V8. Override (Advanced)

Every message sent by ViewIt to a control driver can be intercepted by a main program. This allows the program to selectively modify all aspects of control behavior. At one extreme, this feature can be used to tweak the behavior of an existing control (such as filtering key events in some program-specific manner). At the other extreme, controls can be completely managed by program code that intercepts and handles all messages from ViewIt, allowing, in effect, the program to control any region of a ViewIt window.

Also note that this scheme is much more powerful than the Dialog Manager's support for "userItem" procedures, or the Control Manager's support for "action" procedures, since ViewIt allows you to intercept all control messages passed to any ViewIt control.

An example of intercepting control messages is presented in the "vDemoXY" program. The following notes review the installation, format, and operation of program "override" procedures that are used to intercept messages.

Direct Installation

The most direct method of intercepting control messages is to reset the "ccOverride" variable found within a control's supplemental control record (Pascal source):

```
 cInfo^^.ccOverride := @MyProc;
```

Supplemental control records and other driver details are discussed in Inside FaceWare, but we have provided a direct ViewIt command, OvrCtl, that can be used by programmers who are not familiar with such details. OvrCtl requires a control handle which can be obtained from GetCtl (see the "Commands" topic for details). For example,

```
 FaceIt(nil,GetCtl,0,0,1,5);
 FaceIt(nil,OvrCtl,ord(cControl),ord(@MyProc)...
```

would get the control handle for the 5th control in the first view, and reset its override procedure to "MyProc". See "vDemoXY" for an example using your preferred language and compiler.

vs. Proc ID Table

The direct method of installing an override procedure has two main disadvantages: the control must already exist in a ViewIt window, and OvrCtl must be called for each control that is to be overridden. These disadvantages are addressed by an alternative method based upon the assignment of an "Override ID" number to each override procedure. These ID numbers are arbitrary 2-byte integers passed to ViewIt in place of the control handle in OvrCtl:

```
 FaceIt(nil,OvrCtl,1001,ord(@MyProc),0,0);
```

where "1001" will be the ID associated with "MyProc", and the ID/ProcPtr pair is stored by ViewIt in a private table.

OvrCtl can be used to add new entries to the ID/ProcPtr table at any time, whether associated controls have been initialized or not. To remove an entry, pass the ID number of the entry to be removed and zero for the proc address:

```
 FaceIt(nil,OvrCtl,1001,0,0,0);
```

To associate a control with a procedure in the ID/ProcPtr table, set its "Override ID" in ViewIt's Control dialog to the ID number of the procedure. This override ID is saved with control-related resources, meaning that such controls can be added, pasted, imported, duplicated, etc., without losing their link to the override procedure. If the override ID is zero, or is not found in the table, then ViewIt ignores it and calls the driver directly.

Procedure Format

Override procedures must be of type "Pascal", and have a single, 4-byte parameter. When working within an FCMD, this parameter is used to recover the global fRec address and other information (see Inside FaceWare for details). When working within a main program, however, the fRec is available as a global record and so the passed parameter is not needed to access fRec (although it is used in jumping back to the driver - see below). See "vDemoXY" for an example using your preferred language and compiler.

Procedure Execution

Before ViewIt sends a message to a control, it checks the "ccOverride" variable. If this variable is not zero (i.e., it is a procedure address or ID), then the override procedure is called instead of the control driver. This means that the override procedure will see all control messages and can selectively override or modify any message.

Before jumping to the override procedure, ViewIt places the control handle in cControl, the baseID of the control driver in cBaseID, copies the control's private records to the bottom of fRec (cNext to cString), sets uCommand to the message constant, and uParam to its parameters (if any). In many cases, you'll want to pass messages on to the control driver so that it can execute its default behavior. This is done by calling the "JumpIt" procedure using the 4-byte parameter passed to the override procedure (see "vDemoXY" for example code).

The actions taken by an override procedure can vary from doing almost nothing to taking complete responsibility for the corresponding area of the window. These actions often correspond to one of five types:

• Posting Commands

One approach to take is to post a pseudo-menu event back to your own event loop before passing the message on to the driver. This approach is relatively safe because it does not involve making any changes to fRec variables or executing code that might affect the default operation of the driver. A pseudo-menu event can be posted using UtilIt's PstEvt:

```
 ...
 if [some condition] then
   begin
     BlockMove(@uCommand,@saveMessage,20);
     FaceIt(nil,PstEvt,13,b,c,d);
     BlockMove(@saveMessage,@uCommand,20);
   end;
 JumpIt(thePtr);   pass message to driver
 ...
```

where "saveMessage" is a 20-byte array being used to save uCommand and uParam for use by the driver, and where a = 13 puts the event in a private event queue, and b, c, & d correspond, respectively, to the uMenuID, uResult, and uMenuItem values that will be seen by the program's main event loop. uMenuID can be thought of as the "class" of message, uMenuItem as a message ID, and uResult as any accompanying data. To avoid conflicts with other uses of uMenuID, choose a value that is outside of the range of FCMD IDs (1100-7499), and is not equal to any label ID, menuID, or FWND ID (a number > 7499 is a good choice). NOTE: uMenuItem gets temporarily stored in the 2-byte "modifiers" field of an event record, so don't try to use it to pass 4 bytes of information. uResult, on the other hand, gets stored in the 4-byte "where" field, so you can use this to return addresses, handles, etc.

• Stealing Messages

Another safe action to take is to simply steal the message and not pass it on to the driver. An example of this can be found in "vDemoXY" where clicks in the arrow buttons are completely managed by the override procedure.

• Tweaking Messages

Another safe action is to make minor changes to messages being passed to drivers. An example of this can be found in "vDemoXY" where its override procedure converts all space characters to underline characters passed to the editable text item.

• Major Action Before

More complex actions within override procedures that are taken before passing a message to a driver must be careful to preserve the fRec variables set by ViewIt for use by the driver (cNext to cString, uCommand, and uParam). The "c" variables, for example, can be reset using GetCtl, but the "u" variables must be saved in program variables:

```
 ...
 BlockMove(@uCommand,@saveMessage,20);
 saveControl := cControl;
 [do stuff that clobbers fRec]
 FaceIt(nil,GetCtl,0,0,0,ord(saveControl));
 BlockMove(@saveMessage,@uCommand,20);
 JumpIt(thePtr);
 ...
```
where "saveMessage" is a 20-byte program array being used to save uCommand and uParam (original message).

• Major Action After
More complex actions within override procedures that are taken after passing a message to a driver must consider that the driver is allowed to change any "c" or "u" variables for its own purposes. Again, GetCtl can be used, if necessary, to update the contents of fRec if the control's control handle has been saved:
```
 ...
 saveControl := cControl;
 JumpIt(thePtr);
 FaceIt(nil,GetCtl,0,0,0,ord(saveControl));
 ...
```
Also note, if executing code after "JumpIt", that the fRec variables uResult and uMenuHdl are sometimes used to return information from the driver to ViewIt, so you will need to preserve these for some types of messages.

Control Message Documentation
The messages sent to control drivers are described in the source code of the "ShellCt" driver. This file can be found in the "Other vStuff" folder along with "FaceStorLP2" which contains various message, menu, and part constants. For simple override procedures these files will provide adequate information about the messages sent by ViewIt to control drivers. For more complex cases, or when you are interested in developing new drivers, you should purchase the Inside FaceWare product which contains a complete description of the operation of control drivers.
When examining ShellCt, start with the case block at the bottom of the file, and then study just those procedures that support the messages that you wish to override. As you encounter undefined constants, examine the FaceStorLP2 file to determine their numerical values. Tip: You may need to reset the tab size to 1 or 2 when viewing these files with a text editor.

Program-Controlled Regions
The more messages handled by an override procedure, the closer the procedure is to a full-blown control driver. In the extreme case where all, or nearly all, messages are handled by an override procedure (w/o passing them on to a control driver), then the main program can be said to be in control of the window area bounded by the control.
The advantage of controlling an area of a window through such an override procedure is that ViewIt continues to treat the control as a ViewIt control. It can be resized, moved, hidden, inactivated, etc., by executing toolbox or ViewIt commands, and from within editing mode. ShellCt controls (try the "Empty Shell" example FCTL) are the best controls to use if overriding most messages since the ShellCt driver contains very little code and has minimal default behavior.
WHAT TO DO: To place a region of a ViewIt window under direct program control, add a dummy ShellCt control and override all or most of the messages sent to it by ViewIt.

Limitations & Warnings
- Most ViewIt controls are very flexible. Do not waste time attempting to override behavior until you fully understand a control's default capabilities.
- Messages to standard CDEF controls cannot be intercepted.
- The override procedure must be within a code segment that is never moved or removed (unless you are taking the trouble to keep the override address updated).
- The "initCntl" message will not be seen by an override procedure that was installed by the direct method. Any initialization-related code that you wish to add, however, can be executed just after the override proc is installed.
- Controls that are reinitialized from within editing mode will lose their connection to an override procedure

if the procedure was installed by the direct method.
- The fRec variables ciIndex to ccIndex are not updated by ViewIt when calling control drivers and override proc.s.   The reason for this is that it would take ViewIt too long to calculate these values when passing messages to windows with large numbers of controls.   If you need such indices for use within override proc.s, then use GetCtl with the control handle passed in cControl (be careful to preserve uCommand and uParam!).   A better approach, however, is to not rely on control or view numbers, but rather write procedures that understand how to deal with control types.   Controls can be easily distinguished by variables such as cBaseID (the driver type), cResType (linked resource type), cDataType (linked data type), cRefCon (a value set by you), etc.