VM LABS

Easy XLISP/TK API Manual

GUI extensions to the Puffin Debugger's built-in XLISP scripting language

Revision 1.10
2-Jun-99

Note: This document is continually updated to reflect the current state of the Nuon development system hardware and software. If you have a version that is more than five or six months old, it is likely out of date.

Please address comments or report errors to Mike Fulton at VM Labs (mfulton@vmlabs.com).

VM Labs, Inc.
520 San Antonio Road
Mountain View, CA 94040

Tel: (650) 917-8050
Fax: (650) 917-8052

Table of Contents

1. Introduction

This document describes a high level API for XLisp/TK, the TK extensions to XLisp. Easy X-TK is built on top of XLISP/TK and therefore provides full compatibility with it. All widgets created with Easy X-TK can be modified using the XLisp/TK API.

The idea behind Easy X-TK is to have a simple XLisp call for each existing widget in the TK widget set. The goal is to keep things simple so that it is very easy to build a graphical user interface using XLisp. However, the full power of TK is always available by falling back onto XLisp/TK.

For advanced usage it is recommended to take a look at XLisp/TK document as well as at one of the many existing books on TCL/TK.

This page intentionally left blank

2. Tags

When creating a user interface widget, one can specify a unique name tag for that object. Using that name tag it is possible to retrieve the value of the object without having to introduce a symbol for that object. The only symbol needed is the one of the root object, which in most cases will be the dialog box.

Container widgets like the Dialog Box widget provide the `'get-child` method which allows you to get to any child object of the dialog box.

Example:

```
(define dialog (make-dialog :title “Test Dialog Box”))
(make-entry :tag 'the-entry :text "Enter Name"  :parent dialog
:anchor 'w)
.
.
.
(define current-value ((dialog ‘get-child ‘the-entry)
‘get-value))
```

This page intentionally left blank.

3. Callback Handlers

Most widgets allow the specification of callback handlers. A callback handler will be called whenever the user changes the state of that widget.

Example:

```
(make-button :tag 'button1 :parent frame1 :text
"Print":side 'left :handler (lambda () (format #t
"~%Pressed Print Button")))
```

This page intentionally left blank.

4. Base Class: The TK-Widget

The widget is the base class for most TK GUI objects. It defines basic operations. It is an abstract class that should never be instantiated.

4.1 Methods

4.1.1.1 initialize

```
(widget 'initialize &key parent)
```

Initializes the widget. Takes the parent widget as an argument.

4.1.1.2 tk-widget

```
(widget 'tk-widget)
```

Returns the TK widget.

4.1.1.3 add-child

```
(widget 'add-child child)
```

Adds children to the contianer.

4.1.1.4 get-value

```
(widget 'get-value)
```

Returns the value of the widget. For a check box button for example this would be the state of the button.

4.1.1.5 set-value

```
(widget 'set-value!)
```

Sets the value of the widget.

4.1.1.6 configure

```
(widget 'configure &rest args)
```

Configures widget dependant properties. See the XLisp/Tk manual for properties of individual widget types.

5. Top-level Window Widgets

5.1 Class: Dialog-Box

```
(make-dialog-box title)
```

Creates a top level Dialog Box.

Arguments	
Title	Title string for the dialog box.

5.1.1 Methods

5.1.1.1 get-child

```
(widget 'get-child tag-name)
```

Returns the child widget corresponding to the given tag name. See the above chapter about tags for an example.

5.1.1.2 get-children-alist

```
(widget 'get-children-alist)
```

Returns a list of all objects in the dialog box.

This page intentionally left blank.

6. Container Widgets

6.1 Class: Frame

```
(make-frame &key tag parent border relief width height
       side anchor padx pady ipadx ipady fill expand)
```

Creates a Frame Container object.

Arguments	
Tag	Object's name
Parent	The parent of the object
Width	Width of the frame
Height	Height of the frame
Border	Width of the border
Relief	3D relief: `flat`, `sunken`, `raised`, `groove` or `ridge`
Side	Is the side which to place the widget: `top, bottom, right, left`
Anchor	Indicates how to locate the widget within the space allocated to it: `Center, n, ne, e, se, s, sw, w, nw`
Padx	Horizontal external padding
Pady	Vertical external padding
Ipadx	Horizontal internal padding
Ipady	Vertical internal padding
Fill	Control fill of packing space: `x, y, both,` none
Expand	Control expansion into the unclaimed packing cavity: 0, 1

This page intentionally left blank.

7. User Interface Widgets

The following widgets are the basic user interface objects. Most of them share a common set of arguments which will be listed below. For most of these widgets, two ways of placing them inside a container widget exists: by default the TK pack manager will be used. By using the grid option though, the TK grid manager can be used.

7.1 Common User Interface Widget Arguments

```
(make-xxx &key tag parent font fg bg justify width
       height grid column columnspan row rowspan sticky
       side anchor padx pady ipadx ipady fill expand)
```

Arguments	
Tag	Object's name
Parent	The parent of the object
Font	Font to use for a label
Fg	Foreground color of the widget
Bg	Background color of the widget
Justify	Justification of a label: `left, right, center`
Width	Width of the widget
Height	Height of the widget

7.1.1 Properties for the default place manager, the TK Pack Manager

Properties	
Side	Is the side which to place the widget: `top, left, right, bottom`
Anchor	Indicates how to locate the widget within the space allocated to it: `Center, n, ne, e, se, s, sw, w, nw`
Fill	Control fill of packing space: `x, y, both, none`
Expand	Control expansion into the unclaimed packing cavity: 0, 1
Padx	Horizontal external padding, in screen units
Pady	Vertical external padding, in screen units
Ipadx	Horizontal internal padding, in screen units
Ipady	Vertical internal padding, in screen units

7.1.2 Properties for the TK Grid manager:

Properties	
Grid	If set, the grid manager will be used instead of the pack manager
Column	Column of where to place this widget inside the grid
Columnspan	Tells the grid manager to span the widget over 'n' columns
Row	Row of where to place this widget inside the grid
Rowspan	Tells the grid manager to span the widget over 'n' rows
Sticky	Position the widget to any combination of: `n, s, w, e`
Padx	Horizontal external padding, in screen units
Pady	Vertical external padding, in screen units
Ipadx	Horizontal internal padding, in screen units
Ipady	Vertical internal padding, in screen units

7.2 Class: Label

```
(make-label &key tag parent text variable font fg bg
       justify width height
       grid column columnspan row rowspan sticky
       side anchor padx pady ipadx ipady fill expand)
```

Creates a Label object.

Arguments	
Text	Text string for the label
Variable	Optional variable to use as the text string for the label. The label will change whenever the contends of the variable changes.

Examples:

```
(make-label :tag 'label16 :text "Orange" :parent
frame0 :grid 'yes :row 2    :column 1 :sticky 'ws)

(make-label :tag 'label2 :text "Hello World" :parent
dialog :anchor 'w)
```

7.3 Class: Button

```
(make-button &key tag parent text handler font fg bg
       justify width height grid column columnspan row
       rowspan sticky side anchor padx pady ipadx ipady
       fill expand)
```

Creates a Button object.

Arguments	
Text	Text string for the button's label
Handler	Callback handler for the button: `(lambda() (cb-handler)`

Example:

```
(make-button :tag 'button2 :parent dialog :text "Press Me!"
:handler (lambda () (press-me-button-callback)))
```

7.4 Class: Radio-Button

```
(make-radio-button &key tag parent text handler variable value
       font fg bg justify width height grid column columnspan
       row rowspan sticky side anchor padx pady ipadx ipady
       fill expand)
```

Creates a Radio Button object.

Arguments	
Text	Text string for the radio button's label
Handler	Callback handler for radio button: `(lambda() (cb-handler))`
Variable	Optional TK-Variable storing the radio button's state
Value	Value associated with this radio button

Example:

```
(make-radio-button :tag 'radiobutton1 :parent frame2 :variable
my-radio-var :value "AM" :text "AM Band")
```

7.4.1 Methods

7.4.1.1 get-value

```
(widget 'get-value)
```

Returns the current value of the radio-button.

7.4.1.2 set-value

```
(widget 'set-value! value)
```

Set the radio-button to a given value.

7.5 Class: Radio-Button-Group

```
(make-radio-button-group &key tag parent
       label-list value-list variable
       border relief handler
       side anchor padx pady ipadx ipady fill expand)
```

Creates a Group of Radio Button objects.

Note: The grid manager is not available for this class.

Arguments	
Label-List	List of radio button's labels. One for each button
Value-List	List of radio button's values. One for each button
Variable	Optional TK-Variable storing the radio button group's state
Border	Width of the border
Relief	3D relief: `flat`, sunken, `raised`, `groove` or `ridge`
Handler	Specifies a callback handler for the radio button group: `(lambda (value) (callback-handler value)`

Example:

```
(define my-radio-label-list (list "AM Band" "FM Band"
"MW Band")
(define my-radio-var (make-tk-variable))
(define (show-radio-value value)
(format #t "~%You selected: ~S" value))

(make-radio-button-group :tag 'radiobuttongroup
       :parent dialog1
       :side 'left
       :initial-value "AM"
       :variable  my-radio-var
       :label-list my-radio-label-list
       :value-list '(“AM” “FM” “MW”)
       :handler (lambda (v) (show-radio-value v)))
```

7.5.1 Methods

7.5.1.1 get-value

```
(widget ‘get-value)
```

Returns the current value of the radio-button group.

7.5.1.2 set-value

```
(widget 'set-value! value)
```

Set the radio-button group to a given value.

7.5.1.3 get-value-index

```
(widget 'get-value-index)
```

Returns the index of the selected radio button.

7.5.1.4 set-value-index

```
(widget 'set-value-index! n)
```

Selects the n-th radio button.

7.6 Class: Check-Button

```
(make-check-button &key tag parent text handler variable
       font fg bg justify width height
       grid column columnspan row rowspan sticky
       side anchor padx pady ipadx ipady fill expand)
```

Creates a Check Button object.

Arguments	
Text	Text string for the check button's label
Handler	Callback handler for check button: `(lambda() (cb-handler))`
Variable	Optional TK-Variable storing the check button's state

Example:

```
(make-check-button :tag 'chechkbttn  :parent frame1  :text
"Create backup" :variable backup-bttn-var)
```

7.6.1 Methods:

7.6.1.1 get-value

```
(widget 'get-value)
```

Returns the current value of the check-button.

7.6.1.2 set-value

```
(widget 'set-value! value)
```

Set the check-button to a given value.

7.7 Class: Check-Button-Group

```
(make-check-button-group &key tag parent
       label-list variable  border relief initial-value
       handler side anchor padx pady ipadx ipady fill expand)
```

Creates a Group of Check Box Button objects.

Note: The grid manager is not available for this class.

Arguments	
Label-List	List of check box button's labels. One for each button
Variable	Optional TK-Variable storing the check box button group's state
Initial-value	Initial value of the group given as a vector.
Border	Width of the border
Relief	3D relief: `flat`, `sunken`, `raised`, `groove` or `ridge`
Handler	Callback handler for check button group: `(lambda (button-index value) (cb-handler (button-index value))`

Example:

```
(define mylabel-list (list "Red" "Green" "Blue"))
(define red-bttn-var (make-tk-variable))
(define green-bttn-var (make-tk-variable))
(define blue-bttn-var (make-tk-variable))
(define myvar-list (list red-bttn-var green-bttn-var
blue-bttnvar))

(make-check-button-group :tag 'checkbuttongroup
```

```
:parent dialog2
:side 'left
:label-list mylabel-list
:variable-list myvar-list)
```

7.7.1 Methods

7.7.1.1 get-value

```
(widget 'get-value)
```

Returns the current value of the check-button group as a vector.

7.7.1.2 set-value

```
(widget 'set-value! value)
```

Set the check-button group to a given vector value.

7.7.1.3 get-nth-box

```
(widget 'get-nth-box)
```

Returns the current value of the n-th button in the group. Value is #t or #f.

7.7.1.4 set-nth-box

```
(widget 'set-nth-box)
```

Sets (checks) the n-th button in the group.

7.7.1.5 clear-nth-box

```
(widget 'clear-nth-box)
```

Clears (unchecks) the n-th button in the group.

7.7.1.6 toggle-nth-box

```
(widget 'toggle-nth-box)
```

Toggles the n-th button in the group.

7.8 Class: Entry

```
(make-entry &key tag parent text variable fg bg font justify
       width grid column columnspan row rowspan sticky
       side anchor padx pady ipadx ipady fill expand )
```

Creates a Text Entry object.

Arguments	
Text	Text string for the entry's label
Variable	Optional TK-Variable containing the value of the entry field.

7.8.1 Methods

7.8.1.1 get-value

```
(widget 'get-value)
```

Returns current value of the text entry.

7.8.1.2 set-value

```
(widget 'set-value! string)
```

Sets the text entry to a given string.

7.9 Class: Scale

```
(make-scale &key tag parent label initial-value from to
       handler length sliderlength variable orient
       grid column columnspan row rowspan sticky
       side anchor padx pady ipadx ipady fill expand)
```

Creates a Scale object.

Arguments	
Label	Optional label of scale
Initial-value	Initial value of the scale
From	Scale's lower limit
To	Scale's upper limit
Handler	Callback handler for scale: `(lambda (value) (cb-handler value))`
Length	Length of the scale box in screen units
Sliderlength	Length of the slider in screen units

Arguments	
Variable	TK Variable holding the value of the scale
Orient	Orientation of the scale: `horizontal` or `vertical`

Example:

```
(make-scale :tag 'scale42 :label "First" :parent
dialog
            :from 1 :to 10 :initial-value 5
            :handler (lambda (v) (format #t
           "~%First ~S" v)))
```

7.9.1 Methods

7.9.1.1 get-value

```
(widget 'get-value)
```

Returns the current value of the scale.

7.9.1.2 set-value

```
(widget 'set-value! value)
```

Sets the scale to a give value.

7.10 Class: Scale With Buttons

```
(make-scale-with-buttons &key tag parent label initial-value
from to handler length sliderlength variable orient grid column
columnspan row rowspan sticky side anchor padx pady ipadx ipady
fill expand)
```

Creates a Scale object inside a frame with two buttons, one to the left and one to the right. Clicking on those buttons increases/decreases the value of the scale by one.

Arguments	
Label	Optional label of scale
Initial-value	Initial value of the scale
From	Scale's lower limit
To	Scale's upper limit
Handler	Callback handler for scale: `(lambda (value) (cb-handler value))`

Arguments	
Length	Length of the scale box in screen units
Sliderlength	Length of the slider in screen units
Variable	TK Variable holding the value of the scale
Orient	Orientation of the scale: `horizontal` or `vertical`

Example:

```
(make-scale-with-buttons :tag 'scale42 :label "First"
            :parent dialog
             :from 1 :to 10 :initial-value 5
             :handler (lambda (v) (format #t
            "~%First ~S" v)))
```

7.10.1.1 Methods:

7.10.1.2 get-value

```
(widget 'get-value)
```

Returns the current value of the scale.

7.10.1.3 set-value

```
(widget 'set-value! value)
```

Sets the scale to a give value.

7.11 Class: Scrollbar

```
(make-scrollbar &key tag parent orient scrollx scrolly grid
column columnspan row rowspan sticky side anchor padx pady
ipadx ipady fill expand)
```

Creates a ScrollBar object.

Arguments	
Orient	Orientation of the scale: `horizontal` or `vertical`
Scrollx	Specifies the attached widget for horizontal scrollbars
Scrolly	Specifies the attached widget for vertical scrollbars

7.11.1 Methods:

7.11.1.1 get-value

```
(widget 'get-value)
```

Returns the current value of the scrollbar.

7.12 Class: ListBox

```
(make-listbox &key tag parent item-list value-list handler fg
bg font width height grid column columnspan row rowspan sticky
side anchor padx pady ipadx ipady fill expand)
```

Creates a List Box object.

Note: Tcl has a bug where it will unselect the current list box selection when a different widget gets selected. Use the List-Selection-Box object instead.

Arguments	
Item-List	List of strings to place into the list box
Value-List	Optional value list. Maps a given value to each item in the list.
Handler	Callback handler notifying of selection changes: `(lambda (selected-item) (cb-handler selected-item))`
Height	Note: Height is specified in number of list items that should be visible

Example:

```
(define (show-listvalue value)  (format #t
"~%List Selection: You selected index #: ~S" value))

(make-listbox :tag 'listbox
      :parent dialog
      :height 3
      :item-list '("Orange" "Banana" "Apple")
      :handler (lambda (v) (show-listvalue v)))
```

7.12.1 Methods

7.12.1.1 get-value

```
(widget 'get-value)
```

Returns the value of the currently selected item. If no value list has been given, this will be the index.

7.12.1.2 set-value

```
(widget 'set-value! value)
```

Set the current selection of list box to the given value.

7.12.1.3 insert

```
(widget 'insert index item &key value)
```

Inserts an item at position index into the list. If a value-list was provided at creation time, a value for the inserted item has to be specified as well.

7.12.1.4 delete

```
(widget 'delete index)
```

Deletes item at position index out of the list.

7.12.1.5 get-value-index

```
(widget 'get-value-index)
```

Returns the index of the currently selected item.

7.12.1.6 set-value-index

```
(widget 'set-value-index! n)
```

Selects the n-th item in the list.

7.13 Class: ListBox with Slider

```
(make-listbox-with-slider &key tag parent item-list handler
initial-value value-list height fg bg font width height grid
```

```
column columnspan row rowspan sticky side anchor padx pady
ipadx ipady fill expand)
```

Creates a List Box object with an attached slider.

Note: Tcl has a bug where it will unselect the current list box selection when a different widget gets selected. Use the List-Selection-Box object instead.

Arguments: Same as ListBox object.

7.13.1 Methods:

Same as ListBox object.

7.14 Class: ListSelectionBox

```
(make-listbox-selection-box &key tag parent item-list handler
initial-valiue value-list text height fg bg font width height
grid column columnspan row rowspan sticky side anchor padx pady
ipadx ipady fill expand)
```

Creates a List Box object with an attached slider and a label showing the currently selected item.

Arguments	
Text	Specifies a title for the list box.
Item-List	List of strings to place into the list box
Value-List	Optional value list. Maps a given value to each item in the list.
Handler	Callback handler notifying of selection changes `(lambda (value) (cb-handler value))`
Height	Note: Height is specified in number of list items that should be visible
Initial-Value	Sets the initial value of the object.

Example:

```
(define (show-listvalue value)  (format #t "~%List
Selection: You selected index #: ~S" value))

(make-list-selection-box :tag 'listbox
      :parent dialog
      :text “Fruit:”
      :item-list '("Orange" "Banana" "Apple" “Cherry”
                  “Plum” “Strawberry”)
      :handler (lambda (v) (show-listvalue v)))
```

7.14.1 Methods:

Same as ListBox object.

7.15 Class: OptionMenu

```
(make-option-menu &key tag parent text item-list value-list
handler initial-value fg bg font width height grid column
columnspan row rowspan sticky side anchor padx pady ipadx ipady
fill expand)
```

Creates a Drop Down Menu object.

Arguments	
Text	Specifies a label for the menu.
Item-List	List of strings to place into the menu
Value-List	Optional value list. Maps a given value to each item in the menu.
Handler	Callback handler notifying of selection changes `(lambda (selected-item) (cb-handler selected-item))`
Initial-Value	Sets the object to the given initial value.

7.15.1 Methods

7.15.1.1 get-value

```
(widget 'get-value)
```

Returns the value of the currently selected item.

7.15.1.2 set-value

```
(widget 'set-value value)
```

Sets the value of the menu.

8. Miscellaneous Widgets

8.1 Class: Message-Box

```
(make-message-box message title icon type)
```

Displays a modal Message Box.

Arguments	
Message	The text to be displayed in the box
Title	Specifies a title string for the message box
Icon	Specifies the icon to display: `error, info, question, warning`
Type	Specifies which of the predefined sets of buttons to display: `abortretryignore, ok, okcancel, retrycancel, yesno, yesnocancel`

8.2 Class: File-Selector-Box

```
(make-file-selector-box type &key title initialdir,
       initialfile defaultext, filetypes)
```

Displays a modal File Selector Box.

Arguments	
Type	specifies the type of file selector, open or `save`.
Title	Specifies a string to display as the title of the dialog box.
InitialDir	Specifies that the files in directory should be displayed when the dialog pops up. If this parameter is not specified, then the files in the current working directory are displayed.
InitialFile	Specifies a filename to be displayed in the dialog when it pops up (only used in Save mode)
DefaultExt	Specifies a string that will be appended to the filename if the user enters a filename without an extension
FileTypes	If a FileTypes listbox exists in the file dialog on the particular platform, this option gives the filetypes in this listbox. When the user choose a filetype in the listbox, only the files of that type are listed.

```
(make-file-selector-box type ‘open)
```

This page intentionally left blank.

9. Utility Functions

9.1 Function: TK-Variable

`(make-tk-variable)`

Creates a TK Variable to be used for objects like radio buttons.

Example:

```
(define myvar (make-tk-variable))
```

This page intentionally left blank.

10. Configuring Easy TK Widgets through XLisp/TK

It is possible to change TK widget attributes, even those not exposed through the Easy TK API, using XLisp/TK.

Every Easy TK object knows of the method `'tk-widget` which returns the corresponding TK widget. The `'configure` method then lets you change the attributes of a widget.

Example:

Creating a Text widget:

```
(define my-entry (make-entry :tag 'entry1 :text "Enter"
:parent frame1))
```

Create a Scrollbar and attach it to the Text widget:

```
(define my-scrollbar (make-scrollbar :tag 'scrollbar1
:parent frame1
:expand 'yes :fill 'y
:side 'right :anchor 'nw
:scrolly (my-entry 'tk-widget)))
```

Change the text (foreground) color of the Text widget to red:

```
((my-entry 'tk-widget) 'configure :fg 'red)
```

Attach the Text widget to the scrollbar:

```
((my-entry 'tk-widget) 'configure :yscrollbar
(my-scrollbar 'tk-widget))
```

This page intentionally left blank.