

C5. ViewIt Control Commands

ViewIt supports a complete set of commands for managing controls within ViewIt windows.

Name Number Parameters & Variables used

GetWnd -1209 a,b,c,d,wWindow-cString

GetCtl -1211 a,b,c,d,cControl-cString

GetWVC -1208 a,b,c,d,wWindow-cString

ShoCtl -1212 a,b,c,d,cControl-cString

ActCtl -1219 a,b,c,d,cControl-cString

SelCtl -1220 a,b,c,d,cControl-cString

HitCtl -1230 a,b,c,d,cControl-cString

SavCtl -1223 a,b,c,d,cControl-cString

DspCtl -1221 a,b,c,d,cControl-cString

GetVal -1213 a,b,c,d,cControl-cString

SetVal -1214 a,b,c,d,cControl-cString

These commands all make similar use of parameters a, b, c, and d to locate a window and control. a & b are used to designate the window (4 options):

- a = 0, b = control handle (control's "owner" is used)
- or a = 0, b = 0 = top modal or active modeless window
- or a = FWND ID, b = nth such window (0 or 1 = topmost)
- or a = window pointer, b = 0

and c & d designate a control in that window:

- c = view number, d = control number in that view
- or c = 0, d = item ID (= RefCon in Control dialog)
- or c = 0, d = control handle (a & b are ignored)
- or c = 0, d = 0 apply to all (Get/SetVal, Sav/DspCtl)
- or c = 0, d = 0 = skip getting control info (GetWnd)
- or c = 0, d = 0 = get FWND-type handle (GetWVC)

If d is a control handle, then a, b, and c are ignored since the control record itself specifies the owning window. If a window or control cannot be found, then the corresponding wWindow or cControl variable returns zero (nil).

GetCtl updates the contents of the global fRec record with information about the specified control. This information is copied from relocatable control records into the fRec variables cControl (the control handle) to cString. (Note that the cTitle and cString strings are returned as Pascal strings, but can, if necessary, be converted to other string types using the CnvStr utility command.)

GetWnd is identical to GetCtl except that it also returns information about the ViewIt window in the fRec variables wWindow (the window pointer) to wiCount.

GetWVC calls GetWnd (if c & d = 0) or GetCtl (if c or d ≠ 0), and then converts the specified window, view, or control into the corresponding FWND, FVEW, or FCTL-type relocatable block, returning the block's handle in uResult. If uResult is zero (nil), then either the control, view, or window could not be found, or there was not enough memory to create the block.

An FWND handle returned by GetWVC is the handle to the FWND resource associated with the window. The command SavWnd can be used, if desired, to save this updated FWND to disk as a means of saving window settings.

An FVEW or FCTL handle returned by GetWVC is not a resource, and you are responsible for disposing of the block when finished with it (using "DisposHandle"). One use of this form of GetWVC is in the cloning of existing controls or views. In this case GetWVC is used to create an FVEW- or FCTL-type block, and the block's handle is then passed to AddVew or AddCtl to add one or more copies of the view or control to the window.

ShoCtl and ActCtl are just convenient commands that take the place of calling GetCtl followed by a toolbox call that gets passed cControl. ShoCtl calls ShowControl to show the control if the view or control number is positive, and HideControl if the number is negative:

Facelt(nil,ShoCtl,0,0,-1,0); hide first view

Facelt(nil,ShoCtl,0,0,1,-6); hide 6th control

ActCtl calls HiliteControl to either activate the control if the view or control number is positive, or inactivate it if the number is negative. (NOTE: Complex controls such as lists and help text will not be properly act/deactivated if they are not visible or are in a hidden view. This limitation occurs because hidden controls do not receive the drawing message that notifies the driver of a hilite change.)

ShoCtl and ActCtl also support the use of parameters c and d to affect two views at the same time when the first view is being hidden or inactivated. In this case parameter d is interpreted as a second view to show or activate:

Facelt(nil,ShoCtl,0,0,-1,2); hide v1/show v2

which facilitates switching from one view to another.

SelCtl attempts to select the designated control and then, if necessary, scroll it into view. To become selected, the control must be editable, visible, active, and within the active modeless window or topmost modal window. If the window is not active, then the control will become selected when the window is brought to the front. If the control is not editable, then it is simply scrolled into view. Calling SelCtl is equivalent to tabbing to or clicking in an editable control to select it, and results in updating the vSelectCtl, vSelectRec, and vSelectID variables in fRec.

HitCtl simulates a hit in the designated control. The control must be type button, check box, or radio button, but need not be visible or in a visible view or window. Whatever effects a hit by the user would have had on the control will also occur after a call to HitCtl. The control is not, however, auto-scrolled into view if out of view in a scrollable view (use SelCtl to do this).

SavCtl notifies the designated control that it should ask the user if changes made to the control's contents should be saved. If the user does not cancel the operation, or if the control does not support the "Save" message, then uResult returns zero. If uResult ≠ 0, then the program should abort whatever operation led it to call SavCtl. If c = d = 0 is passed, then all controls in the window that are set up to support the "Save" message are notified. SavCtl is usually called before a DspCtl or EndWnd call that involves the disposal of a control whose contents can be saved. Also note that, when quitting, Facelt automatically calls SavCtl for all ViewIt controls that support the "Save" message.

DspCtl disposes of the designated control, removing it completely from memory and redrawing the affected area of the window. If c = d = 0, then all controls in the window are disposed of. IMPORTANT: DspCtl must be used in place of "DisposeControl" and "KillControls" since the latter do not update or dispose of private data maintained by ViewIt.

GetVal is used to update a program's linked data record with values from the window's controls, and SetVal is used to update the window's controls with values from the linked data record. When used to get or set values for one control, these commands also result in updating the fRec variables cControl to cString. See "Data Links" for further info.

WARNING: Do not assume that the content of any "u", "w", or "c" prefixed fRec variables returned by these commands is preserved across calls to the Control Manager or Facelt dispatching procedure. Values that might need to be used again, such as a cControl control handle, should be saved in program variables. One exception: Most utility commands preserve "w" and "c" variables.

AddVew -1200 a,b,c,d,uResult

AddCtl -1216 a,b,c,d,uResult

Uses the FVEW (AddVew) or FCTL (AddCtl) type block designated by parameter a to add b copies of a new view or control, within the window or above the control designated by c, at a screen position offset by d pixels:

a = FVEW (AddVew) or FCTL (AddCtl) resource ID

or a = FVEW or FCTL block handle returned by GetWVC

b = number of new views or controls to create (0 = 1)

(use -b to indicate that new objects should be hidden)

c = object above/within which new object is to be placed

0 = top window (AddVew) and top view (AddCtl)

other = FWND ID, window pointer, or control handle

d = screen offset from top, left of object designated by c

(hi word = pixels across, lo word = pixels down)

If successful, uResult returns with the control handle of the new view or control, otherwise it is set to zero (nil). The following call, for example, would use FCTL 1010 to add one new control to top, left of top view

in top window,

```
Facelt(nil,AddCtl,1010,0,0,0);
```

and return the new control's control handle in uResult. Calling AddView or AddCtl has the same effect as using the standard menu item "Paste" when in ViewIt's editing mode.

When working with a large number of potential views in a window, AddView should be considered as an alternative to hiding/showing views (i.e., DspCtl/AddView would be used to show next view instead of Hid/ShoCtl). AddView has the disadvantage of being a bit slower than a simple ShoCtl, but does not require that all of the views be initialized when the window is first opened.

RESOURCE NOTE: To create a new FVEW or FCTL resource to be later used with AddView or AddCtl, simply copy the corresponding view(s) or control(s) when in editing mode and then use a resource editor to paste it into a resource file. Any number of views or controls can be copied, respectively, to a single FVEW or FCTL resource.

LnkCtl -1210 a,b,c,d

Establishes link used by GetVal/SetVal between program variable and control value (see "Data Links" topic) where,

a = control handle (typically obtained from GetCtl)

b = memory address of program variable

(use b = 0 to "unlink" a control)

c = data type of program variable

d = digits (hi word) and number format (lo word)

(0 = general format, 4 fig.s, & decimal 0s removed)

(pass d = -1 to use digits/format set in Control dialog)

The possible data type values are listed in the "Data Links" topic, and the numbers used to denote digits and format are the same as those used with the NumToS utility command (see "String Utilities" topic).

Parameter d is ignored if the link does not involve a number-to-string conversion.

For example, LnkCtl could be used to link an editable text control at v1c4 to an integer variable "i" by calling,

```
Facelt(nil,GetCtl,0,0,1,4); view 1, control 4
```

```
f := 2 + (3 * 65536); fixed point, 3 decimals
```

```
Facelt(nil,LnkCtl,ord(cControl),ord(@i),2,f);
```

where "@i" is the memory address of variable "i", "2" is the data type of a 2-byte integer, "f" is the combination of digits and format numbers, and "ord" is Pascal's way of converting the control handle and address to long integers.

LnkCtl is particularly useful in cases where variables to be linked are not located within a single record, or when links need to be established with controls that have been dynamically added to a window (using AddCtl). Also note that LnkCtl can be used in combination with the linking of an entire record to a ViewIt window (see "Data Links"), and that it is not necessary to "unlink" a control (b = 0) before linking it to a new variable.

TECHNICAL NOTE: Calling LnkCtl results in resetting the control's private ccDataPtr, ccDataType, ccDataDigits, and ccDataFormat variables. It also resets ccDataOffset to -1 to prevent the control from being later linked to a program record. The alternative approach of setting the control's data linking info in ViewIt's Control dialog, and then passing a record address when calling NewWnd, is simply another way of getting the control's private ccDataPtr, ccDataType, ccDataDigits, and ccDataFormat set, which are the variables used by ViewIt when executing GetVal and SetVal.

OvrCtl -1215 a,b

Installs an override procedure where a is either a control's control handle or a procedure ID number, and b is a main program procedure address. To deinstall an override proc, simply pass 0 in parameter b. See the "Override" topic for further details.

DrwCtl -1217 a,b

Redraws the ViewIt control whose control handle is given by parameter a. Use b = 0 to redraw the entire control (same as toolbox call Draw1Control), or b = 2 to only redraw the visible content area of the control (i.e., without drawing its frame and indent area). Alternatively, you can force all controls in a window to be redrawn by passing a = 0 and b = window identifier (= FWND ID, WindowPtr, or 0 to designate the

active or top modal window).

DrwCtl works best with controls that have solid bodies (are not transparent) since it does not erase the control area before drawing it. For transparent controls (such as transparent static text), it is better to use the toolbox call InvalRect to invalidate the control area (cRect or cClip) so that it is completely erased and redrawn. Data linking can also be used to get transparent text redrawn properly.

After drawing a single control, the "c" variables in fRec will contain information about that control (i.e., same information returned by GetCtl).

WARNING: Do not use DrwCtl in place of the toolbox calls SetCtlValue or HiliteControl (or the ViewIt commands SetVal and ActCtl) since these perform other operations that are necessary when changing the control's value or hilite state.

SizCtl -1225 a,b,c,d

Resizes the ViewIt control whose control handle is given by parameter a. Parameter c is the new horizontal size, and d is the new vertical size (in pixels). This command is similar to the toolbox call "SizeControl", but always works properly with hidden controls, and limits redrawing to the visible window areas needing updating. Pass b = 1 if you wish to have the control immediately redrawn, otherwise the area to be updated is simply invalidated and redrawn on the next DoLoop (modeless) or MdlWnd (modal).

NOTE: After resizing one or more attached controls, call SizWnd with b = c = 0 to force ViewIt to check the final size of all such attached controls.

MovCtl -1226 a,b,c,d,uRect,wEvent

Moves or drags the ViewIt control whose control handle is given by parameter a. Parameter b determines the type of action taken (note that views cannot be dragged):

- b = move or drag mode

- 0 = move control

- 1 = move control and immediately redraw

- 2 = drag image of control and return image rectangle

- 3 = drag image of control and move when mouse released

If the control is being moved (b = 0 or 1), then

- c = new horizontal position (local coordinates)

- d = new vertical position (local coordinates)

If the control is to be dragged (b = 2 or 3), then

- c = constraint mode

- 0 = no constraint on direction of dragging

- 1 = horizontal dragging only

- 2 = vertical dragging only

- d = rectangle within which dragging is to be allowed

- 0 = use parent view's content rect (cContent)

- other = address of rectangle (must be in view's content)

If moving the control (b = 0 or 1), MovCtl is similar to the toolbox call "MoveControl", but always works properly with hidden controls, and limits redrawing to the visible window areas needing updating. Pass b = 1 if you wish to have the control immediately redrawn, otherwise the areas to be updated are simply invalidated and will be redrawn on the next DoLoop (modeless) or MdlWnd (modal). NOTE: After moving one or more attached controls, call SizWnd with b = c = 0 to force ViewIt to check the final position of all such attached controls.

If dragging the control (b = 2 or 3), MovCtl is similar to the toolbox call "DragControl", but is much smarter about drawing the dragged image, restricting dragging to within the parent view, & auto-scrolling within scrollable views. In addition to setting parameters c and d, the starting point of the drag must be passed as global coordinates in the variable wEvent.where. If responding to a single or double click in an enabled control, however, wEvent.where will already contain the global coordinates of the click, so you may not need to directly set this variable. (Recall that "enabled" = checking "Returns On Hit" in the Control dialog.)

An image of the control (rounded if Round field in Bounds dialog is greater than 0) will be dragged within the area of the control's parent view defined by parameter d until the mouse is released. If b = 2, then the final position of the image is returned in uRect, and the program can interpret this in some

program-specific manner (i.e., it may need to slightly adjust the rectangle before calling MovCtl again to actually move the control). If b = 3, then the control is moved to the final position of the dragged image.

ScrCtl -1218 a,b,c,d

Scrolls (or resizes, see below) content area (= cContent) of the ViewIt view or control whose control handle is given by parameter a. Parameter c is the horizontal number of pixels to scroll, and d is the vertical pixels. Positive values of c correspond to scrolling the content to the right, and negative values to the left. Positive values of d correspond to scrolling the content down, and negative values up. No scrolling is done if the content area in the corresponding dimension is less than or equal to the visible content area (= cClip, see "Controls" topic for definitions).

ScrCtl always scrolls the content area in multiples of 8 pixels to minimize the distortion of QuickDraw patterns, and never results in the content being scrolled completely out of view (i.e. you can pass large c or d values to scroll to an edge). Any content area needing updating is also redrawn before returning from ScrCtl.

ScrCtl can also be used to resize the content area of views or controls. To do this, pass b = 1 to inform ViewIt that the size of the content area is being adjusted, and use c and d to define the new horizontal and vertical dimensions (pixels). ViewIt resizes the control's content area, and, if necessary, scrolls the content back into view.

Resizing the content area is usually used by programmers who dynamically add controls to scrollable views. In such a case, for example, the size of the content area of the view control might be adjusted to fit the number and position of daughter controls added.

NOTE: Both hand scrolling and the ScrCtl command take advantage of ViewIt's built-in support for scrolling control contents. This support makes use of a "content" rectangle that can be larger than the control's visible content area. Some controls, however, ignore this scheme or use more complex, private scrolling schemes, and cannot be scrolled with ScrCtl. A good rule to follow is that if the control can be hand scrolled, then it also supports ScrCtl.

StlCtl -1222 a,b,c,d

Resets the text font, size, and style of the ViewIt control whose control handle is given by parameter a. Parameters b, c, and d have the same meaning as those used in the SetFSS utility command (see "Window Utilities" topic). If visible and the font, size, or style changes, then the control is redrawn. If selected and FSSC menus are supported, then these menus are updated to reflect the changes.

ClrCtl -1227 a,b,c,d,uRGB

Resets the designated part color of the ViewIt control whose control handle is given by parameter a. Parameters b, c, and d designate the update mode, part number, and color:

b = update mode

0 = invalidate area needing updating w/o redrawing

1 = immediately redraw area needing updating

c = part number

0 = frame

1 = body (background)

2 = content

d = color

0 = use color in uRGB

other = address of RGB color OR old-style color constant:

33 = black, 30 = white, 205 = red, 341 = green,

409 = blue, 273 = cyan, 137 = magenta, 69 = yellow

If b = 1, the control is visible, & the part color is changed, then the control is redrawn. If the control is selected and FSSC menus are supported, then these menus are updated to reflect the color change.

NOTE: ClrCtl assumes that a color table is associated with the control and that this table has an entry for the part color being modified. To ensure that this is the case, always enter edit mode at least once for such controls and choose a color for the corresponding part (even if black or white) to force ViewIt to allocate a color table for the control. The "Colors" subtopic in the "Controls" topic contains further info about control color tables.

SetVCod -1228 a,b

Resets bits in the VarCode value associated with the control whose control handle is given by parameter a. Parameter b is used to pass one or more bit values that ViewIt uses to set or clear bits in the control's VarCode. Positive values of b set bits, and negative values clear bits.

The BaseCt menu controls at the top of this window, for example, support VarCode options such as,

4 = display menu above control when pressed

64 = don't hilite control when pressed

To set both of these options from within a program, pass $b = 4 + 64 = 68$ when calling SetVCod. To clear both options, pass $b = -(4 + 64) = -68$. ViewIt only sets or clears the bits whose bit values are being passed in b, so other bit flags in the VarCode are not affected.

WARNING: Many VarCode bit flags cannot be changed without reforming the control since they are tied directly to operations performed during initialization of the control. SetVCod should only be used to set or clear bit flags that can be changed without reforming the control. The driver's on-line help may indicate which flags are "pokable".