Welcome!

BWidget goes Tile

Autor: Johann Oberdorfer
With special thanks to: Harald Oehlmann
Bwidget is a script-only package for tcl/tk offering GUI elements and its own mega widget system. The package was originally developed by Eric Boudailler (1997 to 2000) at Unifix.

In the meantime, ttk:: theme support has been integrated into Bwidget.

The current version is 1.9.1 (available in CVS) in addition, contains a lot of bug fixes, thanks to Harald Oehlmann who chased behind problems.

As of now, ActiveTcl 8.6.0.0b3 ships with Bwidget 1.9.1!

Hint: On the Mac you can find the package down below: /Library/Tcl/teapot/package/tcl/lib/BWidget1.9.1

A few good reasons, why it's worth to take a look at the actual development state.
It looks like, that for existing programs, the migration to tile is still an issue and most of the applications are running on a lower package version.

Bwidget so far is mostly **down-ward compatible** and so, friendly to existing code – that was also the main focus during development!

Migrating existing app's to tile can be done with reasonable amount of effort and time. That's why, programmers should consider to activate the style option when using the package.
The following comparison between BW and ttk should give us an overview of widgets, which might be replaced in the code by native ttk:: widgets. Widgets flagged as “deprecated” will be supported for compatibility reasons as usual.

Legend:
* ... deprecated
+ ... bwidget offers more options than ttk

<table>
<thead>
<tr>
<th>BWidge</th>
<th>ttk</th>
<th>Remark</th>
</tr>
</thead>
<tbody>
<tr>
<td>ArrowButton</td>
<td>-</td>
<td>* Doesn’t exist in ttk, could be replaced by a styled ttk_button</td>
</tr>
<tr>
<td>Button</td>
<td>ttk_button</td>
<td>+ -link option is not compatible with ttk, Button: offers build in DynamicHelp as well</td>
</tr>
<tr>
<td>ButtonBox</td>
<td>-</td>
<td>+ megawidget based on Button</td>
</tr>
<tr>
<td>ComboBox</td>
<td>ttk_combobox</td>
<td>+ ComboBox does support images within listbox!</td>
</tr>
<tr>
<td>Dialog</td>
<td>-</td>
<td>megawidget</td>
</tr>
<tr>
<td>DragSite</td>
<td>-</td>
<td>Drag’nDrop implementation, does work basically everywhere</td>
</tr>
<tr>
<td>DropSite</td>
<td>-</td>
<td></td>
</tr>
<tr>
<td>DynamicHelp</td>
<td></td>
<td>Does work for other widgets as well. There are quite a lot of “balloon help” packages available, e.g. tklib’s tooltip</td>
</tr>
<tr>
<td>Entry</td>
<td>ttk_entry</td>
<td>*</td>
</tr>
<tr>
<td>Label</td>
<td>ttk_label</td>
<td>*</td>
</tr>
<tr>
<td>LabelEntry</td>
<td>-</td>
<td>+ Convenient to use, less code</td>
</tr>
<tr>
<td>LabelFrame</td>
<td>ttk_labelframe</td>
<td>*</td>
</tr>
<tr>
<td>ListBox</td>
<td>-</td>
<td>Rich set of options, multicolumn, image support</td>
</tr>
<tr>
<td>MainFrame</td>
<td>-</td>
<td></td>
</tr>
</tbody>
</table>
## Identifying deprecated functionality:

<table>
<thead>
<tr>
<th>Widget</th>
<th>Ttk Replacement</th>
<th>Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>MessageDlg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>NoteBook</td>
<td>ttk_notebook</td>
<td>ttk_notebook looks nicer, but lacks tab management functionality</td>
</tr>
<tr>
<td>PagesManager</td>
<td></td>
<td>like this widget, for emulating mac 'a like UI experience</td>
</tr>
<tr>
<td>PanedWindow</td>
<td>ttk_panedwindow</td>
<td>*</td>
</tr>
<tr>
<td>PanelFrame</td>
<td></td>
<td>megawidget</td>
</tr>
<tr>
<td>PasswdDlg</td>
<td></td>
<td></td>
</tr>
<tr>
<td>ProgressBar</td>
<td>ttk_progressbar</td>
<td>*</td>
</tr>
<tr>
<td>ProgressDlg</td>
<td></td>
<td>megawidget</td>
</tr>
<tr>
<td>ScrollableFrame</td>
<td></td>
<td>Similar to tklib's autoscroll package, less code though</td>
</tr>
<tr>
<td>ScrolledWindow</td>
<td></td>
<td>-*</td>
</tr>
<tr>
<td>ScrollView</td>
<td></td>
<td>-*</td>
</tr>
<tr>
<td>SelectColor</td>
<td></td>
<td>megawidget</td>
</tr>
<tr>
<td>SelectFont</td>
<td></td>
<td>megawidget</td>
</tr>
<tr>
<td>Separator</td>
<td>ttk_separator</td>
<td>*</td>
</tr>
<tr>
<td>SpinBox</td>
<td>ttk::spinbox</td>
<td>*</td>
</tr>
<tr>
<td>StatusBar</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TitleFrame</td>
<td>ttk_labelframe</td>
<td>*</td>
</tr>
<tr>
<td>Tree</td>
<td>ttk_treeview</td>
<td>+ No question of doubt, Tree has rich set of functionality</td>
</tr>
</tbody>
</table>

Summary: 32 widgets, 9 of them can be possibly replaced by their ttk/tile counterparts.
So, does the Notebook widget for instance getting obsolete now due to the new ttk::notebook?

There is a simple answer to this question:
From a styling point of view yes, but still the Notebook widget has it's build in tab-scrolling behaviour, which –in real life– is hard to beat.

For a practical example, of how this feature can be used to create a dynamic notebook widget (similar to what is used in modern browsers), see:
http://wiki.tcl.tk/24470
Goals:

Backward compatibility whenever possible as a priority one issue.

Ability to dynamically switch styles for tk and ttk widgets to mimic ttk's behavior as much as possible.

Possibility to synchronize appearance and graphical properties of standard tk widgets with themed widgets.

Coding:

How to activate tile:

```bash
package require Bwidget 1.9.1
Bwidget::use -package ttk -style winxpblue -setoptdb 1
```
Style related Argument usage:

```tcl
# BWidget::use
# Argument usage:
#    -package ttk
#    |
#    specify a package name to be initialized, currently
#    support for the following packages is implemented:
#    ttk ... try to use tile'd widget set (if available)
#
#    -style default / native / myFavoriteStyleName
#    |
#    specify a valid style name,
#    use "BWidget::_get_colordcls" which gives
#    you a list of what's available for tk
#    |
#    if specified, BW tries to emulate OS color scheme,
#    a specific color scheme associated to each individual
#    operationg system is going to be used
#    |
#    same behaviour as before, stay compatible
#    with previous releases
#
#    -setoptdb [no=default|0|yes|1]
#    |
#    maintain the option database
#    if you need a dynamic behavior when changing
#    the underlying style, activate this option!
#
#    -themedirs {} = default / a list of valid directory names,
#    to specifying additional ttk theme packages
```
In respect of color settings, BW basically runs in 2 different modes:
- without tile:
The standard initialization sequence (as usual) is:

```
package require BWidget
```

In this case, color codes are set according to the OS currently running
on and acc. to a predefined color scheme, which is a "best guess" of
what might look good for most of the users ...

- With tile - "themed":
In addition to the standard initialization sequence, the following line
must be present in order to activate theming:

```
BWidget::*use -package ttk -style winxpblue -setoptdb 1
```
As styles are not supported within the BW distribution (except some themes found in the demo), a programmer needs to support required theme packages separately.

A typical initialization code block might look like:

```c++
BWidget::use \
-package ttk -style winxpblue -setoptdb 1 \n-themedirs [list [dir where to find specific theme package]]
```

In addition, a separate procedure must be provided, to manage and colorize standard tk "widgets", in case a `<<ThemeChanged>>` virtual event arises.

The procedure to handle this needs to follow the naming convention:

"Bwidget::<mythemeName>_Color"
In order to synchronize both kind of widgets (tk & ttk), a minimum set of keywords is used to achieve this behavior.

Unfortunately tk's option database does not have a dynamic behavior. Once a tk widget has been created, changing a graphical property in the option database won't have any effect.

When declaring the -setoptdb flag, it is possible to change colors of already created widgets. In this case, the <<ThemeChanged>> virtual event in addition with a callback routine is used, to retrieve and trace back relevant GUI elements.

<table>
<thead>
<tr>
<th>Component</th>
<th>Keyword</th>
</tr>
</thead>
<tbody>
<tr>
<td>SystemWindow</td>
<td>-background</td>
</tr>
<tr>
<td>SystemWindowFrame</td>
<td>-background</td>
</tr>
<tr>
<td>SystemWindowText</td>
<td>-foreground</td>
</tr>
<tr>
<td>SystemButtonText</td>
<td>-activeforeground</td>
</tr>
<tr>
<td>SystemButtonFace</td>
<td>-activebackground</td>
</tr>
<tr>
<td>SystemDisabledText</td>
<td>-disabledforeground</td>
</tr>
<tr>
<td>SystemHighlight</td>
<td>-selectbackground</td>
</tr>
<tr>
<td>SystemHighlightText</td>
<td>-selectforeground</td>
</tr>
<tr>
<td>SystemMenu</td>
<td>-background</td>
</tr>
<tr>
<td>SystemMenuText</td>
<td>-foreground</td>
</tr>
<tr>
<td>SystemScrollbar</td>
<td>-troughcolor</td>
</tr>
</tbody>
</table>
Reasons why we need the additional color mapping:

Styled color declarations and names do not follow a strict rule, so - most likely - there might be differences from theme to theme.

As a consequence of this fact, we have to support a minimum set of color declarations within Bwidget, which needs to be declared for each individual theme. Unsupported themes 'll fall back to the "default" color scheme!

During intialization, Bwidget looks for the existence of the following procedure, which needs to be adopted for a new theme:

```tcl
proc ::BWidget::aquativo_Color { } {
    variable colors

    set colors(style) "aquativo"
    array set colors {
        SystemWindow       "#EDF3FE"    SystemHighlight    "RoyalBlue"
        SystemWindowSize   "White"      SystemHighlightText "White"
        SystemWindowText   "Black"      SystemMenu          "LightGrey"
        SystemButtonFace   "#fafafa"    SystemMenuText      "Black"
        SystemButtonText   "Black"      SystemScrollbar     "White"
        SystemDisabledText "#fafafa"    
    }
}
```
Migrating to Tile – a practical example:

As a prove of concept, we are going now to migrate a well known application, e.g. ased3.0b16.vfs to tile:

Step1: copy bwidget's source to the required directory branch
Step2: and copying the requ. theme package to ased.../lib
Step3: Finally add the magic options:

```
package require Bwidget 1.9.1
BWidget::use \
  -package ttk \
  -style winxpblue \
  -setoptdb 1
```

The result looks like:
For sure, there is a bit more coding required, such as:
Try to get rid of arguments which interfere with tile like:
highlightthickness, highlightcolor, borderwidth, bd, ...
Replace button widgets with either Button (preferable) or ttk::button. The same is true for oder kind of widgets (frame, etc. ...).

For Contributed widgets, replace all occurrences of:

Widget::getoption with Widget::getMegawidgetOption
and [Widget::theme] with [BWidget::using ttk]

Make sure to use predefined colorcodes, rather than color declarations:

{-foreground Color "SystemButtonText" 0}
{-background Color "SystemButtonFace" 0}
{-activeforeground Color "SystemButtonText" 0}
{-activebackground Color "SystemButtonFace" 0}
{-disabledforeground Color "SystemDisabledText" 0}
{-troughcolor Color "SystemScrollbar" 0}
As long as a frame doesn't have a decoration (rounded frame or whatever,...) the only difference is that a ttk::frame changes its background color immediately when a ttk style change takes place.

Using both types of frame objects at the same time (tk + ttk), some additional logic is needed for synchronizing colors, etc...

Typically, this is done via binding to the <<ThemeChanged>> virtual event.

No question anyway, if we use Bwidget's Frame object!
Minimum tile version is tile 0.8!
As there are changes between 0.7 and 0.8 which would cause
quite a lot of additional coding such as:

```tcl
if { [info commands ::ttk::style] ne "" } {
    set styleCmd ttk::style
} else { set styleCmd style }
```

it is required to have at least tile 0.8!
Simpler is easier: Update to tile 0.8, otherwise you won't get it!
Regarding Bwidgets future development, what is planned so far:

Font management:
Additional functionality allowing to “register” user defined fonts which can then be easily increased/decreased by the user (like in modern browsers “ctrl+/ctrl-”).

Additional widgets:
Imageflow ?

Additonal demo applications + improved theme support.
Attached I would like to offer a collection of themes gathered from the web and adopted slightly for base64 support.

**Benefits:**

To prove imperformance when calling up an application,

To establish a convenient method to ship images together within example code.

See as well:

http://wiki.tcl.tk/24595themes.tar.gz
Thank you very much for your attention and the idealism to contribute to the open source idea!

Johann Oberdorfer