F8. Hybrid Programs

Modal FaceWare windows and the commands supported by the UtilIt module can be used within any program. Modeless windows, however, require use of the FaceIt or FaceSt event handling modules. Most FaceWare programmers use FaceIt, but FaceSt is useful in cases where modeless windows must be added to programs or programming environments that do their own low-level event handling.

What Is FaceSt?

FaceSt ("FaceStub") is a scaled-down version of the FaceIt module that accepts FaceIt commands but does not provide most of the functionality of FaceIt. It primarily serves to pass events and messages between the main program and FaceWare window drivers like ViewIt, and does not itself call WaitNextEvent or GetNextEvent. Rather, FaceSt must be fed events by the main program. This makes it possible to add FaceWare window event handling to existing event loops, and thereby support modeless FaceWare windows within any program or programming environment.

When To Use FaceSt

FaceSt is most useful when either (a) you are trying to add FaceWare modeless windows to one of your programs that already has a complex event loop and its own windows, or (b) you are adding modeless FaceWare windows to a high-level programming environment that does not give you direct access to its main event loop. In the latter case we may have already done most of the work needed to make FaceSt work in the environment (ProFace for Prograph, HyperFace for HyperCard, etc.).

When NOT To Use It

If you are new to Mac programming then use FaceIt before trying FaceSt. If you are using FaceIt but think you need program-specific windows that differ from the FaceWare windows, then try using ViewIt windows for such windows and overriding control behavior within these windows. Responding to control messages in a control override proc is much easier and more powerful than creating windows "from scratch".

What You GAIN

FaceSt supports any ViewIt, TextIt, GrafIt, or ShowIt modeless window and their associated standard menu items. Without FaceSt, programs can only make use of FaceWare windows that are modal since these can be opened and closed without interacting with the main program's event loop.

What You LOSE

The following functionality is lost when using FaceSt in place of FaceIt (much of this, however, may already be provided by the program to which FaceSt is being added):

- auto-loading of main program menus
- auto-initialization of windows (from STR# 1000)
- adding font names to STR# 1106
- all help opening or printing files "from the Finder"
- DoLoop command (replaced by program's event loop)
- the program-wide standard items "About", "Delete", "Transfer", "Quit", "Select", "Hide", "Send Behind", "Send to Back", "Hide Others", and "Show All"
- all Desk Accessory menu and event handling
- picture palette floating menus
- auto-hiding of non-FaceWare windows that generate update events
- auto-closing of modal ViewIt windows left open
- several of the DoInit options defined by parameter a (see DoInit description under "Commands" topic)

Initializations Done

Toolbox initializations, stack space resetting, and program palette resetting to clut 1000 are still done. The toolbox initializations can be disabled by adding 1 to parameter c when calling DoInit. Stack space resetting can be disabled by passing b = -1 when calling DoInit. Palette resetting can be disabled by removing clut 1000 or by adding 8 to parameter c when calling DoInit. (All of these options are standard

features of the DoInit command.)

STACK TIP: Many programmers do not realize that calling "MaxApplZone" has the effect of permanently fixing the size of the stack to the default size set by the System. Since FaceWare modules use additional stack space (especially during on-line editing of ViewIt windows), we recommend letting the DoInit command make the "MaxApplZone" call which it will do when resizing the stack (b > -1).

Installing FaceStub

FaceSt takes the place of FaceIt. The simplest way to make FaceSt available to a program under development is to use MoveIt to move a copy of FaceSt from the FaceSt.FCMD file to a resource file opened by the program. (You'll also need LoadIt and any other program-specific resources required by modules used by the program). FaceSt will then be used in place of the FaceIt module in the FaceWare file, thereby avoiding the need to modify the FaceWare file.

Using FaceStub

The "FaceStub" folder in the ViewIt demo folder contains a set of files whose names contain the "*" character. These files correspond to the original vDemoLP example (which uses FaceIt), but have been modified to make use of the FaceSt module. The source file contains all of the code from the original vDemoLP program, plus new code that takes the place of functionality lost from not using FaceIt. The new code is denoted by comments of the form "* ... *", so it is easy to distinguish this code from the original.

The following discussion assumes that you are familiar with vDemoLP*, so you may want to print the source and try running the program before reading further.

• Main Menus

Programs that use FaceSt are responsible for installing main menus. In vDemoLP* this is done using SetItm2 so that any labeled items ("#n") get processed and stored. If you are working in an environment where you do not have an opportunity to use SetItm2 to load entire menus, then you can still use SetItm2 to add labels to individual menu items after a menu has been installed. The following, for example, would make menu item 3 in a menu with menuID 129 the standard "Save" item (#5):

```
FaceIt(nil,SetItm2,129,3,11,5);
```

Labeled items operate as expected, with their appearance and behavior being controlled by the current program "context". If the active window is not a FaceWare window, then FaceSt disables all standard labeled items. If you wish to make use of such items while a non-FaceWare window is active, then have control returned whenever the active window is changed (by adding 2 to parameter a when calling DoInit), and respond by reenabling items that you wish to make available. vDemoLP* uses this approach to make the standard "Clear" item available when its non-FaceWare window "myWindow" is active:

```
FaceIt(nil,DoInit,2,0,0,0);
...
if (uMenuID = 1100) and (uMenuItem = 2) then
  if (fActiveWnd = myWindow) then
   FaceIt(nil,SetItm2,103,8,2,1); enable Clear
```

WARNING: Labeled items in a menu must be "de-labeled" before disposing of that menu. This can be done by using SetItm2 to dispose of the menu, or by using SetItm2 to remove one label at a time from the items in the menu. ("Removing a label" means that UtilIt removes references to that item from its private labeled item data structures.) In many cases, however, programs never dispose of the menus containing standard items and need not be concerned with this issue.

• Menu Events

FaceIt-based event loops begin with DoLoop and then proceed with the processing of any menu or pseudo-menu events returned by FaceIt. When using FaceSt, the DoLoop command is replaced by the main program's own event loop that processes raw events returned by WaitNextEvent. In vDemoLP* this event loop is contained in the "MyLoop" procedure (described below). MyLoop was set up to return any menu or pseudo-menu events to the main case block. This makes the main case block of the program look similar to that seen in all FaceIt-based programs:

```
repeat
  MyLoop;
```

```
    if (uMenuID = 101) then
      ...
  until false;
```
The only difference (other than calling "MyLoop"), in fact, between this block and that seen in the original vDemoLP program is that there are more menu items to handle corresponding to things not done by FaceSt or to new things done with the program's own windows: opening DAs, Quit, Clear, etc.

Note that when adding FaceSt to existing programs, there is no reason to split event handling into a raw event loop like "MyLoop" and a separate menu event block like that shown above. We simply did it this way to illustrate the relationship between programs that use FaceIt and those that use FaceSt.

• Raw Event Loop

Most of the new code in vDemoLP* is found in its "MyLoop" procedure. This procedure performs the 4 event-related tasks that are required by programs using FaceSt:

1. keep identity of active window updated
2. process any messages returned by modules
3. process any pending events returned by WaitNextEvent
4. give idle/hook time to modules

where these tasks must be performed in the above order (i.e., the identity of the active window must be correct before getting messages, all messages must be processed before getting raw events, and all events must be processed before giving idle/hook time to modules). A description of each task follows:

1. The first task is to determine if fActiveWnd contains the active window's window pointer. In many programs the active window is simply the front window, but in others it is the first window below floating windows. Your program should do whatever it needs to do to find its active window, and then call DoUpdt2 with d = 8 to update the fActive... variables in fRec if fActiveWnd is not correct. In the vDemoLP* program "FrontWindow" is used to make this check:

```
  if (fActiveWnd <> FrontWindow) then
    FaceIt(nil,DoUpdt2,0,0,0,8)
```
Also note that calling DoUpdt2 with d = 8 will reset the current port to the active window if it is a FaceWare window, but all other calls to FaceSt preserve the current port. This differs from the behavior of FaceIt which resets the port to the active window each time DoLoop is called.

2. The second task is to check if there are any pending messages (= menu or pseudo-menu events) that have been posted by modules. These special messages are stored in a private queue maintained by FaceSt and UtilIt. The number of available messages is given by fMsgCount, and the next message can be removed from the queue by calling GetMsg. The vDemoLP* program leaves the loop in MyLoop after getting such a message in order to mimic the behavior of the FaceIt module (which returns control from DoLoop with a menu or pseudo-menu event), but you could alternatively call a separate procedure to handle such pseudo-menu and menu events:

```
  else if (fMsgCount > 0) then
    begin
      FaceIt(nil,GetMsg,0,0,0,0);
      leave;
    end
```

3. The third task is to process any pending events returned by WaitNextEvent. In general, the program must determine whether the event belongs to one of its windows or to a FaceWare window. If it belongs to a FaceWare window, then the event is passed to FaceSt via fEvent and the DoEvnt command. The code in vDemoLP* illustrates the typical logic used to determine when DoEvnt should be called. In general, do not assume that you know how to handle an event belonging to a FaceWare window. (You might think, for example, that you already know how to drag windows and can therefore call DragWindow to drag FaceWare windows, but doing so will rob the window of an opportunity to update its zoom states and mess up future zooming of the window.)

Menu selections returned by "MenuSelect" or "MenuKey" are most easily processed by passing them to FaceSt. If FaceSt does not know how to deal with the item (i.e., if it is not a standard item), then it posts a message back to the program that will be seen as a menu event returned by GetMsg (which explains why vDemoLP* has no routine for handling just menu selections). You could alternatively use GetItm to

pre-process a menu selection to determine if it corresponds to a standard item, but FaceSt already contains such code so it does not make much sense to do this in your program.

Finally, note that it is not necessary to process events in exactly the manner shown in the vDemoLP* program. In some environments, for example, you may be fed events or menu selections by code that you cannot modify. In such cases, however, you can still do tasks #1 and #2 before processing an event that was passed, can still call DoMenu to further process a menu selection, and should still remain in a loop until all module messages have been processed.

4. The fourth task is to give idle and hook time to modules. This is done by simply using DoEvnt to pass a null event in fEvent:

```
  else
   FaceIt(nil,DoEvnt,0,0,0,0);
```

• Switching

FaceSt tracks whether a program is switched in (is the top application) or out (is in the background) under System 7 or MultiFinder by updating the fSleep variable in fRec. If switched in, then fSleep = fFrontSleep. If switched out, then fSleep = fBackSleep. This scheme relies upon the feeding of "osEvt" events to FaceSt that indicate when the program is about to be switched in or out.

Upon receiving an "osEvt" event, FaceSt updates fSleep, notifies all affected FaceWare windows, and then activates or deactivates the active window if it is a FaceWare window. To ensure that your program sees and responds properly to osEvt events, both the "Accept suspend events" and "Does activate on FG switch" flags in the SIZE resource should be set. If these flags are not set, then no osEvt-type events will be seen, and FaceSt will not have an opportunity to notify FaceWare windows when a switch occurs.

• Window Kind

The identity of the active window can be determined from the fActive... variables in fRec. If fActiveID = 0, then the window is not a FaceWare window.

The "refCon" and "windowKind" fields of each window's window record are not used by FaceWare windows, meaning that a program can use these fields for its own purposes.

If using ViewIt, then the GetWnd command can be passed the window pointer of any window, and resets wResID to indicate whether the window is a ViewIt window: 0 = not a ViewIt window, other = FWND ID associated with window.

A faster way to determine whether a window is a FaceWare window is to examine 4 bytes within the "windowDefProc" handle associated with the window. If the window is a FaceWare window, then the 4 bytes located 6 bytes into this handle block will be equal to the fRec variable fWDEF. The following code illustrates use of this fact to check for a FaceWare window:

```
/* C */
 if (fRec.fWDEF == *(long*)(6 + (long)*theHdl))…
Pascal
 if (fRec.fWDEF = long(6 + ord(theHdl^))^ then…
C AF Fortran
 if (fRec.fWDEF == long(6 + long(theHdl)) then…
```

where "theHdl" is the "windowDefProc" handle from the window's window record, and "long" in the Pascal source is a type defined as "long = ^longint".

• Cursor Management

FaceSt takes no responsibility for managing the cursor when a non-FaceWare window is active. vDemoLP* handles cursor management by resetting the cursor to an arrow when a non-FaceWare window becomes the active window (indicated by fActiveID = 0):

```
 if (uMenuID = 1100) and (uMenuItem = 2) then
   if (fActiveID = 0) then
    FaceIt(nil,ChgCur,0,0,0,0);
```

where the use of ChgCur also ensures that fCursor in fRec is reset to -1 to inform FaceSt that the cursor will later need updating if a FaceWare window is made active. In general, if the program changes the cursor without calling ChgCur, then set fCursor to -1 so that FaceSt knows that the cursor will need updating.

• Program Termination

When quitting a program that contains FaceWare windows, these windows should first be sent two messages: a "Save Documents" message that gives the user a chance to save changes before a window is closed, and a "Clean System" message that gives the window driver a chance to clean up stuff before the program quits.

These two messages can be posted via the UtilIt command PstNot ("Post Note"), where a = 4096 = Save Documents, and a = 32 = Clean System. In the case of Save Documents message, uResult will return with a non-zero value if the user cancels the operation (in which case you should not quit the program). vDemoLP*, for example, posts these messages when responding to its Quit menu item:

```
FaceIt(nil,PstNot,4096,0,0,0);
if (uResult = 0) then
 begin
  FaceIt(nil,PstNot,32,0,0,0);
  ExitToShell;
 end;
```

• Custom Procedures

FaceSt (and FaceIt) make use of several procedures whose addresses are stored in fRec. Any of these addresses can be replaced by the address of a program procedure (Pascal type) that will then be called by FaceSt or FaceIt in place of the default procedure. The procedure addresses most likely to be changed by programs using FaceSt are (where "fPtr" is the address of fRec):

address: fUpdateOther
form: procedure MyUpdateOther(fPtr:FacePtr; theWindow:WindowPtr);
purpose: Updates the content of the designated window. This procedure is called by FaceSt or ViewIt whenever it finds a window that it does not know how to update. It is usually a good idea to reset this address so that windows below modal ViewIt windows get updated when the modal window is dragged (the default procedure does nothing). vDemoLP* illustrates resetting fUpdateOther with the address of its own update procedure:

```
fRec.fUpdateOther := ord(@MyUpdateOther);
```

address: fSelectWindow
form: procedure MySelectWindow(fPtr:FacePtr; theWindow:WindowPtr);
purpose: Brings the designated window to the front. FaceSt's default procedure simply calls SelectWindow. If the program supports its own floating window scheme, then you may need to replace this procedure with one that does not bring the window above floating windows (unless SelectWindow itself has already been patched).

address: fActiveWindow
form: function MyActiveWindow(fPtr:FacePtr) : WindowPtr;
purpose: Finds the active window. FaceSt's default procedure simply calls FrontWindow. If the program supports its own floating window scheme, then you may need to replace this procedure with one that finds the topmost window beneath any floating windows (unless FrontWindow itself has already been patched).

address: fNewWindow
address: fNewCWindow
address: fGetNewWindow
address: fGetNewCWindow
address: fDisposeWindow
form: same as corresponding toolbox call, but with additional "fPtr:FacePtr" as first argument
purpose: Creates or disposes of window. These are only used by FaceWare modules when creating or disposing of modeless windows. Substitute your own procedures in cases where the environment in which you are working requires the use of special commands to open or close windows (i.e., if it needs to perform additional operations when a window is created or destroyed).

• FSSC Menus

UtilIt supports Font, Size, Style, and Color menus that can be made available via main program menus in programs using FaceSt if a little work is done by the program to set up these menus properly. Note that this discussion only applies to main program menus, and not to the pop-up or pull-down menu controls found in ViewIt windows.

The first step is to make the FSSC menus available as hierarchical menus by adding hierarchical menu items associated with menuIDs 196 (Font), 197 (Size), 198 (Style), and 199 (Color) to one of the program's main menus.

The next step is to add code to the program's event loop that checks to see if the current context supports the FSSC menus, and, if so, then enables these menus just before MenuSelect or MenuKey is called, and disables them after returning from these calls. For example, to determine whether an active FaceWare window supports the FSSC menus, check whether an editable control is selected in the window (vSelectCtl ≠ nil) and whether that control supports the "UsesFSSC" bit flag (BitTst(@cType,9)):

```
 if (vSelectCtl <> nil) then
   FaceIt(nil,GetCtl,0,0,0,ord(vSelectCtl));
 if (vSelectCtl<> nil) and BitTst(@cType,9) then
   begin
     EnableItem(fFontMenu,0);
     EnableItem(fSizeMenu,0);
     EnableItem(fStyleMenu,0);
     EnableItem(fColorMenu,0);
   end;
 theCommand := MenuSelect(fEvent.where);
 if (HiWord(theCommand) <> 0) then
   FaceIt(nil,DoMenu,HiWord(theCommand),...
 HiliteMenu(0);
 DisableItem(fFontMenu,0);
 DisableItem(fSizeMenu,0);
 DisableItem(fStyleMenu,0);
 DisableItem(fColorMenu,0);
```

It is also possible for a non-FaceWare window to make its own use of the FSSC menus. In this case the enabling and disabling shown above must also be done if such a program window is active, and an additional step is needed to update the content of the FSSC menus to reflect the state of the program window. This can either be done using UtilIt's FixFSC command when the window becomes active (and whenever its current Font, Size, Style, or Color is changed while the window is active), or by calling FixFSC just before MenuSelect and MenuKey are executed. NOTE: Avoid calling FixFSC when a FaceWare window is active since this will clobber any settings made by FaceWare modules associated with the window.

Selection of an item from an FSSC menu is automatically processed by FaceSt to determine whether the selection would change the state of the FSSC menu. If the FSSC menu would be changed, and the active window is not a FaceWare window, then FaceSt returns a program menu item event to the program (as a message returned by GetMsg), and adjusts the current port's corresponding txFont, txSize, txFace, or fgColor (or rgbFgColor) WindowRecord field to reflect the selected font, size, style, or color. In the case of a style change, txFace is set to contain just the selected style item (Plain OR Bold OR Italic...) so that the selected style can be turned on or off without affecting other styles in the window. The program can then read the updated font, size, style, or color from the window record and adjust the contents of its window accordingly.