Tk Widgets in Javascript

An implementation of Tk widgets using Javascript.

A paper for the Eighteenth Annual Tcl/Tk Conference

Abstract

Tk in Javascript is a work in progress, which started about May 2011 and is part of the incr Tcl in Javascript project is, which is intended to be one possible frontend/client part of ATWF and Reporting Tools with Tcl. It tries to implement Tk widget using javascript and DOM trees.

That includes a mapping of for example button/label/entry widgets to something which can be done with HTML parts in creating DOM trees and adding properties and attributes to the DOM nodes, that includes mapping of Tk option model to javascript style model and properties of DOM nodes.

Second goal is to map Tk event handling and bind functionality to the javascript event model and the javascript event listeners/handlers. There are also more complex widgets in work like Tree, Tktable, panedwindow, combobox etc.

The selection on which widgets are implemented first is driven by: what is needed for a reporting environment, that includes the decision on which options are implemented first.

Contact information

Arnulf Wiedemann
Lechstr. 10
D-86931 Prittriching
Email: arnulf@wiedemann-pri.de

Inhaltsverzeichnis

1How it started

During the implementation of [incr Tcl in Javascript] there did raise the question, how to get easy and efficient access to the DOM information. First idea was to give a user direct access to javascript DOM commands using a Tcl wrapper, but that would force the user to learn a lot of javascript and its model for widgets and events. So after some thoughts in that direction it seemed better to try to map Tk functionality to javascript DOM and event model. There followed the decision on directly using DOM for the implementation without first using HTML code and let the browser convert that to the DOM info.

Next there was the need to find out how to map basic Tk widgets to the DOM model like:

•button

•entry

•frame

•label

•toplevel

As I had in mind to use that as a client/frontend for [Reporting Tools with Tcl], these were the base widgets for starting.

After some experiments I decided to use for a label widget, <button> for a button widget, <input> for an entry widget and again for a frame widget and a toplevel widget.

2The initially implemented Tk widgets

2.1 [TkWidget js Object]

Parameters:

- interp
- name
- full_name
- type

To hold all the relevant information for a [Tk] [widget] a [TkWidget js Object] was implemented, which stores in it's properties for example the widget name (i.e. .fr.b1), a reference to the DOM node, the type of the widget (for example Tk.WIDGET_TYPE_BUTTON) a reference to a javascript Object with properties for that instance of the widget, a list of allowed options for that widget and a list of bind infos.

The javascript Objects for the above mentioned widget are:

- [TkButton js Object]
- [TkEntry js Object]
- [TkFrame js Object]
- [TkLabel js Object]
- [TkToplevel js Object]

The instances for the widgets are implemented very similar to the implementation of itcl class objects as new commands, The javascript object used for this is [TkObject js Object]. It contains as properties the path name and a reference to the [TkWidget js Object].

2.2 [TkButton js Object]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information for a Tk button widget instance.

2.3 [TkEntry js Object]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information for an Tk entry widget instance.

2.4 [TkFrame js Obejct]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information for a Tk frame widget instance.

2.2 [TkLabel js Object]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information for a Tk label widget instance.

2.3 [TkToplevel js Object]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information for a Tk toplevel widget instance.

3Implemented Widgets

All Tk widget commands are implemented in namespace ::tk but for compatibility there are interp aliases to be able to use for example button without the ::tk:: namespace prefix for creating a new button instance.

So we have the following commands:

•::tk::button

•::tk::entry

•::tk::frame

•::tk::label

•::tk::toplevel

From experiments I found out, that you need a element around most of these widgets to be able to force width and height requirements and also when packing these widgets. So in HTML a button widget would look like so:

  <button>button1</button

The sub commands for a button could not be implemented directly as a namespace ensemble, as the button itself is implemented like an itcl class, so that namespace cannot be additionally be used for a namespace ensemble.

So for example the cget and configure command of a button widget are implemented as ::tk::button::configure and ::tk::button::cget and are handled like itcl class methods. As a result you can use .b1 cget … and .b1 configure …, if .b1 has been created as a button widget instance using: button .b1 …

But be aware: a Tk widget instance is no class object and a sub command of the widget is no class method. It is only handled very similar to these parts, to be able to use common code!

4Tk Options

Now there was the question on how to get all the different standard and widget specific options mapped to the style properties of a DOM element.

4.1 [TkStandardOptions js Object]

Parameters:

- interp

For handling of all the possible options (including Tk standard options), the attachment of the allowed options to a widget type and a mapping between Tk options and javascript options a [TkStandardOptions js Object] has been implemented.

It contains all the possible options of the Tk widgets according to the Tk option model with configure name, option name, option class and default value for an option and for every widget type there is a list containing all the options, which are allowed for a specific widget.

The allowed options are used for checking against illegal options and for producing an error message with the allowed options.

4.2 [TkOptionTemplate js Object]

Parameters:

- interp
- configure_name
- alias_name
- option_name
- option_class
- default_value

To hold information for one specific option a [TkOptionTemplate js Object] is used. It has properties for configure name, option name, option class and the default value as well as a possible alias name for example -background for the -bg option.

4.3 [TkOption js Object]

Parameters:

- interp
- option_template_obj
- option_value

The [TkOption js Object] contains a reference to the [TkOptionTemplate js Object] for the relevant option and a property for the current value (this is

used for fast reference instead of looking it up in the DOM tree). Maybe this will change in the future to save space.

5Javascript Objects for Tcl usage

5.1 [JsDomNode js Object]

Parameters:

When working with rhino (a command line javascript interpreter for Linux) it was necessary for being able to test simple DOM related parts easily to implement a dummy javascript Object for at least creating and appending DOM nodes. This has been done with the implementation of [JsDomNode js Object]. It has functions for creating elements and setting attributes and appending elements, as that functionality is not available in rhino.

5.2 [JsOption js Object]

Parameters:

- interp
- configure_name

A container for a javascript option to be mapped to an Tk option.

To hold information for one specific option a [JsOptionTemplate js Object] is used.

5.3 [JsOptionTemplate js Object]

Parameters:

- interp
- configure_name

Used in [JsStandardOptions js Object] to hold information about a specific style property.

It has properties for configure name and a container for sub options, as in javascript options can be structured like border can be set directly identical for all 4 sides or via borderTop, borderLeft, borderRight and borderBottom.

The [TkStandardOptions js Object] contains references to the [TkOptionTemplate js Object] for the relevant option.

5.4 [JsStandardOptions js Object]

Parameters:

- interp

A container which initializes the standard javascript options and has a simple mapping between Tk option and js options (not yet complete)

6Tk Javascript Objects

As a base for the Tk bind command a simple parser for an event sequence has been implemented, which can parse the modifier, type and detail parts. This information is stored in a [TkEventSequence js Object].

6.1 [TkEventSequence js Object]

Parameters:

- interp
- modifier1
- modifier2
- type
- detail

A container for storing information about a [Tk] [event] sequence for use for example by the [Tk] [bind] command.

For positioning widgets within a browser window [Tk] [pack] and [grid] commands have been implemented. [Tk] [grid] command is still very rudimentary and not yet really usable.

6.2 [TkGrid js Object]

Parameters:

- interp
- widget_obj

Container for grid information for example the widgets and the row/column info etc.

6.3 [TkObject js Object]

Parameters:

- interp
- name
- widget_obj

Container for a [Tk] [widget]. This is the instantiated command object, which is created when a [Tk] [widget] like a [button] for example is instantiated. It contains a [TkWidget js Object], which holds the specific info for that instance of the widget. Also all the specific option values for that widget are collected here and can be modified and fetched using the configure and cget sub command of the widget.

When setting an option with the configure command, the option name and

value are mapped from the [Tk] values to the the style or attribute name and the appropriate value for javascript.

For example the Tk option -foreground is mapped to the javascript style attribute “color” and the -text option of a Tk button is mapped to the “textContent” field when using Firefox. That is still a different field name for IE, which has yet to be implemented.

6.4 [TkPack js Object]

Parameters:

- interp
- widget_obj

Container for pack information for example the widgets and other related info. The pack command appends DOM nodes created for the different widgets as DOM nodes with no connection to the visible window content to an existing DOM node for example the node of the HTML <body> tag. For a [toplevel] [widget] a HTML node is created and appended to the <body> tag.

The style attributes width and height have to be provided as a “100px” pixel value with “px” at the end and determine the width and height of the toplevel widget/window.

For implementing the -side left and -side right option of the pack command one can use the style attributes {float: left} and {float: right} respectively.

For -side top the style float part is omitted, for -side bottom I have not yet found out how to do that. To avoid wrapping of widgets it is normally necessary to have style attribute display set to {display: table-row}, if you have {float: left} or {float: right} but, what a pitty not always, there are special cases where you also for that case have to use {display: block} I have not yet found out all the rules on how to use that. For the small test cases it works, but I am pretty sure there are still a lot of failing cases.

7BWidget Tk Widgets

7.1 [TkTree js Object]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information for a BWidget Tree widget. That one was rather hard, as in the end it is implemented as an own table for every node in the tree. That is necessary as otherwise you do not easily get all the columns of the tree rendered with the same size.

I have got inspiration from a treeview implementation from GubuSoft and from YUI treeview. The tick is to use little images for the opentree, closetree and for the lines between the nodes and the image for the node. The last field in the table does not handle overflow, so that the text labels can be as long as they are needed.

There is an event handler attached to the opentree and closetree images and another one to the node image and the node text, these can also be different using the Tk bindImage and bindText options.

The opentree and closetree event handlers just set the first level sub nodes style of the selected node to {display: none}, which makes that subtree invisible after rendering (it needs no space any more and looks like the nodes have been removed). When opening the tree again one just has to set the sub nodes style to {display: block} or {display: table-row} and the subtree is visible again.

7.2 [TkLabelEntry js Object]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information for a BWidget LabelEntry widget. It combines a label (a) element with an entry (a <input>) element, surrounded by another element (in principle a frame). You can set the configure options for the label and the entry part.

7.3 [TkScrollableFrame js Object]

Parameters:

- interp
- path_name

- widget_obj

A container for holding information for a BWidget ScrollableFrame widget. It combines a frame (a) element with another frame and for the second frame the style attribute overflow is set to {style: auto}. The sub command getframe returns that inner frame path name (DOM element).

8TkTable Widget

8.1 [TkTable js Object]

Parameters:

- interp
- path_name
- widget_obj

A container for holding information about a TkTable table widget. This one is tricky too. It is built with 5 tables. The outermost is used to hold four other tables:

- top_left_table
- top_right_table
- bottom_left_table
- bottom_right_table

This is necessary for being able to have title rows and title cols which are not scrolled. Top_left_table has the part in the top left corner with the row parts and col parts, which are never scrolled. Top_right_table has the title row parts, which are only scrolled horizontally when the table is scrolled. bottom_left_table has the title cols, which are only scrolled vertically when the table is scrolled, and bottom right table contains the rows and columns which are visible and can be scrolled horizontally and vertically.

Here are also the scrollbars and when scrolling depending on the direction either the top_right_table is scrolled accordingly or the bottom_left_table. This is done by setting the scrollLeft or scrollTop attribute of these tables to the same value as the corresponding value of the bottom_right_table when scrolling the bottom_right_table. With that trick it looks like both tables are scrolling synchronous.

When calculating the position and table of a cell title rows and title cols have to be taken in account on deciding which of the four tables holds the desired cell. And for giving back the index of a cell has to be done the same way for getting the absolute index of the cell.

When the -command option is used, every cell gets attached the click event and in the event handler the command script is called with the pah name of the table and the index of the cell in nn,nn syntax are passed as parameters to the script.

9Advanced Tk Widgets

9.1 TkPanedWindow

Parameters:

- interp
- path_name
- widget_obj

A container for holding information about a panedwindow widget. A panedwindow is built as a element. Every adding of a pane adds two elements, one for the sash, and one for the pane.

The sash element knows about the 2 surrounding elements and changes their sizes when dragging the sash. When more than one sash element exist, the second sash element is also moved, when the first one reaches that and the pane before or after that second sash element (depending on the direction of the moving) is also changed in size.

Two sash elements can touch each other that means the pane in between has size 0. And all the sashes can be moved to the left or right or to the bottom or top border.

The -side option when adding a pane determines – as with the pack command – how the panes are arranged.

10YUI revisited

Because of a lot of still missing base functionality in the Tk implementation in Javascript the YUI Implementation and functionality has been revisited and inspected again. Because of some functionality, which was needed and was available there the decision was made to extract some base functionality to use here. After analyzing YUI to see how to use parts without need for ant build system a solution was found to use parts of YUI without using the build system, which resulted in only 2 lines to be added to a source file for using it here. For easily being able to enhance with Tk specifics there was the decision to do a fork of the sources and start only with limited set of modules from YUI. The name for that fork is TUI (Tcl User Interface), but up to now 90% of original code is still used (for the modules taken over, which is only a smaller part of the full implementation about 20-25%). There have been about 25.000 lines of the source code adapted for TUI usage (not much had to be modified).

After that the implementation of “derived classes” has been started to build a button widget with YUI functionality.

The next part (still in progress) is the implementation of a Tktable widget. There exists a datatable implementation, but that has only title lines and there are always at least one title line. Titlecols functionality is missing completely and also the tag functionality is missing so there has been started an implementation of a Tktable widget based on the TUI functionality and similar to the datatable implementation of YUI. As in YUI scrolling will be done with a plugin attached to Tktable widget.

Other widgets like Tree, panedWindow, ScrollableFrame etc. will follow.

11Status

The implementation is in the middle of the minimal necessary functionality. Work will continue to get a version, which has at least the minimal necessary functionality to be able to serve as a frontend for [Reporting Tools With Tcl] to build a minimal reporting system together with [ATWF]. As with [incr Tcl in Javascript] there are missing tests and demos.