# Visual TCL

# - vTcl -

## User Guide

- DRAFT -

## Table of contents

## Document scope

The document relates to people who want to use vTcl.

Technical information can be found in the vTcl-Tech-Guide, that explain the technical innards of vTcl for maintainers and developers of the vTcl GUI builder.

This document explains the aspects of using vTtcl components while in an vTcl session.

Nevertheless, a user must know how to deal with TCL/TK code in order to program his functions and to program 'bindings' for the vTcl event handlers.

Teaching TCL/TK is not in the scope of this document.

In order to get information on TCL/TK the relevant information of TCL/TK can/must be referenced on the Internet:

https://www.tcl.tk/

## What is Vtcl??

It is somewhat difficult to explain what Vtcl is without saying some words about its foundation – TCL and the 'windowing extention' to it: TK.

That spells together TCL/TK … spoken: '*tickle-tee-kay*'

The base of Vtcl is TCL (Tool Command Language) and the TK graphical elements library.

TCL is an extremely powerful scripting language invented in the 90th by Dr. John Ousterhout. John Ousterhout followed an unusual but effective way to write this versatile and powerful scripting language.

What did John Ousterhout do so unusal?? Well, he first wrote a book on TCL and then started implementing the language. The result was a product 'as written in the books' …

The outstanding features of TCL are:

- A mighty data type 'list' with powerful list management functions
- 'Associative arrays' with a powerful set of basic functions
- Namespaces (somewhere since TCL8.x)
- 'variable tracing' as a standard language inart

With TCL one can do some really unusual things.

Practically anything can be a variable name – even strings with spaces in it:

set {Grandmother has birthday} {15. June, buy flowers}

is a legal TCL statement that creates a variable '***Grandmother has birthday***' that can be used like any other variable. Suppose what you might do with it ...

```
% set {Grandmother has birthday} {15. June, buy flowers}
15. June, buy flowers
% puts "Reminder: ${Grandmother has birthday}"
Reminder: 15. June, buy flowers
```

Languages like C/C++, BASIC, FORTRAN and Java support arrays in which the index value is an integer. TCL, like most scripting languages (Perl, Python, PHP, etc...) supports associative arrays (also known as "hash tables") in which the index value is a string.

Example from the TCL docs:

```
set name(first) "Mary"
set name(last)  "Poppins"

puts "Full name: $name(first) $name(last)“
```

So your index in 'associative arrays' can be anything - of course even silly numbers … which are still not integers to TCL but an 'identifying ordered code streams'.

TCL is an 'event driven system' – and not just a 'programing language'. So on top of this John Ousterhout put 'variable tracing'; i.e. if a variable changes that creates an event within a running TCL 'system' and this event can be traced by other TCL constructs that have enroled to this event.

So with the instantiation of a TK widget 'messagebox' an attribute 'textvar' is that names a variable to trace. This way the messagebox becomes a 'tracer' for the variable-change-event of this variable. And guess what happens, if you write this variable? Yes, there is nothing else to do to have the text showing up in this very messagebox widget. Other text widgets have this 'textvar' attribute as well.

All these TCL language features made the language prone for the ideas of Steward Allen ...

Visual-TCL – Vtcl for short- was created by Stewart Allen ~1998. The intentions was to create a GUI builder that follows the idea of „***Pure TCL***“; i.e.:

- written in TCL
- programming in TCL
- with a resulting program/product in TCL.

So it is time to explain what Vtcl is NOT.

- Vtcl is NOT just a GUI builder
- Vtcl is NOT just a programing IDE.

# 1 Founders of Vtcl

(Stewart Allen … )

# 2 Features of Vtcl

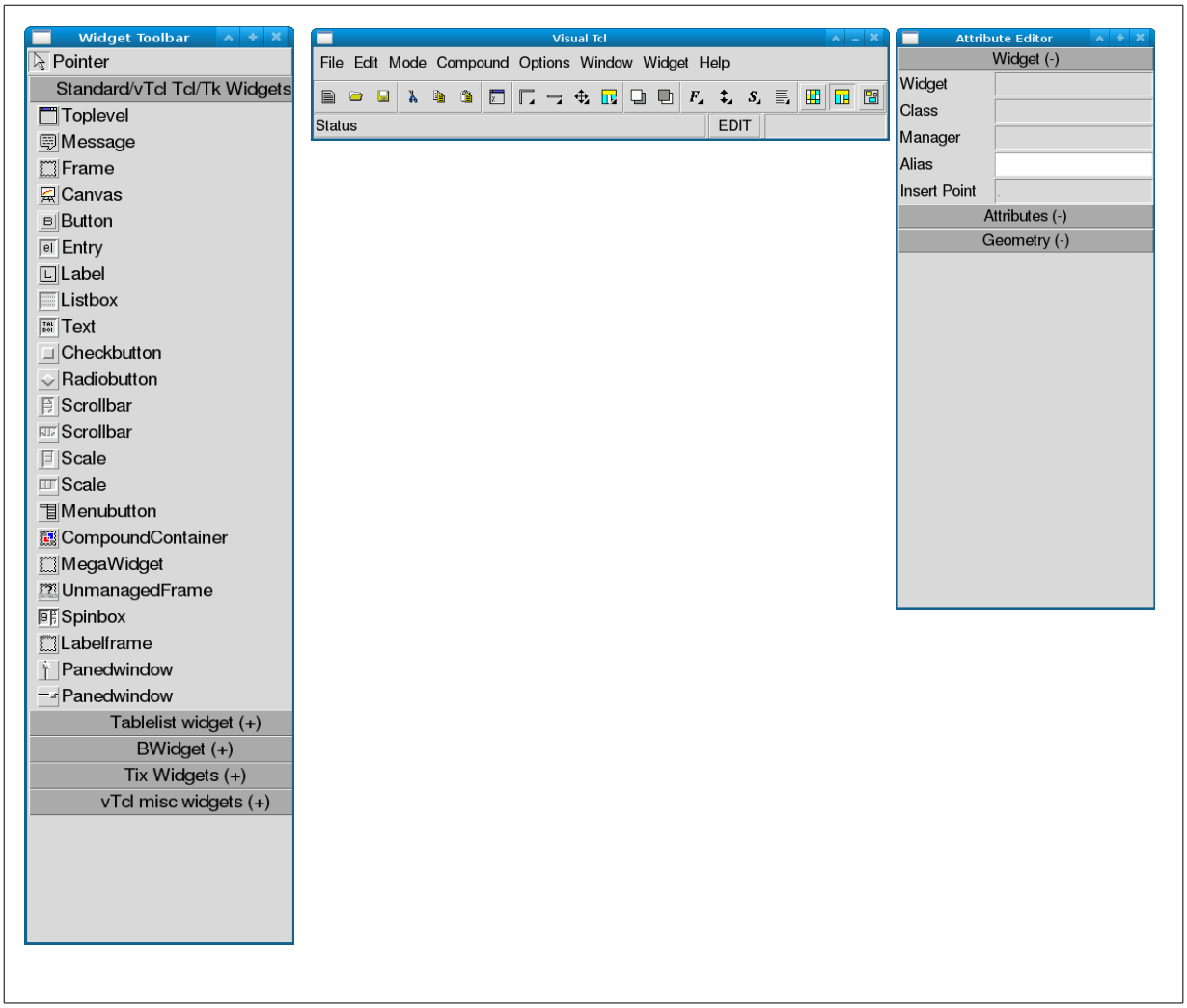
Vtcl is a completely 'selfcontent' system – which does not mean that it cannot be extended from the outside. Like other scripting languages (Perl, Python, …) it can be extended by including code

libraries (import) with a clause 'package require XXXXX'

# 3 Vtcl … the big picture

After starting vTcl at minimum three boxes will open:

- The vTcl main menu
- The vTcl toolbar
- The vTcl attribute manager



At this point you should first create a project by clickin on the left-most icon in the 'Visual Tcl' main window. If you touch the icon a „New project“ bubble pops up. By clicking on the icon the corresponding dialog opens.

The project name you enter here will be a folder below the standard project path that vTcl offers with the default project file 'main.tcl'

Before you press ok, you may change either or both of the offered values for the base path and project file.

Lets stick with the default and just enter the sub folder name for your project which cannot be empty.

After doing so, click the green „OK“ angle in the upper right corner. Taht takes over the values for the later 'save' action and the dialog closes.

Well … still nothing esle heppend so far. How about an 'application' window in order to sketch up your application.

What you need first is a so called 'top level' – but there is none. In order to not getting it into a blind flight, click [Window] in the main menue and choose [Widget tree]. A blank white window pops up and says „Visual Tcl“ with an window icon in front. This is the 'root' of your project … but still there is no 'toplevel' object.

OK. Go to the widget toolbar on the left and open the „Std vtcl … Widget“ folder it is is not already open. Here we are: There is an 'Toplevel' choice thats to be clicked (the icon not the text!) - so just click on it … a little bubble pops up. Grab the lower right corner with your mouse, hold the mousebutton and stretch this 'bubble' to the right.

Well, now the 'bubble' says: „New toplevel 1“ … and watch the „Widget-tree“ window that you have previously opened. It say the „New toplevel 1“ is now a child of your project.

But other things happend as well – just look at the attribute editor on the right. It displays now all the properties that your toplevel window has.

From here on things are pretty straight forward.

Select a „Frame“ from the Widget toolbar on the left and click on your toplevel. A rectangular box will pop up on the toplevel an shortly shows grippers that just vanish. But watch your „Widget-tree“ and „Attribute-Editor“. You now have a „frame“ as a child of your toplevel and the Attribute-Window shows its properties.

Do the same thing again … but take care to drop the next frame on the toplevel – not on the frame you created before. And the same thing for a 3rd time …

You now have 3 boxes stacked from the top towards the bottom on your toplevel window. But this has not yet much to do with an grapical application, since frames actually do nothing. They are used to give your 'screen' an order. So let's go for this now …

Select the topmost frame by clicking on it … and gain watch your „Widget tree“ window and the „Attribute Editor“. Your selected object gets grippers, will be highlited in the „Widget tree“ and its properties show up in the „Attribute Editor“.

Now watch the „Attribute Editor“ for the values in the „Geometry“ section that ist shows for selected box and conentrate on the „expand“, „fill“, and „side“ fields. We now select the box next to „fill“ and a pulldown box opens from which we select 'x'. Watchout now how your frame changed. It filled all the room in x direction that it had – without messing with the other frames.

Now select the 2nd frame from the toplevel window and go to the „Attribute editor“ to change its default values. We now select „expand yes“ and „fill both“. The middle box did exactly this. It first expanded its outside to the available root and then filled up this room in both (x+y) direction.

Now the step with the 3rd frame. Select the frame at the bottem of the stack and change its properties in the „Attribute Editor“. Set its fill direction to 'x' and then change its height property to '32'. At first the frame just fills the room in x direction. But when you change/reduce the height it certainly reduces its height … but the room that the 3rd box gave up will be instantly occupied by the middle box. Well, thats the magic of the 'packer' as the „geometry manager“ that is set up by default, since it is the traditional geometry manager of TK.

Anyway. You now have a verticaly ordered application window with a top, middle and bottom section.

For the next steps we ignore the top and bottom frame an concentrate on the middle frame to gieve it an horizontal order.

So drop 3 frames into the middle frame as you did before with the toplevel. Oh well, The boxes again stacked up vertically – how can you get a horizontal ordering by that?

Again we select the topmost box we just created in side the surroundig middle fram from the previuos steps. The we go to the „Attribute Editor“ and change the „Side“ orientation of the selected frame to 'left'. See what happend ?? The frame align left as the remaing frames stack up.

Again we the topmost (now second) frame and change its 'side' property to 'left'. Well, nothing unexpected happen – the frame just changed its orientation and alinged to the left as it was said to do. Anyway, now the same thing with the remaining top aligned frame. Select, side ptoperty 'left'.

Ok. Nothing changed in the „Widget tree“, since nothing we did changed the widget hierarchy.

So, now we do practically the same with these three boxes as we did with the three outside frames.

But we fill the outer boxes in 'y' direction and set the middle box to „expand yes“ and „fill both“. By this have have the inner frame of the three vertical frames split into 3 horizontal sections.

Before I leave you to the Tutorial, we just drop in some simple active widgets to a little of further possibilities.

Pick a Button from the „Widget toolbar“ and drop it to the leftmost section in the middle line. Do this 3..4 times. You will see that the buttons (all labeled) 'Button' will stack up top to bottom. There is nothing further to do, sinde staked up buttons look pretty well.

Now pick a 'Message' widget and drop it onto the frame at the bottom. Take care not to drop a 'message widget' onto another 'message widget' but the bottom frame.

Dammit! They again stacked up vertically!

Well side orientation 'top' is the default for all widgets on insert – but we know meanwhile that we can change this. So now we do as follows: we bump the first teo from above to the left and the other two to the right side.

You should now have something like this:


To see the magic of the 'packer' geometry manager pick your window with the mouse at a corner and drag the corner atound. Pretty cool, isn't it ?!

We now finish your first steps with some cosmetics and we use the most simple widget at all – the „Label Widget“. Pick one from the „Widget toolbar“ and drop it into the top row. Then change its 'text' property to „My 1st application design“ and make the label 'fill-x'. In order to give yout label some beauty we now use the possibilities in the main menu of vTcl instead of using the „Attribute Editor“.

In the main menu you can find buttons for simple property changes. Move your mouse slowly across the main menu buttons and watch the 'bubbles' until you find 'background'. Thens click on the 'background property' and pick a nice color of your choice. Don't mess with the 'font aspects' in the main menu, since this needs other setup activities to a 'nacked' vTcl installation. You can -as always- change properties in the „Attribute Editor“. See for the 'font' field.

# 4 Working with vTcl

## 4.1 The Alias Mapping

In vTcl you dont have to track the widget path names like:

.top1.frame4.frame3.label8

If vTcl instantiates a widget of a particular class it 'aliases' the widget path and creates an alias in it's „Alias“ namespace. So, if you create a label which is the15th instantiation of a label in your application vTcl takes the widget class 'Label' and generates a new sequence number for the instantiation and generates an alias 'Label15' in it's global 'Alias' array. Along with the Alias it stores the widget path. Thus you do not have to remember the widget path but can use its alias that is shown in the header of the property manager for the widget path.

For the example from above you dont have to reference the label widget by

.top1.frame4.frame3.label8 -text „Customer ID“

You can write:

Label15 -text „Customer ID“

instead.

Aliases can even be overwritten, since they are just names in an associative array.

If you findCustomerIDlabel that 'Label15' isn't descriptive enough for you, then you can overwrite it with 'CustomerIDlabel“ and use it as follows:

CustomerIDlabel -text „Customer ID“

in order to set the label text.

## 4.2 The Property Manager

The checkboxes next to each item in the Attributes section refer to which options will be saved when a project is written out. When a widget is created, the options it has are mostly all defaults. If you change or edit the values of these options from the defaults, the checkbox will automatically check itself to signify that that particular option should be saved with the project.

Note: You can explicity tell vTcl not to save an option by turning off its checkbox.

# 5 Vtcl geometry managers

## *5.1 Packer*

## *5.2 Grid manager*

## *5.3 Placer*