modal window (this window) based on FWND 1001.

Modal Event Loops (see "Windows" in ViewIt Guide)

Modal ViewIt windows are controlled by "modal event loops" that are isolated sections of program code. Like DoLoop, the ViewIt command MdlWnd ("Modal Window") is used to get ViewIt to handle most events in modal windows:

```
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, in the case of this window, the only events that require program attention are hits in the close box (wcHit = -1) or the button at the top of the window (wcHit = 1 = "Open Nested...").

The response to a hit in the "Open Nested..." button is to open a new modal window based on FWND 1002 that has an event loop that is "nested" within the first modal window's loop:

```
...NewWnd,1001...
...MdlWnd,1001...
   ...NewWnd,1002...
   ...MdlWnd,1002...
   ...EndWnd,1002...
...EndWnd,1001...
```
where the second modal window is being opened and closed within the first modal's event loop.

Data Linking (see "Data Links" in ViewIt Guide)

The nested modal window based on FWND 1002 includes four controls that are "linked" to variables within a program record named "myRec". This link is established by providing ViewIt with the address of the record in the last parameter of NewWnd,

```
FaceIt(nil,NewWnd,1002,0,110,ord(@myRec));
```
and by setting the "Data Link" options for each linked control.

Once data linking has been established, then the linked controls in the window can be updated to display the "myRec" variables by simply calling SetVal,

```
FaceIt(nil,SetVal,1002,0,0,0);
```
and changes made to control values by the user can be retrieved by calling GetVal,

```
FaceIt(nil,GetVal,1002,0,0,0);
```
where SetVal is called after opening the window with NewWnd, and GetVal before closing it with EndWnd.

Hiding/Showing Views

ViewIt programmers sometimes use multiple views in the same window to support the "paging" of groups of controls. This can reduce the number of windows needed by a program since many more options can be packed into a single ViewIt window without the window becoming cluttered.

The nested modal window in this demo provides an illustration of the use of multiple views in the same window. The window has three views, one of which is always visible, and two others that alternate between being visible or hidden when the "Hide/Show" button is pressed. ShoCtl is used to hide and show views:

```
 ...
```

```
 else if (wcHit = 2) then   hit in "Hide/Show"
   if helpShown then
     begin
      FaceIt(nil,ShoCtl,0,0,-3,2); hide v3, show v2
      helpShown := false;
     end
   else
     begin
      FaceIt(nil,ShoCtl,0,0,-2,3); hide v2, show v3
      helpShown := true;
     end;
```

Override Procedures (see "Override" in ViewIt Guide)

Another powerful ViewIt feature is its support for "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 has three controls whose behavior is affected by the demo's override procedure "OverProc". OvrCtl is used to link each of these controls 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),0,0);
FaceIt(nil,GetCtl,1002,0,2,6);
FaceIt(nil,OvrCtl,ord(cControl),ord(@OverProc),0,0);
FaceIt(nil,GetCtl,1002,0,2,7);
FaceIt(nil,OvrCtl,ord(cControl),ord(@OverProc),0,0);
```

where v2c3 is the editable text item, and v2c6 and v2c7 are the arrow icon buttons.

When the "OverProc" procedure is called with a message for one of these controls, it first determines whether the control is an arrow button (cResID = 1000 = SICN ID) or editable text (cResID = 0), and then selectively modifies certain messages. In the case of the arrow controls, the override procedure handles all mouse down messages and responds to these by incrementing or decrementing the value of the number in another control. In the case of the editable text item, the override procedure looks for space characters and converts them to underline characters. In all other cases the message is passed on to the driver so that it can perform its default action.