vtcl

Visual Tcl graphical programming environment

Syntax

vtcl *commandfile*

The following commands are recognized:

Description

SCO Visual Tcl provides a high level graphical scripting language for both character and graphical terminals. The SCO Visual Tcl system interprets the scripting language commands and calls appropriate low level display functions. The graphical output is constructed through Motif, and the character output is rendered through Curses. Using this scripting language allows construction of sophisticated graphical applications without the complexity of directly using Motif or Curses.

SCO Visual Tcl is an extension to the Tcl scripting language. This document covers only the graphical SCO Visual Tcl commands that augment Tcl. See **Tcl**(TCL) and **TclX**(TCL) for a detailed description of the non-graphical scripting elements.

Program components

SCO Visual TCL has three executable binary components:

vtcl The Visual Tcl interpreter that is executed on a per-script basis.

xm_vserver The Visual Tcl server (running as a daemon) that invokes the GUI (Motif) calls.

ch_vserver The Visual Tcl server that invokes the Curses calls.

The **vtcl** process interprets the Visual Tcl script and sends commands over a communications channel (a named pipe) to the appropriate Visual Tcl server. The server interprets the commands received and makes appropriate calls to the underlying display system.

In graphical mode, there is only one **xm_vserver** daemon per display. A single **xm_vserver** process may have several **vtcl** processes sending it commands. The daemon remains active until the user associated with the daemon logs out or the deamon is explicitly killed. Consequently, after the first application is invoked, all subsequent Visual TCL applications can startup faster, since the initialization of the server needs only be performed once. In contrast, in character mode, there is one **ch_vserver** process for each **vtcl** process. This means that with use of the graphical Visual TCL, multiple scripts may be interacting with a single display, but with the character Visual TCL, only a single script may be interacting with the terminal at a given point in time.

Visual Tcl will automatically run in character mode (that is, run the **ch_vserver** process) on character terminals. To have the **vtcl** script run in character mode on graphical displays (for example, **ScoTerm**), the environment variable **CHARM** must be set to TRUE or the environment variable **DISPLAY** needs to be unset.

Concepts

A SCO Visual Tcl program consists of a main section and a number of procedures. The main section is invoked when the program first starts, and puts up display elements on the screen and then goes into a loop waiting for user input. The rest of the procedures are called in response to user input, such as pressing a button, or entering data into a text field. These procedures are referred to as callbacks. Thus, the program is event-driven.

The basic display element in Visual Tcl is a widget. A widget is a display element such as a pushbutton or menu. The dialog widget is a special type of widget that presents a window on the screen and is used as a container for other widgets. A typical Visual Tcl application consists of presenting the user with a number of dialogs and having the user respond to the information.

Each of the widgets in a dialog can be customized with different options. For instance, the label on a pushbutton, the items in a list, the size and position of the widget, fonts, and so on can all be configured.

Each widget is identified by a name. The programmer can specify options for each widget on the command line. These options include geometry options (such as the position of the widget), object class options (associated properties of the object, such as an error callback, font, height, width, and so on), label options (that define labels, accelerator keys and mnemonics), and form or dialog options (applicable only to forms or dialogs). There are also generic additional callback options that are applicable to any widget.

Widgets are defined hierarchically, similarly to the Tk conventions (see *Tcl and the Tk Toolkit*, John K. Outsterhout, Addison-Wesley 1994). Widget names contain the entire widget hierarchy of the branch the widget resides on, to distinguish between identical child widgets located on different branches of the widget tree; for example, ***toplevel.child1.child2.widget***. A Visual Tcl applet consists of a tree of widgets that are managed by the Visual Tcl application. (An applet is a Tcl script that invokes Visual Tcl: an application refers to the Visual Tcl daemon, Tcl interpreter, and the applet script currently being executed.)

The application enters an event loop, displaying a form (or forms), collecting data returned by Visual Tcl, and acting upon the data until the user wishes to leave the application. (This way of structuring a program, whereby user input determines the execution of program statements, is termed "event driven" programming.) The program sets up the form and widgets on it, and executes the form using the **VtMainLoop** command. Execution of subtasks relies on the *callback* mechanism. When an exit condition is reacted, a **VtClose** command is then issued, ending the application's connection to Visual Tcl.

Basic program structure

The general format of a Visual Tcl application is as follows:

```
#
#Define Tcl procedures/callbacks
#
proc one { } { }

proc butCB {cbs} { }

proc textCB {cbs} { }

proc quitCB {cbs} {
    VtClose
    exit 0
}
   .
   .
   .
#
# Start Program
#
set app [VtOpen myprog]
set form [VtFormDialog $app.mainform -ok -okLabel Goodbye \
                                     -okCallback VtClose]
set label [VtLabel $form.label -label "Hello World"]
VtShow $form
VtMainLoop
```

Visual Tcl programs have two main sections:

- defining Tcl procedures/callbacks
- establishing a connection with Visual Tcl

First of all, all Tcl procedures need to be defined. That means all the callbacks and their supporting procedures. These procedures need to be defined first because Tcl is an interpretive language and needs to have its procedures defined before they are called.

Next, the application must establish a connection to the server and put up the first form. This is typically done by making the following four calls:

```
set app [VtOpen Example]
set form [VtForm $app.form]
VtShow $form
VtMainLoop
```

The first call to **VtOpen** establishes the connection with the server. **VtOpen** returns a handle to the application which is used when creating the first form. The next thing to do is to put up a form. This is done by calling **VtForm**. **VtForm** requires a widget name and hierarchy to be passed to it. The format of widget/object names is as follows:

parent_name.form_name

In this example, the parent of the first form is the application shell (that is, the handle returned from **VtOpen**).

To add other widgets to this form, the handle returned from VtForm should be used as their parent. For example, to add a button to this form, the following call can be made:

```
set button [VtPushButton $form.button -label "Push Me"]
```

After putting up the form, the next thing to do is to call **VtShow** to display the form. Typically, **VtShow** is not called until all the widgets on the form have been placed. This is so that the users do not see the form "building" before their eyes. **VtShow** only needs to be called for forms. Widgets within a form are shown by default.

Finally, **VtMainLoop** is called to hand control over to the user and start processing events. Events are application state changes such as pushing a button, or choosing an item in a list. All widgets have events associated with them, which the program may or may not respond to. To respond to an event, a callback procedure is written. This procedure gets called each time the event occurs. For example, a pushbutton widget callback would get called each time the button is pressed.

Every callback procedure gets passed as its last argument a Tcl keyed list of callback data. This list contains minimally, the widget name in which the event occurred (the key being **widget**) and the dialog in which the widget resides (key is **dialog**). Additional information depending on the type of widget and callback may be put in the callback data. For example, the callback data for a list callback would also contain the items in the list which are selected.

A typical callback procedure would be declared like this:

```
proc buttonCB {cbs} {
    set button [keylget cbs widget]
}
```

Alternatively, if the callback had additional arguments it would declared like this:

```
proc buttonCB {arg1 arg2 arg3 cbs} {
    set button [keylget cbs widget]
}
```

Resource files

Visual Tcl reads the standard SCO and OSF/Motif resource files during startup. See the *Graphical Environment Configuration Guide* for details. In addition, corresponding resource files are used by the character mode environment.

When the graphical server process starts up, it reads the resources in */usr/lib/X11/app-defaults/WidgetServer*. Then it checks **xrdb**(X), and finally **$HOME**/*.Xdefaults-**hostname***. (If **scosession** is running (that is, if you are using the desktop) then **xrdb** is already loaded with the resources found in

*/usr/lib/X11/sco/startup/** and **$HOME**/*.odtpref/ScoSession/**display**:0.0*)

When the character server starts, it reads resources from */usr/lib/charm/app-defaults/WidgetServer* and from **$HOME**/*.Cdefaults*.

To specify resources for both the graphical and character servers, use the classname **WidgetServer**. For example, to specify a foreground of gold and a background of blue, say:

```
WidgetServer*foreground: gold
WidgetServer*background: blue
```

To specify resources for an individual **vtcl** script, use the name passed to **VtOpen** as the application's classname. For example, the following script puts up a dialog with the label "Hello World!" and an "ok" button:

```
set app [VtOpen myprog]
set form [VtStartForm $app.form -ok]
set lab [VtLabel $form.label -label "HelloWorld"]
VtShow $form
VtMainLoop
```

To specify that the ok button should turn green when depressed (armed) for this script, the following line should be added to the resource file:

```
*myprog*armColor:  green
```

NOTE Since the graphical server is a daemon and only reads resources at startup, it must be killed and restarted for new resources to take effect. This can be done using **kill**(C) or **VtQuitServer**.

Character mode and graphical variants

Visual Tcl applications are intended to run equally well in character mode or in the graphical environment. However, the screen size available to an application varies with the type of display, and some programs may be forced to use different visual interfaces.

NOTE All option names can be prefixed with either CHARM_ or MOTIF_. If an option begins with CHARM_ it will only be sent to the character mode server. If it is prepended with MOTIF_ it will only go to the Motif-based (graphical) server. These prefixes are typically used when specifying offsets, since the graphical server needs pixels and the character mode server needs characters as measurements. If neither prefix is used, then the option and its value are sent to both servers.

Generic Visual Tcl Options

The following is a list of generic widget options that can be assigned to any widgets created by Visual Tcl. All widgets have Geometry options and Object class options (used for specifying their positioning and their basic attributes). Some widgets also have the standard Dialog class options, Form class options, or Label class options. (For obvious reasons, these are not universally applicable to widgets; for example, a radio button does not need the global Dialog or Form options because it is not a dialog box or a form.)

Some common terminology is used:

cmd The name of a Tcl command or procedure associated with some widget. (For example, if the **-callback** ***cmd*** option is set for a given widget, it specifies that the procedure ***cmd*** is to be executed as a callback by that widget.)

object_name
The widget hierarchy of an object. Used when creating the object; for example, to create ***name*** as a child of ***parent***, use the ***object_name parent.name***.

widgetName
The name returned when creating an object. This name is used when doing commands such as Get, Set, Select, and Delete on widgets. It is also used when specifying a widget hierarchy for a new object.

The form of a widget name is defined as a path through the widget tree. As commands are sent to the server, widgets are created; then further child widgets are added. For example, a form may have a dialog as its child, and the dialog may have a text box and some buttons as children. The dialog will therefore have a name of the form: ***formname.dialogname***; and the buttons will be named ***formname.dialogname.buttonname***.

Note that all widget options are followed by the access attributes. These are as follows:

Attribute	Meaning
C	Indicates that the option can be set at widget creation time.
S	Indicates that the option can be set at any time after the widget has been created, using **VtSetValues**.
G	Indicates that the options can be retrieved by **VtGetValues**.

For example, an option that only has the "C" attribute must be set at widget creation time or not at all. An option with all attributes set may be applied to a widget at creation time, or at any subsequent time (using **VtSetValues**), and it is possible to retrieve the option using **VtGetValues**.

Geometry class options

The following options govern the positioning of widgets.

Description Geometry class options

Options

-above *widgetName* (CS)
Used for widget placement. Puts the object above ***widgetName***. For example to create two pushbuttons with button

“ b ” above button “ a ”:

```
set a [VtPushButton $dlog.a -label "button a" \
    -topSide NONE -bottomSide FORM]
set b [VtPushButton $dlog.b -label "button b"  \
    -topSide FORM -above $a]
```

Be careful not to put widgets in negative space. In the example above, button “ a ” is explicitly attached to the bottom of the form. If it were not button “ b ” would exist outside the viewing form area.

For example, to create two PushButtons, with button b above button a:

```
set a [VtPushButton $fn.a -label "button a" \
        -topSide NONE -bottom FORM]
set b [VtPushButton $fn.b -label "button b" -above $a]
```

Be careful not to put widgets into negative space. In the above example button a is explicitly attached to the bottom of the form. If it were not button b would exist outside the viewing form area.

-alignBottom *widgetName* (CS)
Aligns the bottom side of the source object to the target object.

-alignHorizontal *widgetName* (CS)
Aligns the horizontal axis of the source object to the horizontal axis of the target object.

-alignLeft *widgetName* (CS)
Aligns the left side of source object to the target object.

-alignRight *widgetName* (CS)
Aligns the right side of source object to the target object.

-alignTop *widgetName* (CS)
Aligns the bottom of source object to the target object.

-alignVertical *widgetName* (CS)
Aligns the vertical axis of the source object to the vertical axis of the target object.

-below *widgetName* (CS)
Puts the source object below the target object.

-bottomSide *widgetName* |FORM | ***distance*** | NONE (CS)
Attaches the bottom side of the source object to another object or the FORM, or ***distance*** % of the

height of the form from the bottom.

-bottomOffset *integer*
integer is the Amount to offset the bottomSide attachment.

-leftSide *widgetName* | FORM | ***distance*** | NONE (CS)
Attaches the left side of the source object to the right side of the target object, or to the left side of the FORM, or ***distance*** % of the width of the form from the left.

-leftOffset *integer* (CS)
Sets the left offset of the object.

-rightSide *widgetName* | FORM | ***distance*** | NONE (CS)
Attaches the right side of the source object to the left side of the target object, or to the right side of the FORM, or ***distance*** % of the width of the form from the right.

-rightOffset *integer* (CS)
Sets the offset for the right side of an object.

-topSide *widgetName* | ***FORM*** | ***distance*** | NONE (CS)
Attaches the top side of the source object to the bottom of the target object, or to the top of the ***FORM***, or ***distance*** % of the height of the form from the top.

-topOffset *integer* (CS)
Sets the top offset for an object.

Object class options

Description Object class options.

Options

-allowDuplicateName (C) This allows you to create objects with duplicated names. Use this flag only if you do not intend to reference the object. A typical use of this option could be in a menu routine where references to the separators are not needed. Referencing a object with duplicated names is undefined and not supported.

-autoLock *list of callback procedure names* (C)
Used to lock the server process before calling a callback. The server will lock itself (that is, not accept any more input) before

calling any of the callbacks specified in the list. The lock is exactly the same as calling **VtLock**. The programmer must remember to unlock the application by calling **VtUnLock**.

For example:

```
proc lockCB {cbs} {
    sleep 5
    VtUnLock
}
   .
   .
   .
set ap [VtOpen Lock]
set dlog [VtFormDialog $ap.form ]
VtPushButton $dlog.Lock  -callback \
lockCB -autoLock lockCB
VtShow $dlog
VtMainLoop
```

-baseLineList (G)
: Gets the list of baseLines for a widget. In character mode this always returns 0. The baseline is the distance from the top of the widget to the baseline of the font for the text within the widget.

-borderWidth *integer* (CS)
: Sets the width of the border in pixels. In character mode, if the borderWidth is 0 the border is turned off, if 1 the border is single lined, if > 1 then the border is double lined.

-errorCallback *cmd* (C)
: Sets command ***cmd*** as the handler to be called when an error is thrown in a callback. When an error occurs Visual Tcl will go up the object tree until it finds an errorCallback to handle the error.

Additional callback keys:

widget	Widget which had the callback error
dialog	Dialog which contains the above widget
result	Tcl return string
callback	Callback that caused the return
errorCode	Tcl return code

-font *string* (CS)

Sets the fontList of an object to symbolic font names defined in the Visual Tcl application resource file. Pre-defined symbols are:

```
smallPlainFont
smallBoldFont
smallItalicFont
medPlainFont
medBoldFont
medItalicFont
largePlainFont
largeBoldFont
largeItalicFont
monoNormalFont
monoBoldFont
monoItalicFont
```

For example:

```
VtPushButton $parent.button \
-font largePlainFont
```

-helpCallback *cmd* (C)

Sets the help callback for an object. This callback is called when the ⟨F1⟩ key is pressed over the object. The callback structure returns a keyed list.

Additional Callback list keys are:

widget Widget that received the help callback

dialog Dialog that received the help callback

tree Widget Tree token

-height *integer* (CS)

Sets the height of object in Pixels for the graphical version. In character mode it is in character units. Note that widget must be "managed" before this value can be gotten with **VtGetValues**. Note that if the object has **-rows** as an option, this may be a more convenient way of setting its height.

-hidden *Boolean* (C)

Toggle; set to TRUE the object is not to be displayed after creation, FALSE if the object is to be displayed. All non-dialog objects are by default displayed when created.

-sensitive *Boolean* (CSG)	Set the state of object sensitivity.
-shadowWidth *integer* (CS)	Set the shadow width of an object.
-shortHelpString *string* (CS)	Set the string that is sent to the short help callback
-shortHelpCallback *cmd* (CS)	Set the ***cmd*** to call when the mouse button enters and leaves a widget. Additional Callback list keys: **helpString** shortHelpString set for the widget.
-width *integer* (CSG)	In the graphical environment this sets the width of the widget in pixels. In character mode it sets the width in characters. Note that the widget must be "managed" before this value can be gotten with **VtGetValues**.
-userData *string* (CS)	Sets any string that you want to attach to any widget. (This is typically used to attach comment information to widgets that can be retrieved later using **VtGetValues**.) See also: "VxSetVar" (page 65) "VxGetVar" (page 60)
-xmArgs *list_of_string_pairs* (CS)	This is a "back end" routine for setting Motif resources that have not been implemented. The Motif resources specified with **xmArgs** are passed to the widget creation and manipulation commands, as described in the *Motif Reference Manual*. For example the following changes the background of the button to blue and the foreground to red:

```
VtPushButton $fn.but \
-xmArgs {XmNbackground blue \
XmNforeground blue}
```

See also “Geometry class options” (page 9).

Label class options

Description Label class options.

Options

-accelerator *string* (CS)

Sets the accelerator. An accelerator is a string which describes the button widget’s accelerator. Accelerators are keystroke combinations (usually involving 〈Control〉 or 〈Alt〉 keys to distinguish them from ordinary keystrokes to be sent to the application) that invoke a menu or a menu item even when the menu is not displayed.

The accelerator string format is like that of a translation, but allows only a single key press event to be specified. For example, to specify the accelerator 〈Ctrl〉N, the string **"Ctrl<Key>N"** is used.

NOTE To activate an accelerator in the character server the sequence 〈Ctrl〉A needs to be prepended in the following manner:

Graphical mode	Character mode
〈Ctrl〉S	〈Ctrl〉A〈Ctrl〉S
〈Ctrl〉A	〈Ctrl〉A〈Ctrl〉A
〈Ctrl〉R	〈Ctrl〉A〈Ctrl〉R
〈Alt〉S	〈Ctrl〉A-S
〈Alt〉A	〈Ctrl〉A-A
〈Alt〉R	〈Ctrl〉A-R

-acceleratorText *string* (CS)

Sets the text that is displayed for the accelerator. The string is displayed adjacent to the label string or pixmap. For example the string usually used to specify that ⟨Ctrl⟩N is to be hit is **Ctrl+N**.

-acceleratorString *string* (CS) Sets the string displayed adjacent to the label string or pixmap. Accelerator text for buttons is displayed only for PushButtons and Toggle Buttons in Pulldown and Popup Menus.

-label *string* (CSG) String that is to be displayed in the widget.

-pixmap *filename* (CS) Sets the pixmap to use in a Label or subclass of a Label. If you just specify this without specifying **insensitivePixmap** or **armedPixmap**, those two pixmaps will be automatically generated for you.

-insensitivePixmap *filename* (CS)
Sets the pixmap to use when the object is insensitive. Do not set this if you want it generated for you.

-labelCenter (CS) Centers the label.

-labelLeft (CS) Left aligns the label.

-labelRight (CS) Right aligns the label.

-mnemonic *char* (CS) To make menu functions even more convenient, menus can have mnemonics associated with them. A mnemonic is a letter in a menu button label that can be pressed to activate the button. The first character in the label string that exactly matches the mnemonic is underlined when the button is displayed.

When a mnemonic has been specified, the user activates the button by pressing the mnemonic key while the button is visible. The user can activate the button by pressing either the shifted or unshifted mnemonic key.

NOTE Note that when you specify a mnemonic for a pulldown, both the character and graphical servers automatically add ⟨Alt⟩mnemonic as an accelerator for the menu.

-recomputeSize Boolean (CS) Sets whether the widget shrinks or expands to accommodate its content (label or pixmap). This is only recognized by the Motif version.

Form class options

Description Form class options.

Options

-fractionBase *integer* NA)
Specifies the denominator used in calculating the relative position of an object within a form. The numerator is the "distance" given when specifying options such as **-topSide**, **-rightSide**, **-leftSide**, and **-bottomSide**.

-horizontalSpacing *integer* (CS)
Space between the left and right sides of the objects inside a form.

-marginHeight *int* (CS)
Sets the margin between the top or bottom of the objects inside a form.

-marginWidth *int* (CS)
Sets the margin between the left or right of the objects inside a form.

-verticalSpacing *integer* (CS)
Space between the top and bottoms of objects.

-reset (C)
Sets the reset button on the form.

-resetCallback *cmd* (C)
Sets the callback command ***cmd*** for the reset button.

-resetLabel *string* (C)
Sets the label ***string*** for the reset button.

Dialog class options

Description Dialog class options

Options

-apply (C) Sets an apply button on the form. Also signifies that only the buttons specified with either a button, callback or label reference will be used on the dialog (as opposed to the default buttons for the dialog). (This is not applicable to Message dialogs since there is no apply button)

-applyCallback ***cmd*** (C) Command ***cmd*** to call when apply button is pressed.

-applyLabel ***string*** (CS) Label for the apply button.

-autoDestroy ***Boolean*** (CS)
If ***Boolean*** is **TRUE**, the dialog is automatically destroyed after the user hits either OK or Cancel on the dialog box. The default is **TRUE**. Note that this option is only valid in Selection, File selection, and Message dialogs.

-autoHide ***Boolean*** (CS) If **TRUE**, the dialog is automatically hidden after the user hits either OK or Cancel on the dialog box. The default is **TRUE**. Note that this option is only valid in Selection, File selection, and Message dialogs.

-cancel (C) Sets a cancel button on the form. Also signifies that only the buttons specified with either a button, callback or label reference will be used on the dialog (as opposed to the default buttons for the dialog).

-cancelCallback ***cmd*** (C) Callback ***cmd*** to call when you hit the cancel button in a dialog.

-cancelLabel string (CS) Sets the cancel button label string.

-wmDecoration **ALL** | **RESIZE** | **TITLE** | **BORDER** | **MENU** | **MINIMIZE** | **MAXIMIZE** (C)
Sets the window managers window decoration. This only works in the graphical version running a Motif based window manager. You can combine more than one option; that is, **MINIMIZE** and **MAXIMIZE** only apply on the first dialog created. For example:

```
VtFormDialog $ap.form \
-wmDecoration {TITLE RESIZE MENU}
```

-defaultButton OK | **APPLY** | **RESET** | **CANCEL** | **HELP** | *object_name* (CS)
Sets the default button for a dialog. For dialog boxes that have pre-defined buttons you specify the tokens "OK" or "APPLY", and so on. If you create a form dialog yourself you specify the default button by passing in the *object_name*. This option does not work for FileSelection dialogs.

-help (C)
Sets a help button on the form. Also signifies that only the buttons specified with either a button, callback or label reference will be used on the dialog (as opposed to the default buttons for the dialog).

-helpLabel *string* (C)
Sets the label for the help button.

-modeless (CS)
Sets the dialog to be modeless. This means that the input can go to any window (as opposed to modal behavior, which means that only the topmost dialog can accept input.)

-ok (C)
Sets an ok button on the form. Also signifies that only the buttons specified with either a button, callback or label reference will be used on the dialog (as opposed to the default buttons for the dialog).

-okCallback *cmd* (C)
Sets *cmd* to be the command to call when the ok button in a dialog is pressed.

Selection and FileSelection dialogs will pass the user's selection as part of the callback data (which is a keyed list). The key to reference is "selection".

Additional callback keys:

selection
selection in the selection dialog

-okLabel *string* (CS)
Labels the OK button with *string*.

-tabList *list* (S)
Sets the order that widgets within the dialog will be visited by the ⟨TAB⟩ key. Note that this option only works under character mode; however, its use is recommended because it can be used to make character mode screen interfaces more intuitively navigable.

Visual Tcl Commands

The following commands are recognized by Visual Tcl:

VtAddInput

Syntax **VtAddInput** *fileID cmd*

Description Adds a callback on file activity (read|write|error). Your command *cmd* is called with the fileID appended to it.

VtAddTabGroup

Syntax **VtAddTabGroup** *widgetName*

Description Appends the widget to its parent shell's list of tab groups. This is only operational in character mode.

VtAddTimeOut

Syntax **VtAddTimeOut**

Description Adds a timeout callback. This callback gets called only once. If you want periodic events you must reset the timeout in your callback. This function returns an ID; you will need the ID if you want to remove the timeout with **VtRemoveTimeOut** .

Options

-callback *cmd* (C) Callback to call at end of timeout.

-interval *integer* NA)
Sets the timeout interval to *integer* milliseconds. For example, 1000 = 1 second.

VtAddWorkProc

Syntax **VtAddWorkProc** *cmd*

Description Adds a working procedure. Returns an identifier for the working procedure that is required by **VtRemoveWorkProc** if you subsequently remove it.

A working procedure is called whenever the application is sitting idle. After the working procedure is called, control goes back to the mainloop. If there are still no events in the queue, the working procedure is called again, and is repeatedly called whenever there are no other events happening (such as buttons being hit, text being entered, and so on). If more than one workproc is added via subsequent calls to **VtAddWorkProc**, then they are called one at a time, in a circular fashion.

A common use for working procedures is when the application needs to do some work that the user can cancel out of. The working procedure would do its work in little pieces at a time, keeping track of its state. That way if the user hits cancel, cleanup can be done that is dependent on the value of the state variables.

VtBeep

Syntax **VtBeep** *options*

Description Rings the terminal bell.

Options

-volume ***n***
Sets the maximum volume percentage ***n*** to ring the bell at (as for **XBell( )**; the range is -100 to +100). The default is 50%. (This option is ignored when in character mode).

-duration ***n***
Number of milliseconds to ring bell for (on bitmapped terminals) or 100 times the number of iterations of the bell (on character mode terminals). (For example: **-duration 500** will ring the bell for half a second on a bitmapped terminal, or ding the bell five times in succession on a character terminal.)

-pitch ***n***
Set the pitch in Hz to ring the bell at. (This defaults to the server default pitch. It is ignored when in character mode).

VtCheckBox

Syntax **VtCheckBox** *object_name* [*options*]

Description Creates a checkbox widget. Returns the widget name.

Options

-callback *cmd* (C) Called when a togglebutton in the checkbox is selected. Overrides any individual callback set for the contained toggle buttons.

Additional callback keys:

selectedWidgetList list of selected toggle buttons

-horizontal (CS) Sets the orientation to horizontal.

-numColumns int (CS) Sets the number of columns of objects in a rowcolumn, checkbox, or radiobox.

-spacing *integer* (CS) Sets the spacing between objects inside a row column. Useful for a toolbar.

-value *widgetName* (CS)
Turns on the passed in togglebutton (unsetting all the rest). To set more than one button use **-valueList**.

-valueList *list* (CSG) Takes a list of toggle button widget names and turns those toggle buttons on (unsetting all the rest).

-vertical (CS) Sets the orientation to vertical

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

VtClose

Syntax **VtClose**

Description Closes your connection to Visual Tcl. This command destroys all widgets associated with the current application.

VtComboBox

Syntax **VtComboBox** *object_name* [*options*]

Description Creates a ComboBox widget. Returns the widget name.

Options

-activateCallback *cmd* (C)
Sets the command *cmd* called when an option is selected from the popup list.

-callback *cmd* (C)
Called when an option is selected from the popup list.

Additional callback keys:

value value of combo box

-columns *integer* (CS)
Make the object *integer* columns wide. This means that *integer* characters are shown. In the graphical environment this will make the width of object *integer* * Max Charwidth wide.

-itemList *list* (C)
Sets a list of items for the List object.

-value *string* (CS)
Sets the string to put in the text field in combo box.

-valueChangedCallback ***cmd*** (CS)
Sets the callback ***cmd*** called when the value of an object changes.

Additional callback keys:

value value of the object

See also:

“Object class options” (page 11)

“Geometry class options” (page 9)

VtControl

Syntax **VtControl** suspend | resume

Description This command is used only in the character mode server. It is used to bring the server in and out of curses raw/cooked mode. (In cooked mode, the server preprocesses keyboard input to the application process; in raw mode, all keystrokes are directly transmitted.)

Options

-resume NA)
Resumes the character mode server and sets curses to raw mode.

-suspend (NA)
Suspends the character mode server and sets curses to cooked mode.

VtDestroy

Syntax **VtDestroy** ***widgetName***

Description Destroys an object.

VtDestroyDialog

Syntax **VtDestroyDialog** ***widgetName***

Description Destroys the Dialog given a dialog object. As a convenience you can pass the name of any descendants of the dialog. That way you can destroy the dialog from a callback without walking up the tree yourself. For example:

```
pushButtonCB {cbs} {
        set w [keylget cbs widget]
        # w is a reference to the push button
        VtDestroyDialog $w
   }
```

VtDisplayHelp

Syntax **VtDisplayHelp** *widgetName* -topic [-book]

Description Displays a hard coded topic.

Options

-topic ***string*** (NA)
string specifies the help topic to send to the help server.

-book ***string*** (NA)
string specifies the help book to search for the topic.

VtDrawnList

Syntax **VtDrawnList** *object_name* [*options*]

Description Creates a drawnlist object. Returns the widget name.

Options

-callback ***cmd*** (C)
Sets the callback ***cmd*** to call when you select items in the list.

Additional callback keys:

itemPosition selected item position.

-columns ***integer*** (CS)
Make the object ***integer*** columns wide. This means that ***integer*** characters are shown. In the graphical environment this will make the width of object ***integer*** * MaxCharwidth wide.

-defaultCallback ***cmd*** (C)
Calback to handle double click events.

Additional callback keys:

itemPosition selected item position.

-fieldList ***list*** (C)
Sets a row of data for the drawnlist. The default formatting is used on the list if you do not specify **-formatList** in the command.

-formatList ***list*** (CS)
Describes the columns used in the drawnlist. This field contains a list of column descriptions. Each column description in turn is a list containing the column type, the column width, the column's left and right margins. The syntax is:

```
{ TYPE WIDTH [Left Margin, Right Margin] }
```

The left and right margins are optional parameters. The following specifies a column that is of type ICON, with a width of 1 icon width and a left and right margin of 5 pixels.

```
{ ICON 1 5 5 }
```

Valid types are ICON, STRING, and DATA. DATA does not display on the screen; it is used to store item specific data.

For example:

```
-formatList { {ICON 3} {STRING 20 5} {DATA} }
```

The first columns is an icon field with a width of 3, the second column contains a string with a width of 20 and a left margin of 5, and the third column contains hidden data.

-horizontalScrollBar Boolean (CS)
Set the boolean value to TRUE if you want a horizontal scroll bar, FALSE if not. Default value is FALSE. Ignored in character mode. Maximum size is the width of formatList. If formatList changes, the maximum size will change too.

-iconList ***pixmap_filename_list*** (NA)
Sets a list of pixmap filenames to use in the drawnlist on graphical systems. On character systems, the pixmap filenames are replaced with characters which are displayed on screen in place of the pixmaps. Commands specific to character displays should be prepended with **CHARM_**. For example:

```
-CHARM_iconList {a b c}
-MOTIF_iconList {a.px b.px c.px}
```

(In this example, graphical systems load the pixmaps *a.px*, *b.px*, and *c.px*; character systems load no pixmaps, but display instead the letters “ a ”, “ b ”, and “ c ”.)

-labelFormatList ***list*** (CS)
similar to formatList but applies to the label. (This is not available in character mode)

-labelList ***list*** (CS)
Sets the label above the drawnlist. (This is not available in character mode)

-recordList ***list_of_lists*** (C)
Sets 1 or more rows of data for a drawnlist. For example:

```
{ { 0 1 "field one" "field two"}
  { 1 1 "field one" "field two"}
  { 1 3 "field one" "field two"}
}
```

-rows ***integer*** (CSG)
Sets the number of rows visible in the drawn list to ***integer***.

-selection SINGLE | MULTIPLE (CS)
The selection options are as follows:

SINGLE	(This method is used by default.) A single item is always selected. It is possible to select a different item, but not to deselect an item.
MULTIPLE	An item is always selected. It is possible to drag-select a different item, or to select more than one item by dragging. De-contiguous selection is achieved by repeatedly clicking each item with ⟨Ctrl⟩left-mouse-button; ⟨shift⟩left-mouse button selects all the items between the item under the pointer and the last item selected. The **-callback** is invoked when each item is selected.

-topItemPosition ***integer*** (CSG)
Sets ***integer*** to be the position of the item that is at the top of the list.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

VtDrawnListAddItem

Syntax **VtDrawnListAddItem** ***drawnlist_widgetName*** **[*options*]**

Description Adds an item to the DrawnList. If you do not specify a position **VtDrawnListAddItem** will append it to the end of the list.

Options

-fieldList ***list*** (C)
Sets a row of data for the drawnlist. The default formatting is used on the list if you do not specify **-formatList** in the command.

-formatList ***list*** (CS)
Describes the columns used in the drawnlist. This field contains a list of column descriptions. Each column description in turn is a list containing the column type, the column width, the column's left and right margins. The syntax is:

```
{ TYPE WIDTH [Left Margin, Right Margin] }
```

The left and right margins are optional parameters. The following specifies a column that is of type ICON, with a width of 1 icon width and a left and right margin of 5 pixels:

```
{ ICON 1 5 5 }
```

Valid types are ICON, STRING, and DATA. DATA does not display on the screen; it is used to store item specific data.

Here's an example of a formatList:

```
-formatList { {ICON 3} {STRING 20 5} {DATA} }
```

The first columns is an icon field with a width of 3, the second column contains a string with a width of 20 and a left margin of 5, the third column contains hidden data.

-itemBorder NONE | ONOFFDASH | DOUBLEDASH | SOLID (CS)
Used to set a border around a drawnlist item. This option only works in the graphical environment.

-overrideFont ***string*** (CS)
Sets the font of the drawnlist item to one of the symbolic font names defined in the Visual Tcl application resource file. Pre-define symbols are:

```
smallPlainFont
smallBoldFont
smallItalicFont
medPlainFont
medBoldFont
medItalicFont
largePlainFont
largeBoldFont
largeItalicFont
monoNormalFont
monoBoldFont
monoItalicFont
```

This option only works in the graphical environment.

-position ***integer*** (NA)
Sets the position of the item to to select in a list or a drawnlist. (The base position is 1. To indicate the last item on the list, use 0.)

-recordList ***list_of_lists*** (C)
Sets one or more rows of data.

```
{ { 0 1 "field one" "field two"}
  { 1 1 "field one" "field two"}
  { 1 3 "field one" "field two"}
}
```

VtDrawnListDeleteItem

Syntax **VtDrawnListDeleteItem** *drawnlist_widgetName* [***options***]

Description Deletes an item from a drawnlist. It is an error if the position or the field options do not match something in the list.

Options

-all (NA) Specify entire list.

-field ***column matchStr*** (NA) For the drawnlist you can do a delete on a match of a columns of data. For example, if your list contains:

```
{0 1 "String one"}
{0 1 "String two"}
{0 1 "String three"}
```

specifying -field 2 "String one" would match item 1.

-position ***integer*** (NA) Specify a list item by position. (The base position is 1. To indicate the last item on the list, use 0.)

-positionList ***integer list*** (NA)
Specify a list of items to delete by position.

VtDrawnListDeselectItem

Syntax **VtDrawnListDeselectItem** *drawnlist_widgetName* [***options***]

Description Deselects an item from a drawnlist It is an error if the position or the field options do not match something in the list.

Options

-all (NA)
Deselects the entire list.

-field ***column matchStr*** (NA)
Sets the items to deselect by doing a match on columns of data. For example, if your list contains:

```
{0 1 "String one"}
{0 1 "String two"}
{0 1 "String three"}
```

specifying **-field 2 "String one"** would match item 1.

-position ***integer*** (NA)
Sets the position of an item to deselect.

-positionList ***integer list*** (NA)
Sets a list of positions to deselect.

VtDrawnListGetItem

Syntax **VtDrawnListGetItem** *drawnlist_widgetName* [*options*]

Description Gets the records from the list. Returns a list of lists; a two dimension list of the items in the drawnlist.

```
{
   { 0 {string} {data}}
   { 0 {string} {data}}
}
```

Options

-all (NA)
Gets the entire list.

-field *column matchStr* (NA)
Gets fields on the basis of a match on columns of data. For example, if your list contains:

```
{0 1 "String one"}
{0 1 "String two"}
{0 1 "String three"}
```

specifying **-field 2 "String two"** matches item 2.

-position *integer* (NA)
Sets an item to get by position. (The base position is 1. To indicate the last item on the list, use 0.)

-positionList *integer list* (NA)
Sets a list of item positions to get. If this option is selected, **VtDrawnListGetSelectedItem** returns a list of positions instead of the normal return value.

VtDrawnListGetSelectedItem

Syntax **VtDrawnListGetSelectedItem** *drawnlist_widgetName* [*options*]

Description Returns the selected items in the drawnlist, returns byPosition-List if you do not specify any options.

Options

-byPositionList (NA)
Return items by position list (for example, {1 2 3 4}).

-byRecordList (NA)
For the DrawnList returns a list of lists of fields; for example:

```
{ { 0 1 "field one" "field two"}
  { 1 1 "field one" "field two"}
  { 1 3 "field one" "field two"}
}
```

VtDrawnListSelectItem

Syntax **VtDrawnListSelectItem** ***drawnlist_widgetName*** [***options***]

Description Selects the items in a drawnlist

Options

-all (NA)
Select the entire list.

-field ***column matchStr*** (NA)
Select all items that contain ***matchStr*** in ***column***. For example, if your list contains:

```
{0 1 "String one"}
{0 1 "String two"}
{0 1 "String three"}
```

specifying **-field 2 "String one"** would match item 1.

-next (NA)
Select the next item.

-position ***integer*** (NA)
Select an item by list position. (The base position is 1. To indicate the last item on the list, use 0.)

-positionList ***integer list*** (NA)
Select a list of items by position.

-previous (NA)
Select the previous item.

VtDrawnListSetItem

Syntax **VtDrawnListSetItem** ***drawnlist_widgetName*** [***options***]

Description Replaces the contents of an item in a drawnlist (specified with either **-field** or **-position**) with the new item given in **-fieldList**.

Options

-field column matchStr (NA)
Selects a field to set based on a match of columns of data. For example, if your list contains:

```
{0 1 "String one"}
{0 1 "String two"}
{0 1 "String three"}
```

specifying **-field 2 "String one"** would match item 1.

-fieldList ***list*** (C)
Sets a row of data for the drawnlist. The default formatting is used on the list if you do not specify **-formatList** in the command.

-formatList *list* (CS)
Describes the columns used in the drawnlist. This field contains a list of column descriptions. Each column description in turn is a list containing the column type, the column width, the column's left and right margins. The syntax is:

```
{ TYPE WIDTH [Left Margin, Right Margin] }
```

The left and right margins are optional parameters. The following specifies column that is of type ICON, with a width of 1 icon width and a left and right margin of 5 pixels:

```
{ ICON 1 5 5 }
```

Valid types are ICON, STRING, and DATA. DATA does not display on the screen; it is used to store item specific data.

-position *integer* (NA)
Sets an item by list position. (The base position is 1. To indicate the last item on the list, use 0.)

VtDrawnListSetItemValues

Syntax **VtDrawnListSetItemValues** *drawnlist_widgetName* [*options*]

Description Given an item in a drawnlist (via position or field option), sets the options passed in. Currently, **-overrideFont** and **-itemBorder** are the only options available to set. These options only work in the graphical environment.

Options

-field *column matchStr* (NA)
Select a match on a column of data in a drawnList to set the value for. For example, if your list contains:

```
{0 1 "String one"}
{0 1 "String two"}
{0 1 "String three"}
```

specifying **-field 2 "String one"** would match item 1.

-itemBorder NONE | ONOFFDASH | DOUBLEDASH | SOLID (CS)
Used to set a border around a drawnlist item. This option only works in the graphical environment.

-overrideFont *string* (CS)
Sets the font of a drawnlist item to one of the symbolic font names defined in the Visual Tcl application resource file. This option only works in the graphical environment.

-position *integer* (NA)
Sets the position of the item to set the value of. (The base position is 1. To indicate the last item on the list, use 0.)

VtErrorDialog

Syntax **VtErrorDialog** *object_name* [*options*]

Description Creates an Message Dialog which has an "error" icon in it. Returns the widget name.

This dialog box comes up with the default pushbuttons OK, Cancel and Help. To override the default buttons, the options **-ok**, **-cancel**, **-help**, and **-apply** can be used to specify which buttons go in the dialog. If anyone of these options is used then it is assumed that only buttons which have been individually specified or referenced via either a label or a callback option will be put in the dialog.

If the user chooses either the Ok or Cancel button, the dialog is automatically popped down and destroyed. To override this default **-autoHide** and **-autoDestroy** options can be used.

Options

-message *string* (CSG)
Sets the text of the message (in ***string***). Use "\n" to separate lines.

See also:

"Dialog class options" (page 17)

"Object class options" (page 11)

"Geometry class options" (page 9)

for further information.

VtFileSelectionDialog

Syntax **VtFileSelectionDialog** *object_name* [*options*]

Description Creates a File Selection dialog which displays a list of directories and files for the current directory. The user can select a file from the list or type in the text area. It also allows the user to switch to other directories. The lists are automatically updated to show the list of files and directories in the current directory. Returns the widget name.

By default, File Selection dialogs have an Ok, Cancel, Help and Filter button. The Filter button is used to switch directories. You can choose whether you want the Ok, Cancel and Help buttons, by using the options **-ok**, **-cancel**, and **-help** (see Message Dialog). The filter button is always present.

If the user chooses either the Ok or Cancel button, the dialog is automatically popped down and destroyed, and the user's selection passed as a parameter to the ok callback. To override this default **-autoHide** and **-autoDestroy** options can be used.

Options

-dirListLabel ***string*** (CS)
Sets the label of the directory list box to ***string***.

-fileListLabel ***string*** (CS)
Sets the label of the file list box to ***string***.

-filter ***string*** (CS)
Sets the pattern used for filtering files (e.g. /etc/default/*).

-filterLabel ***string*** (CS)
Sets the label over the box where the user can type in a filter to ***string***.

-hideDirList (C)
Do not display the directory list.

-hideFilter (C)
Do not display the filter text

-selection ***string*** (CS)
Sets the value of the filename in the filename text widget to ***string***.

-selectionLabel ***string*** (CS)
Sets the label over the box where the user can type in a selection.

See also:

"Dialog class options" (page 17)

"Object class options" (page 11)

"Geometry class options" (page 9)

VtForm

Syntax **VtForm** ***object_name*** [***options***]

Description Creates a form. Returns the widget name.

Options See "Form class options" (page 17).

VtFormDialog

Syntax **VtFormDialog** ***object_name*** [***options***]

Description Creates a FormDialog, using all the standard Form class options. You can specify button across the bottom of the dialog by using the **-ok**, **-okLabel**, **-okCallback**, **-cancel**, and **-help** options. The command returns the name of the form dialog. Buttons are created left to right in the following order:

```
OK APPLY RESET CANCEL HELP
```

Note that when creating a form, it is advisable to make the form the child of another form. This enables the new form to inherit attributes such as fonts from the dialog it was launched from. (If the form's parent is a button, it cannot inherit such attributes.)

Note also that you should use **VtGetValues** to get the widgetName of any of the buttons on the bottom of the form and the option used to get the button. For example, to get the widgetName of the ok button, say:

```
set okButton [VtGetValues -ok]
```

For the cancel button, say:

```
set cancelButton [VtGetValues -cancel]
```

Options

-cancelButton OK | APPLY | RESET | CANCEL | HELP | ***object_name*** (CS)
Sets the cancel button for a dialog. For dialog boxes that have pre-defined buttons you specify the tokens OK, APPLY, and so on. If you create a form dialog yourself you must specify the cancel button by passing in the ***object_name***.

See also:

"Dialog class options" (page 17)

"Form class options" (page 17)

"Object class options" (page 11)

"Geometry class options" (page 9)

VtFrame

Syntax **VtFrame** ***object_name*** [***options***]

Description Creates a Frame widget which places a three dimensional border around a single child. Returns the widget name of the Frame.

The border can have different shadow types in the graphical environment, but in the character mode server it is just a single line.

For example, to create an empty box:

```
set frame [VtFrame $form.frame -shadowType ETCHED_IN]
set rowcol [VtRowColumn $frame.rowcol -label "       "]
```

Options

-shadowType IN | OUT | ETCHED_IN | ETCHED_OUT (CSG)
Sets the shadow type of the frame.

-title string (C) Puts the title at the top of the frame widget. This is not supported in character mode.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

VtGetValues

Syntax **VtGetValues** *widgetName* [*option*]

Description Returns the value of the specified option for the given widget. Only options which have a "G" next to the option name in the command reference can be retrieved with **VtGetValues**. Only one option can be retrieved at a time.

Options (Valid option for the particular widget).

Example

```
set label [VtGetValues $myLabel -label]
```

Returns the label string for the widget **$myLabel**.

See also "VtSetValues" (page 52).

VtHide

Syntax **VtHide** *widgetName*

Description Hide but do not destroy an object. This is used to hide the object for later retrieval. Useful if you need to hide a dialog that contains information which should not be destroyed. Use **VtShow** to redisplay the object.

VtHideDialog

Syntax **VtHideDialog** *widgetName*

Description Same as **VtHide**, but is able to walk up the object tree until it finds a dialog to hide.

VtInfo

Syntax **VtInfo** [*options*]

Description Returns some information about Visual Tcl.

Options

-CHARM (NA) flag, returns 1 if running in character mode, otherwise 0.

-colors (NA) This is a flag; it returns the maximum number of colors that can be displayed on the X-Server. This does not mean that you have that many colors free. The character mode environment always returns 2. (foreground and background).

-displayHeight flag (NA)
Returns the height of the display. In character mode it returns the number rows for the terminal; in X it returns the number of pixels.

-displayWidth flag (NA)
Returns the width of the display. In character mode it returns the number columns for the terminal; for X it returns the number of pixels

-version (NA) Returns the Visual Tcl version.

VtInformationDialog

Syntax **VtInformationDialog** *object_name*

Description Creates an Message Dialog which has an "information" icon in it. Returns the widget name.

Options

-message string (CSG)
Text of the message. Use "\n" to separate lines.

See also "Dialog class options" (page 17).

VtLabel

Syntax **VtLabel** *object_name* [*options*]

Description Create a Label. Returns the widget name.

Options All the Label class options are available.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

"Label class options" (page 15)

VtList

Syntax **VtList** *object_name* [*options*]

Description Creates a List object. (Note that the list object may exist with or without scrollbars; see the **-scrollBar** option for details.)

Returns the List widget name.

Options

-callback ***cmd*** (C)
Sets the callback ***cmd*** to call when you select items in the list.

Additional callback keys:

selectedItemList List of items selected

-columns ***integer*** (CS)
Make the object ***integer*** columns wide. This means that ***integer*** characters are shown. In the graphical environment this will make the width of the object ***integer*** * MaxCharwidth wide. In character mode this makes the object ***integer*** -2 characters wide (because the boundary line characters are included in the width).

-defaultCallback ***cmd*** (C)
Callback for to handle double click events.

Additional callback keys:

selectedItemList List of items selected

-filename string (CS)
Sets a file that contains the list of items. Each line of the file is considered a list item. The full pathname of the file must be specified.

-itemList list (C)
Sets a list of items for the List object.

-selection SINGLE | EXTENDED | MULTIPLE | BROWSE (CS)
The selection methods are as follows:

BROWSE (Chosen by default.) An item in the list is always selected. Users cannot deselect an item; only select a different one. A solid line shows around the last selected item.

SINGLE Simple select and deselect is permitted. Only one item can be selected at a time. A dashed line surrounds the last selected (or deselected) item.

EXTENDED An item is always selected, and surrounded by a solid line. Drag select (pressing the mouse button with the pointer over the desired items) extends the selection to cover additional objects.

Multiple non-contiguous objects can be selected using ⟨Control⟩ mouse-click. ⟨Shift⟩left-mouse-button selects all the items between the current item and the last item selected. The **-callback** is invoked when the mouse button is released.

MULTIPLE Any number of items can be selected or deselected. A dashed line surrounds the last selected or deselected items. Drag selection is not available, but non-contiguous selection is direct (that is, selecting additional items does not deselect previous items). The **-callback** is invoked when each item is selected.

-rows ***integer*** (CS)
Sets the number of character rows displayed in an object to ***integer***.

-scrollBar Boolean (CS)
If this option is set to false, a vertical scrollbar is displayed only when the number of items in the List exceeds the number of visible items. If set to true, a vertical scrollbar is always displayed. The default is false.

Note that to get a horizontal scrollbar to appear, the option **-width** or **-columns** must be set. Otherwise the widget will grow as needed and scrollbars will only appear when the list exceeds the available area.

-topItemPosition ***integer*** (CSG)
Sets the position of the item that is at the top of the list.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

VtListAddItem

Syntax **VtListAddItem** ***list_widgetName*** [***options***]

Description Adds one or more items to a list.

Options

-item *string* (C) Sets an item for the List widget.

-itemList *list* (C)
Sets a list of items for the List widget.

-position *integer* (NA)
Specifies the list position of the new item. (The base position is 1. To indicate the last item on the list, use 0.)

VtListDeleteItem

Syntax **VtListDeleteItem** *list_widgetName* [*options*]

Description

Options

-all (NA) Deletes the entire list.

-item *string* (C) Sets an item to delete.

-itemList *list* (C)
Sets a list of items to delete.

-position *integer* (NA)
Sets the position of an item in the list to delete. (The base position is 1. To indicate the last item on the list, use 0.)

-positionList integer *list* (NA)
Sets a list of positions to delete. (The base position is 1. To indicate the last item on the list, use 0.)

VtListDeselectItem

Syntax **VtListDeselectItem** *list_widgetName* [*options*]

Description Deselects one or more items from a list.

Options

-all (NA) Sets the entire list as deselected.

-item *string* (C) Deselects a list item.

-itemList *list* (C)
Deselects a list of items.

-notify (NA) Calls the **select** callback.

-position *integer* (NA)
Sets the position of an item to deselect. (The base position is 1. To indicate the last item on the list, use 0.)

-positionList ***integer list*** (NA)
Sets a list of positions to deselect.

VtListGetItem

Syntax **VtListGetItem** *list_widgetName* [*options*]

Description Gets one or more items from a list. (The returned value depends on the options to the command.)

Options

-all (NA) Return the entire list.

-position ***integer*** (NA)
Return the list item at offset ***integer***. (The base position is 1. To indicate the last item on the list, use 0.)

-positionList ***integer list*** (NA)
Specifies a list of item positions to return.

VtListGetSelectedItem

Syntax **VtListGetSelectedItem** *list_widgetName* [*options*]

Description Gets the list of selected items from a list. Default, returns byPositionList.

Options

-byItemList (NA)
Specifies that a list of selected items is to be returned. For example:

```
{{item 1} {item 2}}
```

-byPositionList (NA)
Sets the routine to return items by list position; for example:

```
{1 2}
```

VtListSelectItem

Syntax **VtListSelectItem** *list_widgetName* [*options*]

Description Selects one or more items in a list.

Options

-all (NA) Select the entire list.

-item ***string*** (C) Select item ***string*** from the list.

-itemList *list* (C)
Select all items in ***list***.

-notify (NA) Calls the select callback.

-position *integer list* (NA)
Selects the item at position ***integer*** in ***list***. (The base position is 1. To indicate the last item on the list, use 0.)

-positionList *integer list* (NA)
Sets a list of positions to select.

VtListSetItem

Syntax **VtListSetItem** *list_widgetName* [*options*]

Description Sets a list item (indicated by **-item** or **-position**) to the value specified by **-newitem**. To set all items in the list use **-itemlist**.

Options

-item *string* (C) Sets the item to change.

-newItem *string* (NA)
Sets the new value to assign to the list item being changed.

-position *integer* (NA)
Sets the position of an item to change. (The base position is 1. To indicate the last item on the list, use 0.)

-itemList *list* (C)
Sets a list of items to change.

VtLock

Syntax **VtLock**

Description Sets the cursor to the watch cursor and locks the application (that is, the application will no longer accept input from the user). New dialogs that are about to pop up are also automatically locked. To unlock the application, see "VtUnLock" (page 56).

VtLock can be called multiple times. Calling **VtUnLock** will free all of the current locks. If you only wish to unlock one layer of the nested locks, use **VtUnLock -once**.

VtMenuBar

Syntax **VtMenuBar** *object_name* [*options*]

Description Creates a Menu Bar. Returns the widget name.

Options

-spacing *integer* (CS)
Sets the spacing between items in a menu bar.. Useful for creating toolbars.

-helpMenuItemList ON_VERSION | ON_CONTEXT | ON_WINDOW | ON_KEYS | INDEX | TUTORIAL | HELP | (C)
Sets the default help menu in a menu bar.

For example:

```
VtMenuBar $parent.menubar \
-helpMenuItemList {ON_VERSION ON_CONTEXT}
```

Note that help ON_CONTEXT does not work in character mode.

See also:

"Object class options" (page 11)
"Geometry class options" (page 9)

VtMessageDialog

Syntax **VtMessageDialog** *object_name* [*options*]

Description Creates a dialog box which contains a message and the default pushbuttons OK, Cancel and Help. Returns the widget name of the dialog. All the standard Dialog options are also supported. To override the default buttons, the options **-ok**, **-cancel**, and **-help** can be used to specify which buttons go in the dialog. If any one of these options is used then it is assumed that only buttons which have been individually specified or referenced via either a label or a callback option will be put in the dialog. If the user chooses either the Ok or Cancel button, the dialog is automatically popped down and destroyed. To override this default **-autoHide** and **-autoDestroy** options can be used.

For example:

```
set msg [VtMessageDialog $but.msg -message "Hi there!" -ok]
VtShow $msg
```

Options

-message *string* (CSG)
Sets the text of the message. Use "\n" to separate lines.

See also:

"Dialog class options" (page 17)
"Object class options" (page 11)
"Geometry class options" (page 9)

VtOpen

Syntax **VtOpen** *Class* [*Bookname*]

Description Establishes a connection to Visual Tcl. An optional second argument is the bookname for scohelp books. Returns the application name to be used as a parent for the top form.

VtOptionMenu

Syntax **VtOptionMenu** *object_name* [*options*]

Description Creates an Option Menu widget which is a menu system component that lets a user select one of several choices. Option Menus are created like Pulldown menus. First the Option Menu is created and then it is "filled" in with other widgets. (The Label class options are supported, so that the Option Menus can be titled.) Returns the Option Menu widget's name.

For example, to create an Option Menu with the options dog, cat and frog:

```
set menu [VtOptionMenu $form.menu -label "Pick one:"
set but1 [VtPushButton $menu.but1 -label dog]
set but2 [VtPushButton $menu.but2 -label cat]
set but3 [VtPushButton $menu.but3 -label frog]
```

Callbacks can be assigned to the individual widgets which make up the Option Menu, or a callback can be assigned to the Option Menu itself. If this is done, callbacks assigned to the Option Menu children will be overridden and the Option Menu callback will be called if any of the children are activated. The child which caused the callback to be called will be passed as callback data.

Options

-callback *cmd* (C)
Sets callback ***cmd*** as the routine to call when any of the options are activated. This callback overrides any activation callbacks assigned to children of the option menu

Additional Callback data:

selectedWidget The name of the widget that activated this callback

-selectedWidget *object_name* ((SG)
Specifies the name of the widget that is selected in the option menu.

See also:

“Label class options” (page 15)

“Object class options” (page 11)

“Geometry class options” (page 9)

VtPulldown

Syntax **VtPullDown** *object_name*

Description Creates a Pulldown menu. (Label resources can be used to title the menu.)

Options

-cascadeButton (G)
This flag can be used only with **VtGetValues**. It returns the widget name of the pulldown's cascadeButton. Use this widget to set options such as **-font**.

-radioBehavior Boolean (CS)
Specify that the Pulldown should act like a Radio Box.

See also:

“Label class options” (page 15)

“Object class options” (page 11)

“Geometry class options” (page 9)

VtPushButton

Syntax **VtPushButton** *object_name* [*options*]

Description Creates a PushButton. (Label class options are used to label the button.) Returns the PushButton widget name.

Options

-callback *cmd* (C)
Sets the callback to call when you press the button. Additional callback keys:

clickCount Number of multi-clicks pressed inside a button. This is used to detect double or triple click events inside a pushbutton.

-armedPixmap *pixmap file* (CS)
Sets the pixmap file to use for the armed pixmap. The armed pixmap is displayed when

you press a button.

See also:

"Label class options" (page 15)

"Object class options" (page 11)

"Geometry class options" (page 9)

VtQuestionDialog

Syntax **VtQuestionDialog** *object_name* [*options*]

Description Creates a Message Dialog which has a question icon in it. (All the Dialog options are available to set up the dialog box.) Returns the widget name.

This dialog box comes up with the default pushbuttons OK, Cancel and Help. To override the default buttons, **-ok**, **-cancel**, and **-help** can be used to specify which buttons go in the dialog. If any one of these options is used then it is assumed that only buttons which have been individually specified or referenced via either a label or a callback option will be put in the dialog.

If the user chooses either the Ok or Cancel button, the dialog is automatically popped down and destroyed. To override this default **-autoHide** and **-autoDestroy** options can be used.

For example:

```
set msg [VtQuestionDialog $but.msg -message "What?"]
VtShow $msg
```

Options

-message *string* (CSG)
Sets the text of the message. Use "\n" to separate lines.

See also:

"Dialog class options" (page 17)

"Object class options" (page 11)

"Geometry class options" (page 9)

VtQuitServer

Syntax **VtQuitServer**

Description This command kills all Visual Tcl applications connected to the **vserver** daemon, then kills the daemon. (This command should not be used to routinely exit an application, because all applications currently using the server will simultaneously die.) It returns nothing.

VtRadioBox

Syntax **VtRadioBox** *object_name* [*options*]

Description Creates a RadioBox. Returns the widget name.

Options

-callback *cmd* (C)
Sets the *cmd* called when a togglebutton in the radiobox is selected. This overrides any callback set for the contained toggle buttons. Additional callback keys:

selectedWidget selected toggle button.

-horizontal (CS) Sets the orientation to horizontal.

-numColumns *integer* (CS)
Sets the number of columns of objects to *integer*.

-spacing *integer* (CS)
Sets the spacing between objects inside a row column.

-value *widgetName* (CSG)
Turns on the widgetName passed in togglebutton.

-vertical (CS) Sets the orientation to vertical.

See also:

“Object class options” (page 11)

“Geometry class options” (page 9)

VtRaiseDialog

Syntax **VtRaiseDialog** *object_name*

Description Raises the Dialog above all others. This works in graphical mode; it is ignored in character mode.

VtRemoveAllTabGroups

Syntax **VtRemoveAllTabGroups** *widgetName*

Description Removes all widgets from the parent shell’s tab group list.

See also “VtAddTabGroup” (page 20).

VtRemoveInput

Syntax **VtRemoveInput** ***fileID***

Description Removes the add input command for ***fileID***.

See also "VtAddInput" (page 20).

VtRemoveTabGroup

Syntax **VtRemoveTabGroup** ***widgetName***

Description Removes the widget from its parent shell's list of tab groups.

VtRemoveTimeOut

Syntax **VtRemoveTimeOut** ***time_out_ID***

Description Removes the timeout callback ***time_out_ID***.

See also "VtAddTimeOut" (page 20).

VtRemoveWorkProc

Syntax **VtRemoveWorkProc** ***workProcID***

Description Removes the working procedure ***workProcID***.

See also "VtAddWorkProc" (page 20).

VtRowColumn

Syntax **VtRowColumn** ***object_name*** [***options***]

Description Creates a RowColumn widget. Returns the widget name.

Options

-horizontal (CS) Sets the orientation to horizontal.

-numColumns *integer* (CS)
Sets the number of columns of objects in a rowcolumn, checkbox, or radiobox to ***integer***.

-packing NONE | COLUMN | TIGHT (CS)
Sets how items contained in a RowColumn widget are packed.

NONE No packing is performed. The x and y attributes of each entry are left alone, and the **column** widget attempts to become large enough to enclose all entries.

COLUMN	All entries are placed in identically sized boxes. The boxes are based on the largest height and width values of all the children widgets. The value of ***numColumns*** determines how many boxes are placed in the major dimension, before extending in the minor dimension.
TIGHT	Given the current orientation (horizontal or vertical) entries are placed one after the other until the **RowColumn** must wrap. A **RowColumn** will wrap when there is no room left for a complete child in that dimension. Wrapping occurs by beginning a new row or column in the next available space. Wrapping continues as often as necessary, until all of the children are laid out. In the vertical dimension (columns), the boxes are set to the same width; in the horizontal dimension (rows) the boxes are set to the same depth. Each entry's position in the major dimension is left unaltered; its position in the minor dimension is set to the same value as the greatest entry in that particular row or column. The position in the minor dimension of any particular row or column is independent of all other rows or columns.

-spacing ***integer*** (CS)
Sets the spacing between objects inside a row column to ***integer***.

-vertical (CS) Sets the orientation to vertical.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

VtScale

Syntax **VtScale** ***object_name*** [***options***]

Description Creates a scale which consists of a bar with a graphical representation that shows the current numerical value of the field. Returns the widget name.

For example:

```
VtScale $form.scale -min 0 -max 100 -value 50
```

Options

-callback cmd (C)
Called when the value of scale is changed. The value of the scale at the time of the callback will be passed as part of the callback data. The key to reference is "value".

Additional callback keys:

value value of the scale

-horizontal (CS) Sets the orientation to horizontal.

-length *integer* (CS)
Sets the length of the scale (specified in number of characters) to ***integer*** characters (in the character environment). In the graphical environment, sets the length of the scale to ***integer*** multiplied by the maximum font width.

-max *integer* (CSG)
Sets the maximum value that a scale can have.

-min *integer* (CSG)
Sets the minimum value that a scale can have.

-readOnly (C) Flag: if set, the slider is disabled.

-showValue *Boolean* (CSG)
When **TRUE**, the **VtScales** value is shown above the scale. The default is TRUE.

-title *string* (C) Sets the title string displayed at the bottom of the scale.

-value *integer* (CSG)
Sets the current value (restricted in range from ***min*** to ***max***.)

-valueChangedCallback *cmd* (CS)
Sets the command ***cmd*** called when the value of an object changes.

Additional callback keys:

value value of the object

-vertical (CS) Sets the orientation to vertical.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

VtSelectionDialog

Syntax **VtSelectionDialog** *object_name* [*options*]

Description Creates a Selection dialog which contains a set of widgets that allows the user to select an item from a list or type the item in a text field. The text field is automatically updated to the item selected from the list. All the Dialog options are available to this widget. Returns the dialog widgets name.

By default, Selection dialogs have an Ok, Cancel, and Help button. To override the default buttons, the options **-ok**, **-cancel**, and **-help** can be used to specify which buttons go in the dialog. If anyone of these options is used then it is assumed that only buttons which have been individually specified or referenced via either a label or a callback option will be put in the dialog.

If the user chooses either the Ok or Cancel button, the dialog is automatically popped down and destroyed, and the user's selection passed as a parameter to the ok callback. To override this default **-autoHide** and **-autoDestroy** options can be used.

For example, to display a Selection dialog with the choices a, b or c:

```
set sb [VtSelectionDialog $but.sb -itemList {{a} {b} {c}}]
VtShow $sb
```

Options

-filename ***string*** (CS)
Sets a file (named ***string***) that contains the list of items. Each line of the file is considered a list item. The full pathname of the file must be specified.

-itemList ***list*** (CS)
Sets the ***list*** of items for the selection box.

-selection ***string*** (CS)
Sets the value of the selection in the selection text widget to ***string***. The item in the list is not highlighted.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

"Dialog class options" (page 17)

VtSeparator

Syntax **VtSeparator** *object_name* [*options*]

Description Creates a Separator widget (a horizontal or vertical line). This is typically used to separate other widgets. Returns the separator widgets name.

For example, to create a separator which goes from left to right side across a form:

```
VtSeparator $form.sep -leftSide FORM -rightSide FORM
```

Options

-horizontal (CS) Sets the orientation to horizontal.

-length ***int*** (CS) Sets the length of the separator (specified in number of characters) to ***int*** in character mode. In graphical mode, set the length of the separator to ***int*** times the maximum font width.

-vertical (CS) Sets the orientation to vertical.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

VtSetAppValues

Syntax **VtSetAppValues** *app_widgetName* [*options*]

Description Set the options for the script (applet). ***app_widgetName*** is the value returned when **VtOpen** was executed to create the application.

Options

-versionString ***string*** (S)
Sets the version string ***string*** for a script. This is displayed when "On version" is picked from the Help menu.

-errorCallback ***cmd*** (C)
Sets command ***cmd*** as the procedure to call when an error is thrown in a callback. When an error occurs, Visual Tcl will go up the object tree until it finds an errorCallback to handle the error.

Additional callback keys:

widget Widget which had the callback error.

dialog Dialog which contain the above widget.

result	Tcl Result string, the error string.
callback	Callback that caused the error.
errorCode	return code from Tcl **eval**.

VtSetFocus

Syntax **VtSetFocus** *widgetName*

Description Sets the focus to an object. If no direction is set the focus will be set on to *widgetName*. If a direction is set, it is interpreted relative to *widgetName*.

Options

-direction *dir* (NA)
where *dir* is one of the following:

DOWN	Move below *widgetName*.
HOME	Go to the first traversable item in *widgetName*'s group.
LEFT	Move to the item to the left of the *widgetName*.
NEXT	Move the next item in the tab group.
NEXT_TAB_GROUP	makes the next tab group the active tab group.
PREVIOUS_TAB_GROUP	make the previous tab group the active group.
RIGHT	Move to the right of the *widgetName*.
UP	Move to the top of *widgetName*.

VtSetSensitive

Syntax **VtSetSensitive** *widgetName* **Boolean**

Description Sets the sensitivity of an object. If **FALSE**, the object is grayed out and the user is unable to traverse to it. The default is **TRUE**.

VtSetValues

Syntax **VtSetValues** *widgetName* [**options**]

Description Used to set the value of a widget option. The option name used is the same option used in widget creation. Only options which have a "S" next to the option name in the command reference can be set with **VtSetValues**.

Options (Valid options for the particular widget. **xmArgs** to pass X resources to the server.)

Example

```
VtSetValues $myLabel -label foo
```

Sets the label string of **$myLabel** to “foo”.

VtShow

Syntax **VtShow** *widgetName*

Description Manages (displays) a previously hidden object.

VtShowDialog

Syntax **VtShowDialog** *widgetName*

Description Same as VtShow, but walks up the object tree to find the object to display.

VtText

Syntax **VtText** *object_name* [*options*]

Description Creates a Text object and returns the widget name. Text fields display text in a box. The text is editable by default, but you can make it read-only. You can create a single line of text, or multiple lines. Text fields are useful for displaying text or for fields where the user needs to enter a value.

Options

-activateCallback *cmd* (C)
Sets the callback to call when a return key is hit in a single line text object, or when the text widget has lost its focus and the contents have changed.

Additional callback keys:

value contents of the text widget.

-callback *cmd* (C)
Sets the callback to call when a return key is hit in a single line text object.

Additional callback keys:

value contents of the text widget.

-columns *integer* (CS)
Make the object ***integer*** number of columns wide. This means that ***integer*** characters are shown. In the graphical environment this will

make the width of the object ***integer*** * Max Charwidth wide.

-filename *string* (CS)
Sets a file (named ***string***) whose contents will be displayed in the text widget.

-horizontalScrollBar Boolean (CS)
Sets whether a horizontal scrollbar will be displayed. Set to TRUE for a horizontal scroll bar, otherwise FALSE. The default value is FALSE. This is ignored in character mode.

-losingFocusCallback *cmd* (C)
Sets the command to call when the text widget loses focus. This routine is called regardless of whether the text has changed.

Additional callback keys:

value value of the text widget.

-noEcho (CS) This is a flag: specifies whether or not echoing should be turned off.

-readOnly (C) This is a flag: when set, disables editing of the text widget.

-rows *integer* (CS)
Sets the number of character rows displayed in an object.

-value *string* (CSG)
Sets the string value of the text.

-valueChangedCallback *cmd* (CS)
Sets the command called when the value of an object changes.

Additional callback keys:

value value of the object.

-verticalScrollBar *Boolean* (CS)
This is a flag: set to TRUE for a vertical scroll bar, otherwise FALSE. The default value is FALSE. This is ignored in character mode.

-wordWrap (C) Sets word wrap on.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

NOTE The following editing keys are available in character mode: Backspace/delete.

⟨Ctrl⟩D Move down a line.

⟨Ctrl⟩E End of line/end of text.

⟨Ctrl⟩I Tab.

⟨Ctrl⟩J New line/return.

⟨Ctrl⟩K Return/Tab.

⟨Ctrl⟩L Move cursor right.

⟨Ctrl⟩M Return.

⟨Ctrl⟩T Start of line/top text.

VtToggleButton

Syntax **VtToggleButton** *object_name* [*options*]

Description Creates a ToggleButton. The button may be labelled using the standard Label class options. Returns the widget name.

Options

-callback *cmd* (C)
Sets the callback ***cmd*** to call when the toggle button is activated.

Additional callback keys:

set state of the toggle button.

-set Boolean (CSG)
Sets the state of a toggle button.

-value Boolean (CSG)
Sets whether the state of the toggle is on or off.

-valueChangedCallback *cmd* (CS)
Sets the ***cmd*** called when the value of the toggle is changed.

Additional callback keys:

set boolean, state of toggle.

See also:

"Object class options" (page 11)

"Geometry class options" (page 9)

"Label class options" (page 15)

VtUnLock

Syntax **VtUnLock** [**-once**]

Description Unlocks the application.

Options

-once
: Used when the application has nested locks (that is, it has been locked multiple times) to unlock one layer of the nested locks as opposed to unlocking all the current locks.

VtWarningDialog

Syntax **VtWarningDialog** *object_name* [*options*]

Description Creates an Message Dialog which has a "warning" icon in it. (The standard Dialog options are used to configure the dialog box.) Returns the widget name.

This dialog box comes up with the default pushbuttons OK, Cancel and Help. To override the default buttons, **-ok**, **-cancel**, and **-help** can be used to specify which buttons go in the dialog. If anyone of these options is used then it is assumed that only buttons which have been individually specified or referenced via either a label or a callback option will be put in the dialog.

If the user chooses either the Ok or Cancel button, the dialog is automatically popped down and destroyed. To override this default **-autoHide** and **-autoDestroy** options can be used.

For example:

```
set msg [VtWarningDialog $but.msg -message "Warning!"]
VtShow $msg
```

Options

-message *string* (CSG)
: Sets the text of the message. Use "\n" to separate lines.

See also:

"Dialog class options" (page 17)

"Object class options" (page 11)

"Geometry class options" (page 9)

VtWorkingDialog

Syntax **VtWorkingDialog** *object_name* [*options*]

Description Creates a Message Dialog which has a "working" icon in it. Returns the widget name.

This dialog box comes up with the default pushbuttons OK, Cancel and Help. To override the default buttons, **-ok**, **-cancel**, and **-help** can be used to specify which buttons go in the dialog. If anyone of these options is used then it is assumed that only buttons which have been individually specified or referenced via either a label or a callback option will be put in the dialog. The standard dialog options are available to this widget.

If the user chooses either the Ok or Cancel button, the dialog is automatically popped down and destroyed. To override this default **-autoHide** and **-autoDestroy** options can be used.

For example:

```
set msg [VtWorkingDialog $but.msg -message "Working..."]
VtShow $msg
```

Options

-message ***string*** (CSG)
Sets the text of the message. Use "\n" to separate lines.

See also:

"Dialog class options" (page 17)

"Object class options" (page 11)

"Geometry class options" (page 9)

Visual Tcl Extended Commands

The following extended commands are recognized by Visual Tcl:

VxAlignBaseLines

Description Given two widgets which have been created with the source widget connected via **-alignTop** to the target widget, this sets the top offset of the source widget so that its baseline lines up with the target widget's baseline.

Syntax **VxAlignBaseLines** ***targetWidget sourceWidget*** [***currentOffset***]

Parameters

targetWidget the widget you are aligning to.

sourceWidget the widget that will be adjusted.

currentOffset any top offset the target widget already has which must be taken into account. (This is optional; the default is 0.)

VxAlignedForm

Description Create one or more vertically aligned widgets within a form, with right justified labels.

Syntax **VxAlignedForm** ***object_name dataList***

Parameters

object_name Widget hierarchy of the form.

dataList A list of lists, each containing a label and another list containing the desired widget call and any desired arguments.

Notes Each Label and Widget are in their own form. Their widget names may be retrieved as follows:

Form **VxGetVar** ***$name*** "form***$n***"

Widget **VxGetVar** ***$name*** "widget***$n***"

Label **VxGetVar** ***$name*** "label***$n***"

Where ***$name*** is the name returned by the **VxAlignedForm** call, and ***$n*** is the position of the widget. The first widget is number 1.

For example:

```
set app [VtOpen "VxAlignedForm"]

set dlog [VtFormDialog $app.Dialog -title "My Aligned form"]

set form [VxAlignedForm $dlog.Align\
  { {"Name:"         {VtText -columns 15 -value "John Doe"}}
    {"Address:"      {VtText -value "123 Hickory Street"}}
    {"Phone Number:" {VtText -value "800-555-1212"}}}]
VtShow $dlog
VtMainLoop
```

Produces a form like this:

```
+------------------------------------+
|        Name:  [John Doe           ] |
|     Address:  [123 Hickory Street ] |
|Phone Number:  [800-555-1212       ] |
+------------------------------------+
```

VxCenterVertically

Description Given two widgets which have been created with the ***sourceWidget*** positioned via **-alignTop** relative to the ***targetWidget***, this sets the top offset of the ***sourceWidget*** so that the ***targetWidget*** is in the center.

Syntax **VxCenterVertically** ***targetWidget sourceWidget***

Parameters

targetWidget the widget you are centering around.

sourceWidget the widget that will be adjusted.

VxCheckBox

Description Creates a **VtCheckBox**, allowing **-title** and **-label**

Syntax **VxCheckBox** *object_name* [**-title** *title*] [**-label** *label*] *VtCheckBoxArgs*

Parameters

object_name Widget hierarchy of CheckBox.

-title Creates a form around the checkbox widget and attaches a label above it. The label is attached to the top side of the form. The checkbox widget is attached to left, right and bottom side of the form and also to the bottom side of the label.

-label Creates a form around the checkbox widget and attaches a label to the left side of it. The label is attached to the top and bottom of the form. The checkbox widget is attached to the right, top and bottom side of the form.

VtCheckBoxArgs
Any argument(s) legal for **VtCheckBox**.

Notes To get the widgetName of the label or form that is created using this command use **VxGetVar**. For example:

```
VxGetVar $widgetName "form"
```

or

```
VxGetVar $widgetName "label"
```

VxComboBox

Description Creates a **VtComboBox**, allowing **-title** and **-label**

Syntax **VxComboBox** *object_name* [**-title** *title*] [**-label** *label*] *VtComboBoxArgs*

Parameters

object_name Widget hierarchy of ComboBox.

-title Creates a form around the combobox widget and attaches a label above it. The label is attached to the top side of the form. The combobox widget is attached to left, right and bottom side of the form and also to the bottom side of the label.

-label Creates a form around the combobox widget and attaches a label to the left side of it. The label is attached to the top and bottom of the form. The combobox widget is attached to the right, top and bottom side of the form.

VtComboBoxArgs
Any argument(s) legal for **VtComboBox**.

Notes To get the widgetName of the label or form that is created using this command use **VxGetVar**. For example:

```
VxGetVar $widgetName "form"
```

or

```
VxGetVar $widgetName "label"
```

VxEndFormCB

Description Destroys the parent form of the widget it was called by. This is useful for calls like:

```
set form [VtFormDialog $app.form -ok -okcallback VxEndForm]
```

In the above example, when the user presses the ok button the dialog is destroyed.

Parameters None.

VxGetShortName

Description Given a widget name, strips off all the parent widgets, leaving the short widget name.

Syntax **VtGetWidgetShortName** *widgetName*

Parameters

widgetName The widget name to strip.

Returns The stripped widget name.

VxGetVar

Description Gets the value of the variable associated with the widget.

Syntax **VxGetVar** *widgetName varName*

Arguments

widgetName name of widget variable is associated with.

varName Name of the variable. May be a scalar or array reference.

See also "VxSetVar" (page 65).

VxList

Description Creates a **VtList**, allowing **-title** and **-label**

Syntax **VxList** *object_name* [**-title** *title*] [**-label** *label*] *VtListArgs*

Parameters

object_name	widget hierarchy of list.
-title	Creates a form around the list widget and attaches a label above it. The label is attached to the top side of the form. The list widget is attached to left, right and bottom side of the form and also to the bottom side of the label.
-label	Creates a form around the list widget and attaches a label to the left side of it. The label is attached to the top and bottom of the form. The list widget is attached to the right, top and bottom side of the form.
VtListArgs	Any argument(s) legal for **VtList**.

Notes Returns the widget name of the list created. To get the widgetName of the label or form that is created using this command use **VxGetVar**. For example:

```
VxGetVar $widgetName "form"
```

or

```
VxGetVar $widgetName "label"
```

VxMenu

Builds a menu, given a menu bar and a list of items.

Syntax **VxMenu** *formDialog menuBar menuList defaultCB*

Description To create a menu bar on a form dialog, you must specify the form dialog to place the menu bar on, the widget name of the menu bar, a list of menu item widgets to place on the menu bar, and a default callback (if no callbacks are specified for the menu items). Menu item widgets are buttons; typically, when a button in a menubar is pressed its callback creates a menu.

Options

formDialog	Sets widgetName of form dialog.
menuBar	Sets widgetName of menubar.
menuList	Sets menu list to build from.
defaultCB	Sets default callback to call if they are not set in the ***menuList***.

Each item in ***menuList*** can contain the following indexes:

0 Type: one of **pd**, **hp**, **cs**, **bt**, or **sp**.

1 Name: name of the button.

2 Mnemonic.

3 Accelerator.

4 acceleratorString of the button.

5 Callback to associate with the button.

6 If the item is a toggle button, this sets the initial state.

For example:

```
set menuList {
    {pd Phone P}
        {bt Add     A "" "" PhoneMenuAddCB    }
        {bt Delete  D "" "" PhoneMenuDeleteCB }
        {sp}
        {bt Exit    E "" "" PhoneMenuExitCB   }

    {pd View V}
        {bt All     A "" "" ""                }
        {bt Friends F "" "" ""                }
        {bt Enemies E "" "" ""                }
        {bt Turkeys T "" "" ""                }
}

set menubar [VtMenuBar $form.menubar]
VxMenu $form $menubar $menuList defaultCB
```

Use **VxMenuGetButton** to get the reference to a menu item widget; for example:

```
VxMenuGetButton $dlog "New"
```

See Also “VxMenuGetButton” (this page).

VxMenuGetButton

Description Gets the widget name of a button in a menu created with **VxMenu**.

Syntax **VxMenuGetButton** *widgetName buttonLabel*

Arguments

widgetName Widget name of the menu created by **VxMenu**.

buttonLabel Label of the button to retrieve.

VxOptionMenu

Description Returns an option list that contains pushbuttons for the options specified.

Syntax **VxOptionMenu** ***object_name label optionList defaultCB selection***

Arguments

object_name	Widget hierarchy of option menu.
label	Menu label (for example, **filename:**).
optionList	Tcl list of options (to be used as labels for the pushbuttons).
defaultCB	Callback to be called when option menu changes.
selection	Option which is initially selected (for example, "one").

VxOptionMenuSetSelected

Description Sets the option menu's value to ***selection*** where ***selection*** refers to the label of the selected pushbutton.

Syntax **VxOptionMenuGetSelected** *widgetName selection*

Arguments

widgetName	name of menu (as returned by **VxOptionMenu**).
selection	label of option to be selected.

VxOptionMenuGetSelected

Description Returns the label of the selected item in an option menu that was created by **VxOptionMenu**.

Syntax **VxOptionMenuGetSelected** *widgetName*

Arguments

widgetName	name of menu returned by VxOptionMenu.

VxOptionMenuReplaceOptions

Description Dynamically changes the options (in an option menu created by **VxOptionMenu**) to the new options passed in.

Syntax **VxOptionMenuReplaceOptions** ***widgetName optionList selection***

Arguments

widgetName	name of menu returned by **VxOptionMenu**.
optionList	tcl list of options (to be used as labels for the pushbuttons).

selection	option which is initially selected (for example, "one").

VxRadioBox

Description Creates a **VtRadioBox**, allowing **-title** and **-label**

Syntax **VxRadioBox** *object_name* [**-title** *title*] [**-label** *label*] *VtRadioBoxArgs*

Parameters

object_name	Widget hierarchy of RadioBox.
-title	Creates a form around the radiobox widget and attaches a label above it. The label is attached to the top side of the form. The radiobox widget is attached to left, right and bottom side of the form and also to the bottom side of the label.
-label	Creates a form around the radiobox widget and attaches a label to the left side of it. The label is attached to the top and bottom of the form. The radiobox widget is attached to the right, top and bottom side of the form.
VtRadioBoxArgs	Any argument(s) legal for **VtRadioBox**.

Notes To get the widgetName of the label or form that is created using this command use **VxGetVar**. For example:

```
VxGetVar $widgetName "form"
```

or

```
VxGetVar $widgetName "label"
```

VxRowColumn

Description Creates a **VtRowColumn**, allowing **-title** and **-label**

Syntax **VxRowColumn** *object_name* [**-title** *title*] [**-label** *label*] *VtRowColumnArgs*

Parameters

object_name	Widget hierarchy of RowColumn.
-title	Creates a form around the rowcolumn widget and attaches a label above it. The label is attached to the top side of the form. The rowcolumn widget is attached to left, right and bottom side of the form and also to the bottom side of the label.

-label Creates a form around the rowcolumn widget and attaches a label to the left side of it. The label is attached to the top and bottom of the form. The rowcolumn widget is attached to the right, top and bottom side of the form.

VtRowColumnArgs
Any argument(s) legal for **VtRowColumn**.

Notes To get the widgetName of the label or form that is created using this command use **VxGetVar**. For example:

```
VxGetVar $widgetName "form"
```

or

```
VxGetVar $widgetName "label"
```

VxSetLeftOffsets

Description Given a list of widgets which have been created with **-alignRight** relative to the previous label, sets the left offset of the first widget so that all the labels fit on the form.

Syntax **VxSetLeftOffsets** *widgets* [*MotifOffset*] [*CHARMOffset*]

Parameters

widgets Sets a list of widgets; the first widget in the list is the one that gets modified

MOTIFOffset Sets the the graphical environment leftOffset possessed by the first widget, which must be taken into account. (This is optional; the default is 0.)

CHARMOffset Sets the character mode **leftOffset** possessed by the first widget, which must be taken into account. (This is optional; the default is 0.)

VxSetVar

Sets the value of a variable and associates that variable with the passed in widget. This is used to store information relevant to the widget (rather than setting a global variable).

Syntax **VxSetVar** *widgetName varName value*

Arguments

widgetName name of widget variable is associated with

varName Name of the variable. May be a scalar or array reference.

value The value assigned to the variable

Returns *Value* is returned.

See also "VxGetVar" (page 60).

VxSpinButton

Syntax **VxSpinButton** *object_name* [*options*]

Description Creates a SpinButton, which consists of a text field and two buttons which increase and decrease the numeric value in the text field within upper and lower bounds.

Parameters

object_name
The widget hierarchy of this SpinButton.

width The width of the Text widget.

lower The lower boundary value.

underCB A callback for when the value goes below the lower bound. If set to "“”, the package automatically wraps the value around to the upper value.

upper The upper boundary value.

overCB A callback for when the value goes over the upper bound. If set to “”, the package automatically wraps the value around to the lower value.

increment Sets how much the value increases or decreases.

default The inital default value.

userCB A callback which checks the value of the Text widget, or “” for no callback.

position Standard *geometryArgs* for the Text widget.

upOp Optional operation to perform on increment when the "up" button is pressed (+ if not present).

dnOp Optional operation to perform on increment when the "down" button is pressed (- if not present).

Globals

SBlower array of lower boundary values, indexed by widget.

SBupper array of upper boundary values, indexed by widget.

Returns The name of the enclosing form widget.

Notes The returned form widget name is attached to the name of the text widget and rowcol widget containing the buttons:

VxGetVar *$sb text* for the text widget.

VxGetVar *$sb rowcol* for the rowcol widget.

VxSpinButtonSetMaxValue

Description Sets the upper boundary for a spin button.

Syntax **VxSpinButtonSetMaxValue** ***widgetName upper***

Parameters

widgetName The instance name of the SpinButton.

upper the maximum value for the Spin Button.

Globals

SBupper array of upper value boundary values, indexed by instance.

VxSpinButtonSetMinValue

Description Sets the lower boundary for a spin button.

Syntax **VxSpinButtonSetMinValue** ***widgetName lower***

Parameters

widgetName The instance name of the SpinButton.

lower the minimum value for the Spin Button.

Globals

SBlower array of lower value boundary values, indexed by instance.

VxText

Description Creates a **VtText** widget, allowing **-title** and **-label** options.

Syntax **VxText** *object_name* [**-title** *title*] [**-label** *label*] *VtTextArgs*

Parameters

-title
Creates a form around the text widget and attaches a label above it. The label is attached to the top side of the form. The text widget is attached to left, right and bottom side of the form and also to the bottom side of the label.

-label
Creates a form around the text widget and attaches a label to the left side of it. The label is attached to the top and bottom of the form. The text widget is attached to the right, top and bottom side of the form.

VtTextArgs
Any argument(s) legal for **VtText**.

Notes — The widget name of the form and the label, if created, are available from **VxGetVar** under the variable names ***form*** and ***label***, respectively.

```
VxGetVar $widgetName "form"
```

or

```
VxGetVar $widgetName "label"
```

VxWidgetVarRef

Description — Return a reference for per-widget variable frames variable. This allows the variable to be passed by reference to other Tcl commands. For example:

```
array names [VxWidgetVarRef my.widget.path data]
```

Syntax — **VxWidgetVarRef** *widgetName varName*

Parameters

widgetName	Name of widget that the variable is associated with.
varName	Name of the variable. May be a scalar or array, but not an element of the array.

Returns — A reference to the variable usable in the current scope.

Files

ch_vserver	the binary for the character mode server
xm_vserver	the binary for the graphical version
vtcl	the binary for the Tcl shell with embedded Visual Tcl command processor

See also

Intro(TCL).

For a discussion of Visual Tcl application style and user interface considerations, see the OSF/Motif *Style Guide*.

Standards conformance

vserver(TCL) is not part of any currently supported standard; it is an extension of AT&T System V provided by The Santa Cruz Operation, Inc.