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. Note that this "override" capability 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.

At one extreme, overriding 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 ("program controls") that intercepts and handles all messages from ViewIt, allowing, in effect, the program to control any region of a ViewIt window.

This topic describes the 3 ways that a program can override ViewIt controls: 1) content drawing within ViewBV controls by a main program procedure, 2) "program controls" that are completely supported by main program code, and 3) overriding of selected messages by a program procedure. These 3 options represent different degrees to which the main program can take responsibility for a control's appearance and behavior.

CONTENT DRAWING

The ViewBV controls described in the "Views" topic support the use of a main program procedure that does the drawing of the view's content. ViewBV can also support non-view controls in a ViewIt window. The "Program Content" example control seen in the Import menu when in edit mode is an example of such a non-view control that is supported by ViewBV (try adding it to a ViewIt window).

The "Program Content" control handles scrolling and drawing of its body and frame, but does not display any "content" unless it is associated with a program drawing procedure (described under "Content" in the "Views" topic). The scrolling is done using ViewIt's built-in support (described under "Scrolling" in "Controls" topic), meaning that the ScrCtl command can be used by a program to directly scroll and resize the content area of the control, and that changes to the content and increment settings in the Bounds dialog will apply.

Scrollable controls based on the ViewBV driver offer a simple way to support scrolling of program-specific drawing. All of the display options of views are supported: scroll bars, hand scrolling, activation/deactivation with active state of window, color, frame thickness, attachment, etc.

The major limits on such controls are:

- maximum 30,000-pixel height and width of content area
- the scrolling increment (set in Bounds dialog) is limited to 255 pixels and is the same in both directions
- the controls must be type "Static" (i.e., display only)

PROGRAM CONTROLS

The "Program Controls" folder included with ViewIt contains the demo projects in C++, C, and Pascal that illustrate how to create "program controls" which are completely supported by main program code (i.e., areas of a window whose appearance and behavior are controlled by the main program). These custom, program controls can be mixed with any number of other controls in the same window, and can be edited like any other control when in edit mode.

Example Projects

The C++ project is in the form of a class hierarchy in which a base class, "VControl" (which contains code that is common to most control drivers), is subclassed to create custom controls: static and editable text controls. The C and Pascal projects are in a procedural format which contains the same code as the C++ "VControl" base class. In this case the default procedures are simply replaced to produce custom controls.

The procedures found in the VControl file have a one-to-one correspondence to the control messages that are sent by ViewIt to control drivers, and each procedure in VControl includes a complete description of what it is that ViewIt expects the procedure to do. Additional information about control drivers can be found in the "Inside Drivers" window when running the "ctlDemo" program which makes use of VControl to support program controls.

FORTRAN Programmers: Although we did not provide examples of FORTRAN-based program controls, note that the form of the program procedures needed to support such controls is the same as the override

procedures described below, so you can use the override example in the FORTRAN version of vDemoXY as the starting point for developing program control drivers.

Procedure Format

Program-based controls are supported by main program procedures that do all of the things done by "control drivers" (first described in "Views" topic). These procedures must be of type "Pascal", and have a single, 4-byte parameter. When working within a code resource, this parameter is used to recover the global fRec address and other information. When working within a main program, however, the fRec is usually available as a global record and so the passed parameter can be ignored:

```
/* C, C++ */
 pascal void function MyDriver(Ptr thePtr);
Pascal
 procedure MyDriver(thePtr : Ptr);
```

where "MyDriver" is the name of the program procedure and "thePtr" is the single, 4-byte parameter.

On entry, the "c" variables in fRec will contain information about the control (as if you called GetCtl), and the variables uCommand and uParam will contain the message being sent to the control by ViewIt. A complete description of the messages sent to controls by ViewIt is presented in the VControl file and Inside Drivers document that are part of the example projects in the "Program Controls" folder.

Procedure Installation

The address of the program procedure must be connected to the program controls in the window. One way of doing this would be to use the OvrCtl command described below, but this would be tedious since OvrCtl would need to be called to connect each such control.

To simplify this task, baseID "7400" has been set aside as a signal to ViewIt that any control with this baseID is a program control (i.e., that it is supported by a program procedure), and that the address of the procedure will be found in the fRec variable fDrvr7400. Thus, to connect all program controls to a single procedure, simply set their baseID to 7400 (in the Control dialog) and fDrvr7400 to the procedure address:

```
 fRec.fDrvr7400 = (long)MyDriver;   /* C, C++ */
 fRec.fDrvr7400 := @MyDriver;         Pascal
```

Note that this assignment should be made before any window is opened that contains associated program controls, and that the procedure must be located in a nonrelocatable code resource.

SELECTIVE OVERRIDES

Occasionally you may find that an existing control almost does what you want it to do, but needs to be modified in some minor way. In this case you should selectively override one or more of its control messages with a program override procedure (instead of creating a program control that must handle all messages from ViewIt). A simple example of an override procedure is presented in the "vDemoXY" program, and the following notes review the installation, format, and operation of these procedures.

Override Format

Override procedures must be of type "Pascal", and have a single, 4-byte parameter. When working in a code resource, this parameter is used to recover the global fRec address and other information. When working in a main program, however, the fRec is usually available as a global record, so the passed parameter can be ignored (although it is used in jumping back to the driver - discussed below):

```
/* C, C++ */
 pascal void function MyOverride(Ptr thePtr);
Pascal
 procedure MyOverride(thePtr : Ptr);
```

where "MyOverride" is the name of the override procedure and "thePtr" is the single, 4-byte parameter.

Override Installation

The most direct method of associating an override procedure with a control is by passing a control handle and a procedure address to the OvrCtl command (Pascal source):

```
 FaceIt(nil,GetCtl,0,0,1,5);
 FaceIt(nil,OvrCtl,ord(cControl),ord(@MyOverride)…
```

where, in this case, the 5th control in the 1st view is being associated with the program's "MyOverride" procedure. See "vDemoXY" for an example using your preferred language and compiler.

The above method of installing an override procedure has two disadvantages: the control must already exist in a 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 OvrCtl in place of the control handle. For example,

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

would associate the override ID "1001" with "MyOverride" by storing this ID/ProcPtr pair in a private table. All controls with an "Override ID" of 1001 (set in Control dialog) will then be associated with the program's "MyOverride" procedure.

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);
```

Since override IDs are saved with control-related resources, 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 control driver directly.

Override Execution

Before ViewIt sends a message to a control, it checks whether the control is linked to an override procedure, and, if so, 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.

On entry, the "c" variables in fRec will contain information about the control (as if you called GetCtl), and the variables uCommand and uParam will contain the message being sent to the control by ViewIt. A complete description of the messages sent to controls by ViewIt is presented in the VControl file and Inside Drivers document that are part of the example projects in the "Program Controls" folder.

In most cases, you'll want to pass messages on to the control driver so that it can execute its default behavior (versus the "program controls" described above that must handle all messages). This is done by calling the "JumpIt" procedure using the 4-byte parameter passed to the override procedure. See the "vDemoXY" programs for examples of calling JumpIt using your preferred language and compiler.

When passing messages on to control drivers, the override procedure must be careful to preserve the contents of the "c" variables in fRec, plus uCommand and uParam since these will be used by the driver. This differs from the procedures that support "program controls" since the latter are drivers and are completely responsible for all messages.

The actions taken by an override procedure usually fall into one of five categories:

• Posting Events

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 ViewIt'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 passing a = 13 to PstEvt puts a new event in the private event queue from which pseudo-menu events are returned to the program's modal or modeless event loop: b -> uMenuID, c -> uResult, and d -> uMenuItem (see the PstEvt command in the "Other Utilities" topic for more info).

If the posted message is to replace the original event, then the code is even simpler:

```
 ...
```

```
 if [some condition] then    message to program
   FaceIt(nil,PstEvt,13,b,c,d)
 else                                message to driver
   JumpIt(thePtr);
 ...
```
ViewIt's "Links" dialog, for example, supports use of the arrow keys to move up and down in its list of controls (versus the default behavior of moving the insertion bar within the current editable text item). This trick is done by overriding all of the editable text controls with the same override procedure:
```
 procedure OverProc(thePtr:Ptr);
 ...
 if (uCommand = 264) then     a key press?
   if (uParam[1] = 30) or (uParam[1] = 31) then
     begin                              arrow keys?
       FaceIt(nil,PstEvt,13,1200,0,uParam[1]);
       exit(OverProc);
     end;
 JumpIt(thePtr);
 ...
```
where the message posted is received by ViewIt from MdlWnd (uMenuID = 1200, uMenuItem = 30 or 31) and interpreted as a request to move up or down in the list.

uMenuID can be thought of as a message "class", 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: The value returned in uMenuItem gets temporarily stored in the 2-byte "modifiers" field of an event record, so don't 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 pass addresses, handles, etc.

• Stealing Messages

Another safe action to take is to simply steal the message and not pass it on to the driver (i.e., do not call "JumpIt"). This strategy works when you wish to replace the default response of the control with a program-specific response, or with no response at all. WARNING: Only the message(s) of interest should be stolen. Stealing other messages (such as a "draw" message) will disable the corresponding default behavior of the control (such as drawing itself!).

• 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 (the 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.

Limitations

- 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 "initCntl" message will not be seen by an override procedure that was installed by passing a control handle to OvrCtl. Any initialization-related code that you wish to add, however, can be executed just after the override procedure is installed.
- Controls that are reinitialized from within editing mode will lose their connection to an override procedure if the procedure was installed by passing a control handle to OvrCtl.
- 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.