

## V9. ViewIt Commands

ViewIt commands are executed via the Facelt dispatching procedure found in the file "FaceProcXY" (or in some other form compatible with your programming environment):

```
Facelt(nil,[command],a,b,c,d);   Pascal
Facelt(0L,[command],a,b,c,d);   /* C */
call Facelt(0,[command],a,b,c,d) !Fortran
```

where parameters a, b, c, and d are 4-byte integers, and where other record elements from the global fRec or other records may be used with particular commands.

Each ViewIt command name, its equivalent number, the parameters and record elements used by the command, and a complete description of the command are presented below. Command names can be used in place of the numbers in calls to the Facelt dispatching procedure since they are declared as constants in the file "FaceStorXY" (or in some other way compatible with your programming environment).

### Name Number Parameters & Variables used

#### NewWnd -1201 a,b,c,d,wWindow

Opens a new ViewIt window according to parameters passed:

- a = FWND resource ID (use -ID to keep window hidden)
- b = initial window modality (see "Windows")
  - 0 = modal
  - 1 = modeless
- c = 0 or byte size of linked record (see "Data Links")
- d = 0 or address of linked record (see "Data Links")

and where wWindow returns with the window pointer (or nil if window was not opened). If the FWND is not found, then ViewIt displays an alert and gives you the option of adding the FWND to the current res file at that time.

**COLOR NOTE:** In some cases you may need to know whether an attempt to open a color window was successful or not (i.e., whether the program is being run on a Mac with Color QuickDraw). You can either check for the presence of Color QuickDraw directly (see fEnvFlags notes in "fRec Record" topic), or can examine the "rowBytes" field of the resulting window record (the 2-byte integer found at an offset of 6 bytes into the GrafPort or CGrafPort record). If rowBytes is negative, then the window is a color window.

**DRAWING NOTE:** If you use NewWnd to open one or more modeless windows that you wish to immediately draw into before calling DoLoop, then make a call to DoUpdt2 with d = 8 before doing any drawing so that Facelt (or FaceSt) has an opportunity to update its fActive... variables and can reset the current port to the active window.

#### EndWnd -1202 a,wiHit,wvHit,wcHit,wEvent

Disposes of the ViewIt window defined by parameter a, where a can be either an FWND ID, a window pointer, zero to close the top modal or active modeless window, or -1 to close all open modal ViewIt windows.

If the ViewIt window being closed is a modal window that lies below other modal windows, then all windows above it will also be closed (or made modeless again if they were modeless windows made modal by MdlWnd).

If the window being closed is modal, then the variables wiHit, wvHit, wcHit, wClick, and wEvent are also restored to the values that they held when the window was opened with NewWnd (or made modal with MdlWnd). This feature facilitates the nesting of modal event loops designed around use of the "Hit" variables, but could be confusing in other cases if you forget that EndWnd changes these variables.

#### MdlWnd -1203 a,b

Passes control to ViewIt to perform event handling for, or modify the current mode of, the window designated by a, where a can be either the associated FWND ID, the window's window pointer, or zero to designate the top modal or active modeless window. Parameter b designates the operation:

- 1 = convert modal window back to modeless
- 0 = pass control to ViewIt for modal event handling
- 1 = pass control but return w/o processing events
- 2 = pass control and return after processing events

The latter modal operations (b = 0, -1, or -2) can be applied to both modal and modeless windows. The window is always brought to the front and made visible, and modeless windows are made modal until MdlWnd is called again with b = 1. MdlWnd is usually found between calls to NewWnd and EndWnd. See the "Windows" topic for more info.

If MdlWnd is used to convert a modal window back to its previous modeless state (b = 1), then the variables wiHit, wvHit, wcHit, wClick, & wEvent are also restored to the values that they held when the window was made modal. As with EndWnd, this facilitates the nesting of modal event loops designed around use of the "Hit" variables, but could be confusing in other cases if you forget that MdlWnd changes these variables.

#### SizWnd -1204 a,b,c,d

Resizes the ViewIt window designated by a, where a can be either the associated FWND ID, window's window pointer, or zero to designate top modal or active modeless window. Parameters b and c are the window's new horizontal and vertical dimensions, respectively. The size of the window may be limited by controls or views that are attached to the bottom or right sides of the window.

For modeless windows, parameter d can be optionally used to designate the amount of updating to be done at the time SizWnd is called, where its meaning is the same as that defined for Facelt's DoUpdt2 command.

To force ViewIt to just update the position and size of any attached controls without resizing the window, pass b = c = 0 to SizWnd. This is useful in cases where your program directly adds, removes, moves, or resizes attached controls or views using ViewIt or toolbox commands.

NOTE: Do not attempt to use the toolbox call SizeWindow in place of SizWnd since the former will not readjust the size of controls that are attached to the window's sides.

#### MovWnd -1205 a,b,c,d

Moves the ViewIt window designated by a, where a can be either the associated FWND ID, window's window pointer, or zero to designate top modal or active modeless window. Parameters b, c, & d are the same as those supported by UtilIt's MovRec command.

#### ShoWnd -1206 a,d

#### HidWnd -1207 a,d

Hides (HidWnd) or shows & selects (ShoWnd) the ViewIt window designated by a, where a can be either the FWND ID, the window's window pointer, or zero to designate top modal or active modeless window. For modeless windows, parameter d can be optionally used to designate the amount of updating to be done at the time HidWnd or ShoWnd is called, where its meaning is the same as that defined for Facelt's DoUpdt2 command.

NOTE: Do not attempt to use the toolbox calls HideWindow, ShowWindow, or SelectWindow in place of Hid/ShoWnd since the former will not maintain proper window layering.

#### SavWnd -1224 a

Saves the FWND resource associated with the ViewIt window designated by a, where a can be either the FWND ID, the window's window pointer, or zero to designate top modal or active modeless window. If the window is opened at a fixed position ("Global Coordinates" option is checked in Window dialog), then SavWnd also saves the current position of the window as part of the FWND resource. SavWnd is often used to give users the option of saving window positions as part of saving other program-specific setup info.

NOTE: SavWnd does not call GetWVC to update the FWND in memory before saving it to disk, meaning that changes which your program directly makes to the window contents (via ShoCtl, ActCtl, AddCtl, etc.) will not be saved by SavWnd unless GetWVC is also called.

#### HlpWnd -1229 b,c,d

Opens ViewIt's main help window as a modeless window in a Facelt or FaceSt-based program. If Facelt or FaceSt is not in use, or ViewIt's on-line help and editing resources are not available, then HlpWnd is ignored. Parameters b, c, and d are the same as those supported by UtilIt's MovRec command for positioning the window. The following call, for example, will place the help window on the main screen

with its top, left corner offset 10 pixels from the edge:  
Facelt(nil,HlpWnd,0,0,10,10);

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  
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 the same 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 (4 options):

- c = view number, d = control number in that view
- or c = 0, d = item number (the dth control in window)
- 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 Utillt's CnvStr 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 AddView 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.

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. NOTE: 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 is preserved across calls to the Control Manager or the 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 UtilIt commands preserve "w" & "c" var.s.

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

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

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 AddVew 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, AddVew should be considered as an alternative to hiding/showing views (i.e. DspCtl/AddVew would be used to show next view instead of Hide/ShowControl). AddVew has the disadvantage of being a bit slower than a simple ShowControl, 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 or control when in editing mode and then use ResEdit to paste it into a resource file.

### 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)  
(use d = -1 to preserve current digits/format)

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. Do not use DrwCtl in place of toolbox calls SetCtlValue or HiliteControl since the latter perform additional operations which are necessary when changing the control's value or hilite state.

NOTE: DrwCtl works best with controls that have solid bodies (are not transparent) since it does not first erase the control area before drawing. 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.

### 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 UtilIt's SetFSS command (see "Windows" topic in UtilIt Guide). If visible, the control is redrawn. If selected, the FSSC menus are updated to reflect the style changes.