

Switching to Tile

Rolf Ade

April 2005

Abstract

The Tile package is probably the most thrilling and ambitious effort to revitalize Tk so far. Tile adds new abilities to control and change the look and feel of Tcl/Tk applications with so-called 'Themes'. Especially, tile provides native looking widgets at Windows and Mac OSX. Plus the package provides some additional widgets. This paper gives an overview over tile at version 0.6.2 and tries to guide Tcl/Tk developers thru the first steps of using it¹.

1 Overview

The tile package had an astonish career. Less then two years ago, Joe English made the first sources public available. Today, although the version number suggest, that tile is still in its early days, a few Tcl Core Team Members are semi-official committed to push the inclusion of tile as a bundled package of the upcoming 8.5 Tcl/Tk release². If that should in fact happen, that wouldn't be a radical change. The tile package is in some sense orthogonal to the current Tk core - just don't use it, and you get the familiar Tk.

At the first look, tile is in short about 'eye candy'. The debate about the look and feel of Tk has a long history. In the early days of Windows and Mac support (version 7.5), Tk looked the same on every supported platform. Later on (starting with 8.0), Tk used some platform specific controls under Windows and Mac. But the world moved on. Windows XP, Mac OSX and under Linux KDE and Gnome brought theming support on the desktop - and that renewed the look and feel problem for Tcl/Tk application programmers. For example, while it is possible with some effort, to write a Tcl/Tk application, which looks nearly native on Windows 2000, this isn't really possible on Windows XP - the Tcl/Tk application will look 'foreign' on this platform.

Tile aims to solve this problem by theme-able reimplementations of the core

¹The author has followed the development of and discussion about tile from early states on but he isn't involved so far into its development. This is a report from an outside viewpoint and like an article about using a new technology. At the Eleventh Annual Tcl/Tk Conference 2004, Joe English presented a paper more from a tile developers view point[4].

²Though there is up to now no TIP, which proposes that really officially.

widgets³. The basic idea of this reimplementaion is, to separate the code responsible for the appearance of a widget from the other parts of the implementation. Selecting a theme means then just switching from one widget drawing code to another behind the scene.

But from where does this code come? Well, sure, it has to be written. Much better: that code is already written and is part of tile. For the first steps, Joe Tcl/Tk Programmer need not to care about creating complete new themes. For the ambitious, this is of course possible. New themes can be implemented as add-on packages written in Tcl or in C, depending on the level of customization required. But on windows, there is already a theme `xpnative`, which uses the Windows "Visual Styles" API to make tile widgets indistinguishable from native controls. And on Mac OSX is the `aqua` theme available, which uses the Carbon Appearance Manager. The default theme has a new, streamlined look, compared with Tks current Motif-like appearance on X11 (which is also available as theme `classic`). And there are more themes included⁴.

Beside all that 'eye candy' it should not be forgotten, that tile provides a handful additional (of course theme-aware) widgets. One additional gift, that tile *may* bring is a new kind of meta-widget framework, but that word isn't spoken yet.

2 Getting started

The tile sources are hosted under the umbrella of the TkTable project on SourceForge[5]. Given, that a recent Tcl/Tk version is installed (the current Tcl/Tk 8.4.9 will do well) building under linux and probably any other sane unix implementation is just a matter of `configure; make all; make install`. For compiling on Windows, a VC++ makefile is included. The tile download page at SourceForge provides a multi-platform storkit. Additionally, tile binaries are provided by (among others) ActiveStates ActiveTcl distribution[1] and Daniel Steffens TclTk Aqua Batteries-Included distribution[8]. Since version 0.6 every released version has an even last number. Version numbers with an odd patch number indicate CVS snapshots. So, the next release will have at least the version 0.6.4.

Tile is a well-behaving Tcl package. To use it, just put

```
package require tile
```

near the top of your main script. The scripted library code of the package selects at package loading time depending on the platform the 'right' theme (that is `xpnative` on Windows XP, `winnativ` on other versions of Windows, `aqua` on Mac OSX and the `default` theme on X11⁵). There is always exactly one theme

³To do this, tile uses the Tk theme engine. Without much notice outside the tcl-core mailing list Frédéric Bonnet laid the ground of Tk theming support with TIP 48[3]. Tile is the first known package, which uses this theme engine. The tile developers revised and enhanced the theme engine on the way.

⁴In fact, early adaptors have already written 3-party themes. Most of them are scripted themes, but Georgios Petasis has provided the start of a new C level theme, which instruments the Qt styling engine, to draw the widgets[7]

⁵If not otherwise set in the X resource database.

in effect at any one time. The proc `tile::availableThemes` returns a list of all available themes. The name of the current theme is stored in the variable `tile::currentTheme`. To switch the current theme, use the helper proc⁶

```
tile::setTheme <theme name>
```

When you switch themes, the tile widgets are redrawn automatically with their new look - unlike Tk, where changes to the option database with `option add` let existing widgets alone and change only the appearance of newly created widgets.

The tile reimplementations of the Tk core widgets are implemented in the `ttk` namespace and have the same name as their Tk counterparts. It's a nice experiment to locally override the Tk core widgets in your application's namespace(s) with their tile reimplementations with

```
namespace import ttk::*
```

Normally, your application should at least start in a new look and even mostly work as normal. But in most cases it will also be obvious, that this was not completely all work to do to switch to tile. Before we discuss the most common problems with the migration to tile, we take a closer look at how tile works.

3 How styling works

Programming GUIs with Tk is like writing a text in a word processor as OpenOffice or Microsoft Word. As the writer may change the font, foreground, background etc. of every text item, so has the Tk programmer with the help of tons of options the control over the appearance of the Tk widgets. Tile is more like a markup language as \LaTeX . Lots of appearance details like border, font, foreground, background etc. are handled by the chosen theme. Even more: the tile widgets doesn't allow to change a lot of this appearance aspects with widget options any more. Though, for compatibility reasons (to make migration easier), the tile widgets 'know' all the options of their corresponding Tk widgets, but they ignore some of them. Figure 1 shows this in detail for the Tk and tile button widgets.

As familiar from the Tk widgets, the default behavior of the tile widgets is controlled by the widget class bindings. The tile reimplementations of the Tk core widgets have the same class name as their counterparts, prefixed with a 'T' (so, the tile button class is `TButton`, the tile entry class is `TEntry` and so on). The class bindings are independent from the theme - switching themes doesn't affect class bindings. In Tk, only the 'container widgets' `toplevel`, `frame` and `labelframe` have a `-class` option, to set the widget's class at creation time. In tile, every widget has a `-class` option. That makes it easier, to create

⁶It is not recommended to use `style theme use <theme name>`, because `tile::setTheme` loads the theme, if necessary and keeps the variable `tile::currentTheme` up to date. The latter is necessary, because the natural `[style theme use]` unfortunately doesn't return the current theme so far.

Options common to Tk 8.4.9 button and tile ttk::button		
-command	-compound	-cursor
-default	-image	-takefocus
-text	-textvariable	-underline
-width		
Tile ttk:button options present for compatibility, but ignored		
-activebackground	-activeforeground	-anchor
-background (and -bg)	-bitmap	-borderwidth (and -bd)
-disabledforeground	-font	-foreground (and -fg)
-height	-highlightbackground	-highlightcolor
-highlightthickness	-justify	-overrelief
-padx	-pady	-relief
-repeatdelay	-repeatinterval	-state
-wraplength		
New tile ttk:button options		
-class	-padding	-style

Figure 1: Tk 8.4 button options versus Tile button options

different behaving controls based on the same widget. The tile distribution has the custom widget class `Repeater` (to be used for tile buttons) as an example.

The `-style` option of the tile widgets may be used to specify a custom widget style.

3.1 Widget Elements

With Tk, the widgets are the 'atoms' of the GUI. The tile widgets are not monolithic blocks, but itself build from smaller, simpler parts, the 'quarks' of a widget or, as they are officially called, the widget elements. For example, the Windows-style `button` has a border, a focus ring and a label, each of which are distinct elements.

Widget elements have options very much like widgets. For example, the default border element has `-borderwidth` and `-relief` options (in fact, most of the options, which have disappeared from the widgets will be found as element options).

Widget elements are usually implemented in C. The tile core provides a default set of elements. Every theme inherits this core elements, but it may overwrite the drawing code of a part or all elements with its own implementations (to get a different visual representation), with more or less or other element options. A theme may even add new elements. Elements can also be defined from Tk images to create pixmap themes.

While

style element names

returns the list of elements defined in the current theme, there isn't as of version 0.6.2 any other way than source code diving to know the valid options of that elements⁷. To give an overview, Appendix A has a table of all core elements with their valid options.

3.2 Widget layouts

Since tile widgets are composed of a collection of elements, that elements has to be layed out somehow to build the widget — this is done by style layouts⁸. Every theme may build a tile widget out of more or less elements and/or may arrange the elements in a different way. For example, the `classic` theme layout of the `scrollbar` widget places an arrow button on each side of the scrollbar, while this sample code by Joe English places one arrow button on the left and two arrow buttons on the right side:

```
style layout Horizontal.TScrollbar {
    Scrollbar.trough -children {
        Scrollbar.leftarrow -side left
        Scrollbar.rightarrow -side right
        Scrollbar.leftarrow -side right
        Horizontal.Scrollbar.thumb -side left -sticky ew
    }
}
```

The arrangement of elements work like a simplified version of Tk's `pack` geometry manager. It's even possible, to adjust the layout of a tile widget after creation.

3.3 Styles and States

Under the umbrella of a style are all the settings collected, which affects the appearance of one group of widgets, (normally) all of the same class. Styles are named, in a hierarchical way. For example, the programmer could use the style name `Toolbar.TCheckbutton` to collect all special settings needed for checkbuttons used in a toolbar. All settings, not explicitly set by the `Toolbar.TCheckbutton` style are looked up in the `TCheckbutton` style. If there are still not explicitly set options, then the settings for the 'root' style `.` will be used, and that style has always a default value for every option, per implementation.

For example, to set the default relief for the example customized style `Toolbar.TCheckbutton` simply use

```
style default Toolbar.TCheckbutton -relief flat
```

⁷The next tile release will allow to query the name of the options of a given element with `style element options <element name>`.

⁸The layout system is one of the enhancements of the TIP 48 style engine made by the tile developers.

But the value of a style option isn't just a static value. The value of a style option for a certain widget depends on the state of that widget. Every tile widget has a map of several flags, currently:

- active
- disabled
- focus
- pressed
- selected
- background
- alternate
- invalid
- readonly

Every state flag is independent from each other. Every tile widget has the widget commands `state` and `instate` to modify and query any combination of that flags. Now, so called 'state maps' could set specific values for every option of a style for any possible combination of states. While this is a powerful concept, the simple cases are easy to understand:

```
style map Toolbar.TCheckbutton -relief {
    disabled flat
    selected sunken
    pressed sunken
    active raised
}
```

If a widget state is changed, then the state map of its style is searched for the first combination of states, that matches. If there is a match, that value is used for the option. If there isn't a match, the default value will be used. Widget state changes usually happen in widget class bindings like:

```
bind TCheckbutton <Enter> { %w state active }
```

3.4 Themes

A theme is a named umbrella for widget elements, layouts and styles. The layouts arranges the elements to widgets and styles control the visual appearance of that widgets. At scripting level, a 3-party tile theme is an ordinary Tcl package, with the package name `::tile::theme::<theme name>`.

4 Migration Problems

It's the details, that matters. While it is often quite easy to start to migrate an application to tile (as shown above), there are also typically some issues to solve. One really obvious problem is, that tile widgets and Tk widgets often doesn't look good side by side in one dialog.

Currently, tile provides reimplementations of the following Tk widgets:

- `button`
- `checkbutton`
- `entry`
- `frame`
- `label`
- `labelframe`
- `menubutton`
- `radiobutton`
- `scrollbar`

Also already in the code (and at least basically working) is the start of a themed `scale` widget, but that isn't currently documented. Not as a replacement (which mimics the interface) of the Tk `panedwindow`, but as a similar widget there is the `ttk:paned` widget.

That means, a few Tk widgets currently haven't a theme-able tile counterpart. There isn't much problem with the `canvas` widget — it simply doesn't need themability. Mostly the same is valid for the `text` widget⁹. But even if we rule this out that means, that theme-able counterparts of the `menu` and the `spinbox` (and eventually `listbox`) widgets are currently missing pieces. If you have, for example, a `spinbox` in your otherwise tile-ified dialog, which looks foreign like a blain, there isn't currently much you can do other than adjusting the `spinbox` options as possible or even rewriting the dialog without the `spinbox`.

Unfortunately, it is as yet not easily possible, to query the styles of the current or other themes, to get, for example, the default font for `entry` widgets. This makes is harder, to adjust the option settings of Tk core widgets as close to the current theme as it may be possible.

Probably even more important also most scripted meta-widgets (like, for example, the popular BWidget package) and additional C coded widgets (like the BLT toolkit[6]) will look foreign side by side with tile widgets. For the most common non Tk core widgets like `combobox` and `notebook` widgets, tile provides its own theme-able versions. But even if they fit feature-wise, using them means rewriting parts of the GUI code.

⁹Though it may asked, why for example the font of `entry` widgets is controlled by the theme, but the font of a nearby `text` widget is not.

Tile has merged the `-padx` and `-pady` options into a single `"-padding"` option, which may be a list of up to four values specifying padding on the left, top, right, and bottom.

Another minor interface difference, which may require a bit code editing, is that the tile `label` widget has `-background` and `-foreground` switches (which overwrite the theme defaults), but the Tk shortcuts `-bg` and `-fg` are only present as unused backward compatibility switches.

For all tile widgets with a `-compound` option the the `-width` option always specifies the width in characters to allocate for the text string. In Tk, it's either the width in characters, or in screen units, depending on the value of `'-compound'`, for the widget.

Even the widget commands of the tile widgets are only *mostly* compatible with the corresponding Tk widgets. A few widget commands are not implemented yet. Figure 2 has the complete list. Really important are probably only the missing `checkbutton` and `radiobutton` widget commands. The obvious work-around is, to use the `-variable` option and then to change the associated variable value.

Widget	Missing widget commands
button	flash
checkbutton	deselect flash select toggle
radiobutton	deselect flash select
scrollbar	activate

Figure 2: Tk widget subcommands not yet implemented by tile

We've already discussed a few times, that the tile widget's appearance isn't specified on a per-widget base with options but is controlled by the settings of the current theme. That means, if you had, for example, an important red button somewhere in your application with the help of the `-background` option, this button looks like any other button in your application, if you use the tile `button` widget. This isn't a bad thing. It's a good basic rule for standard applications, to use standard controls and a standard appearance - and red buttons are not really common. Tile enforces this principle by design¹⁰. But what, if this is all good and well for you, but you need for whatever reason just that red button? A solution is, to sub-class the style of your widget:

```
style default Red.TButton -background red
::ttk::button .redbutton -text "The Red Button" -style Red.TButton
```

¹⁰Although, tile allows you also to give your application a special visual 'branding', which emphasis how outstanding your work is, by creating your own theme.

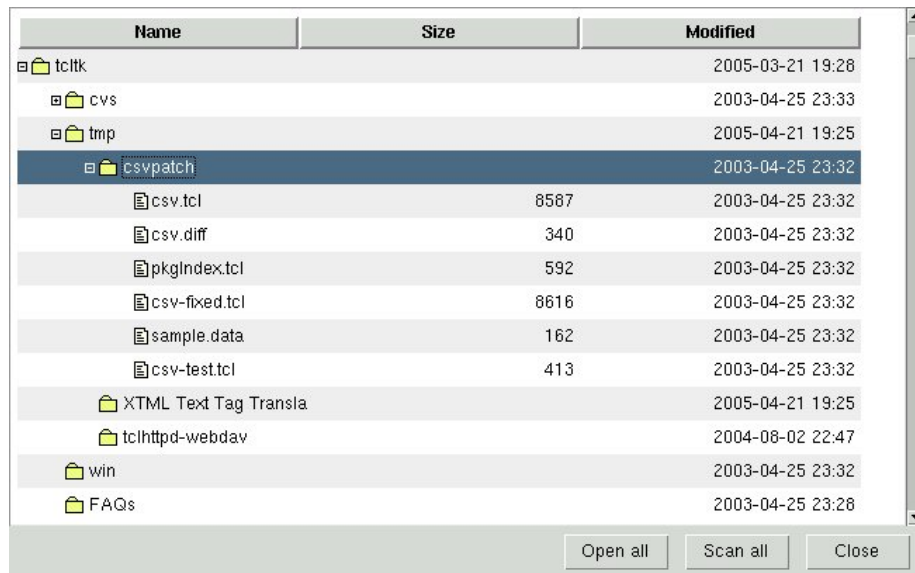
Depending on the style you sub-class you may also need to adjust the style map of your new style.

5 Additional widgets

Beside the styled counterparts of some of the core widgets, the tile package provides a few additional widgets. None of them is really novel. In fact, most of them are desired by Tcl/Tk developers since years and therefore there are (often several) alternative Tcl scripted meta-widget or even C coded implementations available.

For the early adaptors, which are already switching an existing code base to tile or writing a new application with it, especially the tile **combobox**, **notebook** and **progressbar** widgets are very helpful, because the current available scripted counterparts doesn't fit well into an otherwise tile-ified GUI. With the **classic** or **default** theme, the additional widgets are well usable within an otherwise 'pure classic' Tk application.

The tile **combobox** is, well, an entry field with an associated pop-down single-selection listbox. It seems to be thought-out comparatively mature. The tile demo directory even has a simple version of an inline auto-completion code. The tile **notebook** widget is a simple, single-tier notebook widget, similar to BWidget notebook. The **progressbar** widget supports two modes. The **determinate** mode shows the amount completed relative to the total amount of work to be done, and the **indeterminate** mode provides an animated display to let the user know that something is happening.



Name	Size	Modified
tcltk		2005-03-21 19:28
cvs		2003-04-25 23:33
tmp		2005-04-21 19:25
csvpatch		2003-04-25 23:32
csv.tcl	8587	2003-04-25 23:32
csv.diff	340	2003-04-25 23:32
pkgIndex.tcl	592	2003-04-25 23:32
csv-fixed.tcl	8616	2003-04-25 23:32
sample.data	162	2003-04-25 23:32
csv-test.tcl	413	2003-04-25 23:32
XTML Text Tag Transla		2005-04-21 19:25
tclhttpd-webdav		2004-06-02 22:47
win		2003-04-25 23:32
FAQs		2003-04-25 23:28

Figure 3: A screen-shot of the treeview demo script demobrowser.tcl

The tile **separator** widget is a simple Widget, very much like the BWidget

separator (and probably any other separator) widget. By default, it has no behavior in response to the user and just displays a horizontal or vertical separator bar. It is typically used for visual grouping of toolbars, but the tile demo has also an example for using it to visually structuring a dialog. The tile `treeview` widget (see Figure 3) is at the moment a much simple widget, than the also C coded `TkTreeCtrl` widget[2] or, for the tree part, the scripted `BWidget` tree widget. It can be used like a pure tree widget (without headings and columns). If it fits feature wise, its simple interface is an advantage. On a recent computer, this widget is able to handle up to a few hundred thousand tree nodes in fairly low time and with moderate memory needs. That means, it is able to handle a lot bigger trees than `BWidget` tree. The `treeview` widget currently support only `-yscrollcommand`, there is no `-xscrollcommand`. Multi-line entries in a `treeview` column cell doesn't really work as yet.

A Tile Core Elements

Tile Core Element	Options
Checkbutton.indicator	-background -borderwidth -indicatorcolor -indicatediameter -indicatormargin -indicatorrelief
Labelframe.text	-background -embossed -font -foreground -justify -text -underline -width -wraplength
Menubutton.indicator	-background -borderwidth -indicatorheight -indicatormargin -indicatorrelief -indicatorwidth
Progress.bar	-background -borderwidth -orient -sliderlength -sliderrelief -width
Radiobutton.indicator	-background

	-borderwidth -indicatorcolor -indicatediameter -indicatormargin -indicatorrelief
Treeheading.cell	-background -rownumber
Treitem.indicator	-diameter -foreground -indicatorargins
Treitem.row	-background -rownumber
arrow	-arrowcolor -arrowsize -background -borderwidth -relief
background	-background
border	-background -borderwidth -relief
client	-background -borderwidth
downarrow	-arrowcolor -arrowsize -background -borderwidth -relief
field	-borderwidth -fieldbackground
focus	-focuscolor -focusthickness
hsash	-sasthickness
hseparator	-background -orient
image	-background -image -stipple
label	-anchor -background -background -compound -embossed -font

	-foreground -image -justify -space -stipple -text -underline -width -wraplength
leftarrow	-arrowcolor -arrowsize -background -borderwidth -relief
padding	-padding -relief -shiftrelief
pbar	-background -barsize -borderwidth -orient -pbarrelief -thickness
rightarrow	-arrowcolor -arrowsize -background -borderwidth -relief
separator	-background -orient
slider	-background -borderwidth -orient -sliderlength -sliderrelief -width
tab	-background -borderwidth
text	-background -embossed -font -foreground -justify -text -underline -width

	-wraplength
textarea	-font -width
thumb	-background -borderwidth -orient -relief -width
trough	-borderwidth -troughcolor -troughrelief
uparrow	-arrowcolor -arrowsize -background -borderwidth -relief
vsash	-sashthickness
vseparator	-background -orient

References

- [1] ActiveState. Activetcl.
<http://www.activestate.com/Products/ActiveTcl>.
- [2] Tim Baker. Tktreectrl.
<http://sourceforge.net/projects/tktreectrl>.
- [3] Frédéric Bonnet. Tip 48: Tk widget styling support.
<http://www.tcl.tk/cgi-bin/tct/tip/48>.
- [4] Joe English. The tile widget set.
<http://tktable.sourceforge.net/tile/tile-tcl2004.pdf>, 2004.
The so far only published paper about tile from one of the main tile makers.
- [5] Joe English Pat Thoyts et al. The tile package.
<http://tktable.sourceforge.net/tile/>.
- [6] George A. Howlett. Blt toolkit.
<http://blt.sourceforge.net>.
- [7] Georgios Petasis. The tile-qt theme.
<http://cvs.sourceforge.net/viewcvs.py/tktable/tile-themes/tile-qt/>.
- [8] Daniel Steffen. Tcltk aqua batteries-included.
<http://tcltkaqu.sourceforge.net>.