Various widget enhancements and self-explanatory widgets applications of Tcl/Tk GUI

Rahul Dashore,
DFTVisualizer Group, Mentor Graphics Corporation
rahul_dashore@mentor.com

Abstract:
This paper presents a novel method of how enhanced TCL/TK widgets can be used to create self-explanatory widgets for users. Enhanced fancy widgets is useful to create sophisticated and user friendly GUI. However, considering that any GUI would have basic widgets which can be enhanced to fancy widgets, but these widgets are ready to use and can be plugged within any widget.

1. Introduction:

Every GUI has basic widgets which provide bare minimum functionality to build a user friendly environment. DFTVisualizer is a Tcl/Tk based graphical user interface (GUI) to provide debugging environment for EDA tools of Mentor Graphics. It uses basic and modified Tcl/Tk widgets. DFTVisualizer GUI also uses various sophisticated widgets on the top of basic widgets like, MtiWidgets, Text Widgets etc.

As time goes tool requires fancier and user friendly widgets to be long in the market. In addition widgets need to be self-explanatory and easy to use or plug with any GUI specific code.

Following section provides details of various fancy and enhanced widgets. Also describes its uses to plug with existing code and build user friendly environment.

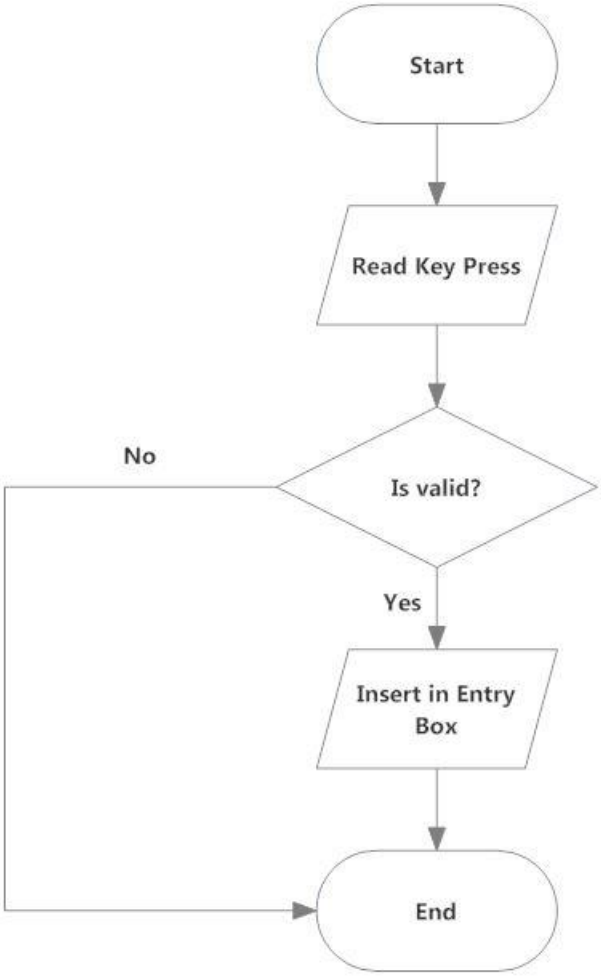
2. Various Enhanced Widgets:

Following widgets are created on the top of existing basic widgets which can be plugged within a frame to create user friendly environment:

2.1 Enhanced entry widgets:

Basic entry widget provides a typing box to user where any input can be typed from the keyboard or paste a text from clipboard. This basic widget takes any charcters/numbers/special-symbols as input, however most of the time GUI developer needs to add more validation criterion on the entered text for its validity.

Various enhanced widgets provide different most-usable validation criteria which developer needs while developing a user interface. Basic flow diagram of functionality is shown below:



1. Flow Diagram of Enhanced Entry Widget

2.1.1 Enhanced integer range entry widget:

As we know basic entry widget takes any character/number/special-symbols as input. Enhanced integer widget validates each keypress and allows entering only integer numeric value.

- Developer syntax to plug enhanced widget:

```
advwidgets::entry .frame_path \
        -labeltext “age” \
        -type “integer”
```

- A range or list of integer values can also be provided:

```
advwidgets::entry .frame_path \
        -labeltext “count” \
        -type “integer” \
        -allowed “0 .. 100”
```

Or

```
advwidgets::entry .frame_path \
        -labeltext “bits” \
        -type “integer” \
        -allowed { 1, 2, 4, 8, 16, 32, 64, 128 }
```

- Allowed type of integers can be specified as below:

```
advwidgets::entry .frame_path \
        -labeltext “all” \
        -type “integer” \
        -negative 1
```

- Enum values can be specified with any enhanced entry widget:

```
advwidgets::entry .frame_path \
        -labeltext “input” \
        -type “integer” \
        -negative 1 \
        -enum { x, auto, on, off }
```

Note: other basic options are also provided with widget such as, editable, foreground, background etc.

Above widgets can be instantiated within a frame which can be plugged with different GUI widgets. Enhanced widgets provide a dropdown option with entry widgets which shows a list of valid values. E.g. if a range of integers are specified and/or enum values are specified advance widget will show following list of values:

Figure 2: Widget to show range of integers and its validation

2.1.2 **Enhanced real numbers entry widget:**

Similar to above entry widget, real numbers entry widget allow floating values. The syntax is shown below:

```
advwidgets::entry .frame_path \
        -labeltext “weight” \
        -type “real” \
        -enum { x, auto, on, off }
```

The widget allow real numbers e.g. 1.2, 3.8, 0.7, -2 etc. Also allow enum as mentioned in the list.

2.1.3 Enhanced binary entry widget:

As self-explanatory by name, binary entry widget is implemented for allowing binary numbers. But it also allow user to specify octal and hex numbers. Allowed base must be specified at the time of instantiation:

```
advwidgets::entry .frame_path \
        -labeltext “weight” \
        -type “binary” \
        -allowed { binary, octal, hexadecimal }
        -enum { x, auto, on, off }
```

When allowed base are octal and hexadecimal, user needs to specify type e.g. o777, h8ade, b1010 etc. If no base is specified it assumes input is in binary form e.g. 111.
User can also specify the number of bits e.g. 4’b1010.

2.1.4 Enhanced time entry widget:

Time entry widget allows non-negative time units which can be specified at the time of instantiation:

```
advwidgets::entry .frame_path \
        -labeltext “time” \
        -type “time” \
        -allowed { s, ms, us, ps } \
        -enum { x, auto, on, off }
```

It allows user to enter only allowed time units as integer values. By default it assumes inputs are in seconds.

2.1.5 Enhanced string entry widget:

Enhanced string widget is similar to basic entry widget except that it provide user list of enum values, so that user can easily select a value from the list:

```
advwidgets::entry .frame_path \
        -labeltext “weight” \
        -type “binary” \
        -allowed enum \
        -default on \
        -enum {auto, on, off }
```

If –allowed enum is specified, value from the list can be specified else user can type any other string. –default will show initial text in the entry box.

Note: All enhanced entry widget has capability to show history, Developer can specify –enablehistory 1. When this flag is on list box widget will show last 10 user specified value (by default only 10 historical values). An option –historycount <count> can be set to increase/decrease to show historical values.

Pseudo code of Enhanced Entry Widgets:

```
itcl::class ::advwidgets::entry {

        #variable list (internal variables used by class)
                #define variables with default values

        #constructor code
                #parses and assign values to corresponding variables
                #constructs entry widget with following components
                        #top frame
                        #label widget
                        #entry widget
                        #dropdown button
                        #popup listbox
                #bindings on keypress, select values from listbox etc

        #validation proc
                #validate based on different types and specified options
}
```

```
#validation proc for different types
::itcl::body ::advwidgets::entry::validate { args } {
        #parse arg

        #if selected from the listbox it's always valid
                #return true

        #set current_entered_text (which is appended current keypress to entrywidget text)

        #validate for enum values for current keypress appended to current text in entrybox
                #set current_entered_length to length of current_enetered_text
                #foreach specified enumvalues
                        #if current_entered_text is matched with first current_entered_legth of enum_value
                                #return true
```

```
#based on type different validation:

#switch type {
        #integer:
                #if allowed_negative and  first keypress is ‘–‘
                        return true
                #if current keypress is not an integer
                        #return false
                #if current_entered_text is not in allowed list or not in range
                        #return false
                #return true


        #real:
                #if keypress is neither an integer nor ‘.’
                        #return false
                #if already a ‘.’ present and keypress is ‘.’
                        #return false
                #return true

        #binary:
                #if [b|o|h] not specified keypress neither 0 nor 1
                        #return false

                #now only for allowed b|o|h binary only is considered above
                #if keypress is integer and allowed is [b|o|h]
                        #return true
                #if ‘ is not present in the entry_box_text and keypress is ‘
                        #set bits_count entry_box_text
                        #return true
                #if ‘ present already and keypress ‘
                        #return false
                #if keypress is not [b|o|h]
                        #retun false
                #else
                        #set base b|o|h
                #if keypress is in range of base (b|o|h) set in the above step
                        #return true
                #else
                        #return false
```

```
                #time:
                        #set time_end 0
                        #if keypress is integer and not time_end
                                #return true
                        #if keypress is in allowed time symbol (s|ms|us|ps)
                                #set time_end 1
                                #return true
                        #return false

                #string:
                        #return true (enum is already considered before)
                }
}
```

Above algorithm handle all different types of symbols enum values, range of integers etc. gracefully. Enhanced entry widgets have capability to color label text and entry widget text. These widgets are useful in real time application where user needs different data type widgets, which can be plugged with any widget.

2.2 Foldable Widget:

In EDA as technology grows the large data handling is required by tools. When data become heavier number of widgets increases to show the data. Most of the time user wants to view/modify subset of large data.

Proposed foldable widget is a technique to group different category data in a single unit of widget. The beauty of this widget is to show/hide a group of widget without impacting other groups. It provides on demand visibility of children widgets.

As shown below DataInPorts group has huge number of widgets which are not visible but grouped and collapsed so that user can concentrate on the other group of widgets:

Figure 3: A Foldable Widget to show/hide children widgets

- Syntax to plug foldable widget:

```
advwidgets::foldablewidget .frame_path \
        -labeltext “DataInPort” \
        -default collapse \
        -foreground “gray”
```

It also provide basic options like –relief, -borderwidth, -state etc. It returns child site to developer to add children widget or nested foldable widgets.

Pseudo code of Enhanced Entry Widgets:

```
itcl::class ::advwidgets::foldablewidget{

    #variable list (internal variables used by class)
        #define variables with default values

    #constructor code
        #parses and assigns values to corresponding variables
        #constructs following components
            #top frame
            #label widget
            #child site frame
        #append [+] or [-] in the label text based on –default option
        #places label at appropriate coordinates (top left corner of widget)
        #if default is collapse
            #set visible 0
        #else
            #set visible 1

        #binding on mouse-click to show/hide children widgets

    #showhide proc to collapse/expand children
}
```

```
::itcl::body ::advwidgets::foldablewidget::showhide { } {
    #if visible is 0
        #modify label_text to label_text [-]
        #pack child site
        #set visible 1
    #else
        #modify label_text to label_text [+]
        #packfoget child site
        #set visible 0
}
```

2.3 Dynamic Row Modification widget:

In some tools input data required variable number of entries. The number of data entries for a subject may vary. For example telephone number of an employee may vary from 1 to many.

A dynamic row modification widget can be used to solve this purpose. It provides on demand addition or deletion of rows. An example of row modification widget is shown below:

Figure 4: Dynamic Widget to add/delete rows

- Syntax to use row modification widget:

```
advwidgets::rowidget .frame_path \
        -labeltext “decoded_values” \
        -default_rows 3 \
        -default_values { 3’b100, “”, “” }
```

User can click on + or – button to add or delete a row, respectively.

Pseudo code of Enhanced Entry Widgets:

```
itcl::class ::advwidgets::rowwidget{

    #variable list (internal variables used by class)
        #define variables with default values

    #constructor code
        #parses and assigns values to corresponding variables
        #constructs following components
            #top frame
            #label widget
            #child frame
            #for 0 to –default_rows
                #create row frame
                #pack + button – button and enhanced entry widget in row frame
                #insert default_value for each row
                # pack row frame in child frame
            #pack label widget and child frame inside top frame
```

```
        #various widget methods
                #addrow
                #deleterow
}
```

```
::itcl::body ::advwidgets:: rowwidget::addrow { current_row_path } {
        #create a new row frame
        #create + button with associated command
        #create – button with associated command
        #created enhanced entry widget
        #pack + button – button and enhanced entry widget
        #pack new row frame below current_row_path

        #adjust indices of each available row (indices required for other functionality)
}
```

```
::itcl::body ::advwidgets:: rowwidget::deleterow { current_row_path } {
        #if number of rows in child frame is 1
                #return (minimum one row is required)

        #destroy current row
        #adjust indices of each available row (indices required for other functionality)
}
```

2.4 A Self-explanatory Enhanced Tree Widget:

An enhanced tree widget has coloring mechanism to show different coloring style to indicate various signals. These signals and coloring style can be modified on demand. In the following example a leaf node contains error which is shown in the red color and all the ancestor nodes will become in orange. This is useful when nodes are collapsed and leaf nodes are not visible. User can easily navigate through orange color nodes to get an erroneous node. As soon as error is resolved leaf color changed red to default and ancestors recolored according to its children.

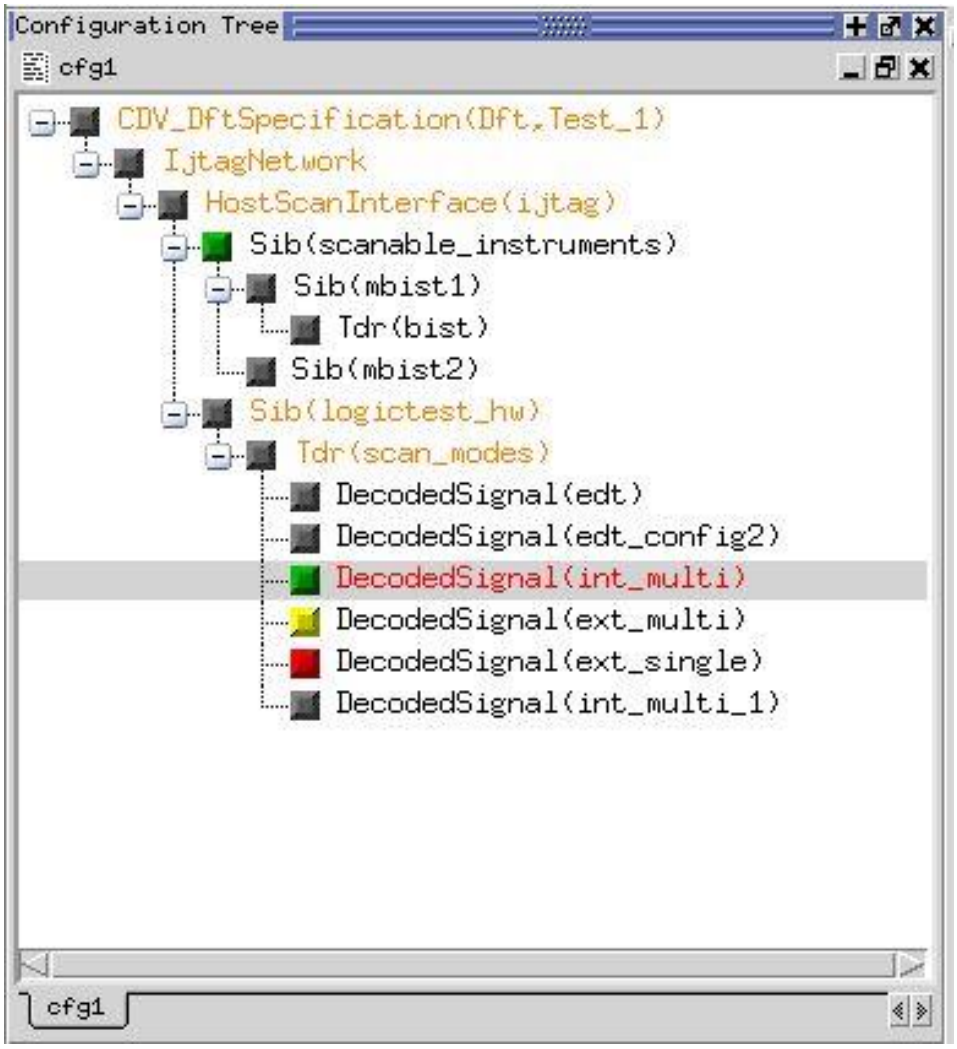


Figure 5: Enhanced Tree Widget

Similarly image can be modified for different purposes. In the above example nodes are shown in four different colors of images to indicate status of a node. E.g. gray is used of default status, green for pass, red for fail and yellow for don't care nodes. The mechanism

3. Summary:

This paper explains various widget enhancements on the top of basic widgets which are useful in real time applications. These widgets are modular and reusable with tcl/tk code. These widgets are created for developer to lower the effort for implementing fancy and sophisticated widgets.

4. Bibliography:

TCL wiki, http://wiki.tcl.tk