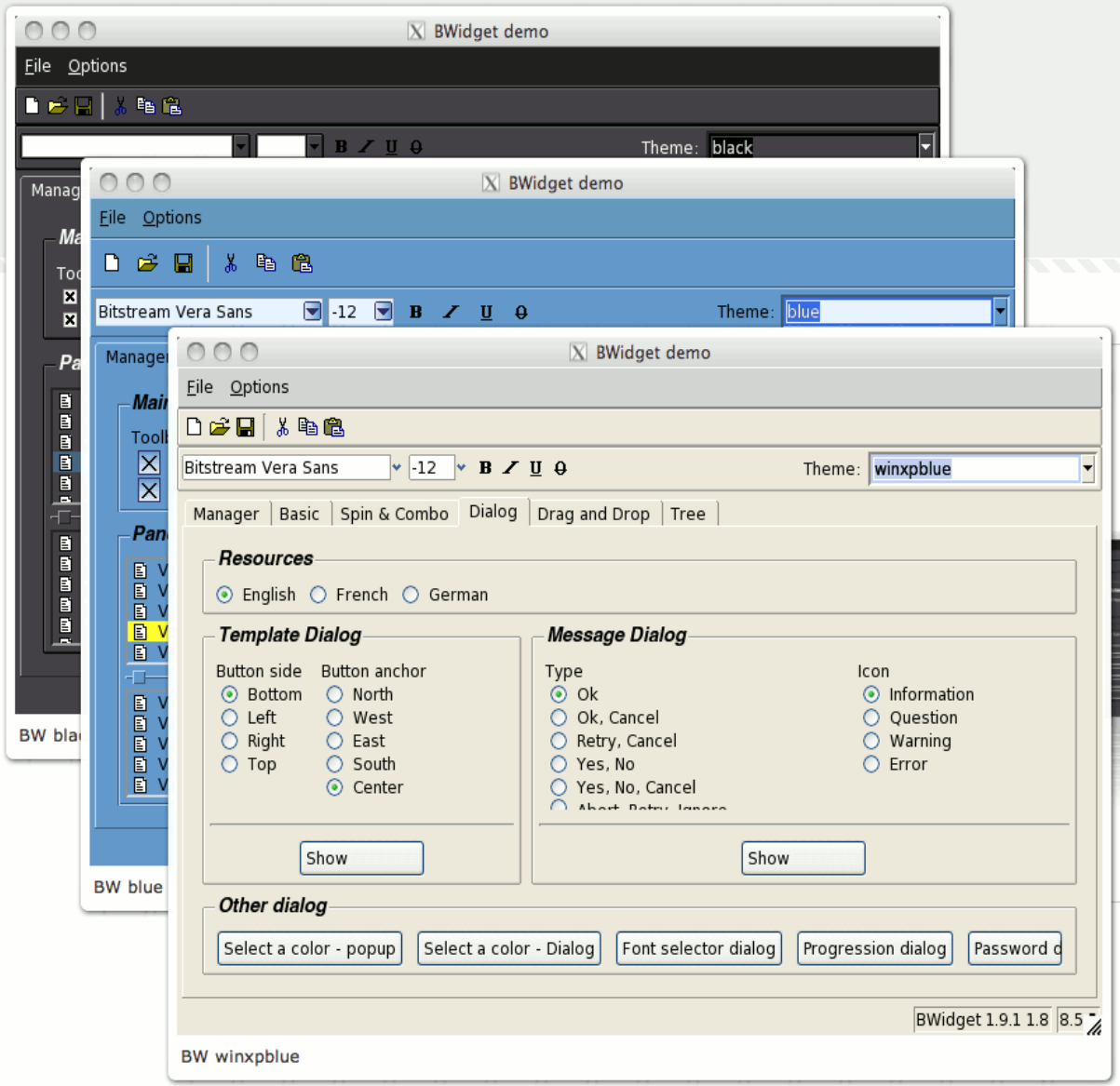
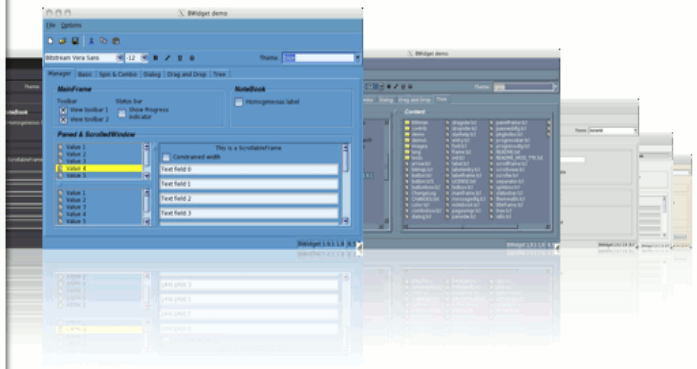
Welcome!

OBERDORFER
Konstruktion | CAD
Software-Entwicklung



BWidget goes Tile

Autor:Johann Oberdorfer
With special thanks to: Harald Oehlmann



Tcl/Tk The Choice for Rapid Application Development

EUROPEAN TCL/TK 2010
USER MEETING
Strasbourg (France)
4-5 June

Introduction

Bwidget is a script-only package for tcl/tk offering GUI elements and it's 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.

http://www.activestate.com/activetcl

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 developement state.

Package usage scenario:

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.

Identifying deprecated functionality :

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” 'll be supported for compatibility reasons as usual.

Legend:

* ... deprecated

+ ... bwidget offers more options than ttk

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

Identifying deprecated functionality :

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

Summary: 32 widgets, 9 of them can be possibly replaced by their ttk/tile counterparts.

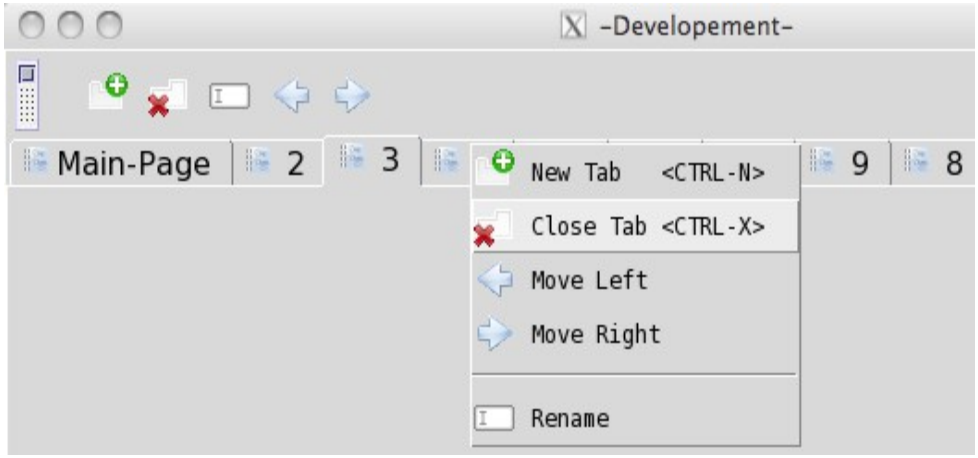
Deprecated functionality - case study:

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



Implementation Details V1.9.1

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:

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

Style related Argument usage:

```
# 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 avaliable for tk
#             |         |
#             |         if specified, BW tries to emulate OS color scheme,
#             |         a specific color schema 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 specifing additional ttk theme packages
```

Style related – further explanations:

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
```

Where does the style come from ?

As styles are not support 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:

```
BWidget::use \
-package ttk -style winxpblue -setoptdb 1 \
-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"
```

Color Mapping:

In order to synchronize both kind of widgets (tk & ttk), a minimum set of keywords is used to achive this behaviour.

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.

```
SystemWindow         -background
SystemWindowFrame    -background
SystemWindowText     -foreground
SystemButtonText     -activeforeground
SystemButtonFace     -activebackground
SystemDisabledText   -disabledforeground
SystemHighlight      -selectbackground
SystemHighlightText  -selectforeground
SystemMenu           -background
SystemMenuText       -foreground
SystemScrollbar      -troughcolor
```

Color Mapping – some good reasons:

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

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

    set colors(style) "aquativo"
    array set colors {
      SystemWindow          "#EDF3FE"    SystemHighlight      "RoyalBlue"
      SystemWindowFrame     "White"      SystemHighlightText  "White"
      SystemWindowText      "Black"      SystemMenu           "LightGrey"
      SystemButtonFace      "#fafafa"    SystemMenuText       "Black"
      SystemButtonText      "Black"      SystemScrollbar      "White"
      SystemDisabledText    "#fafafa"
    }
}
```

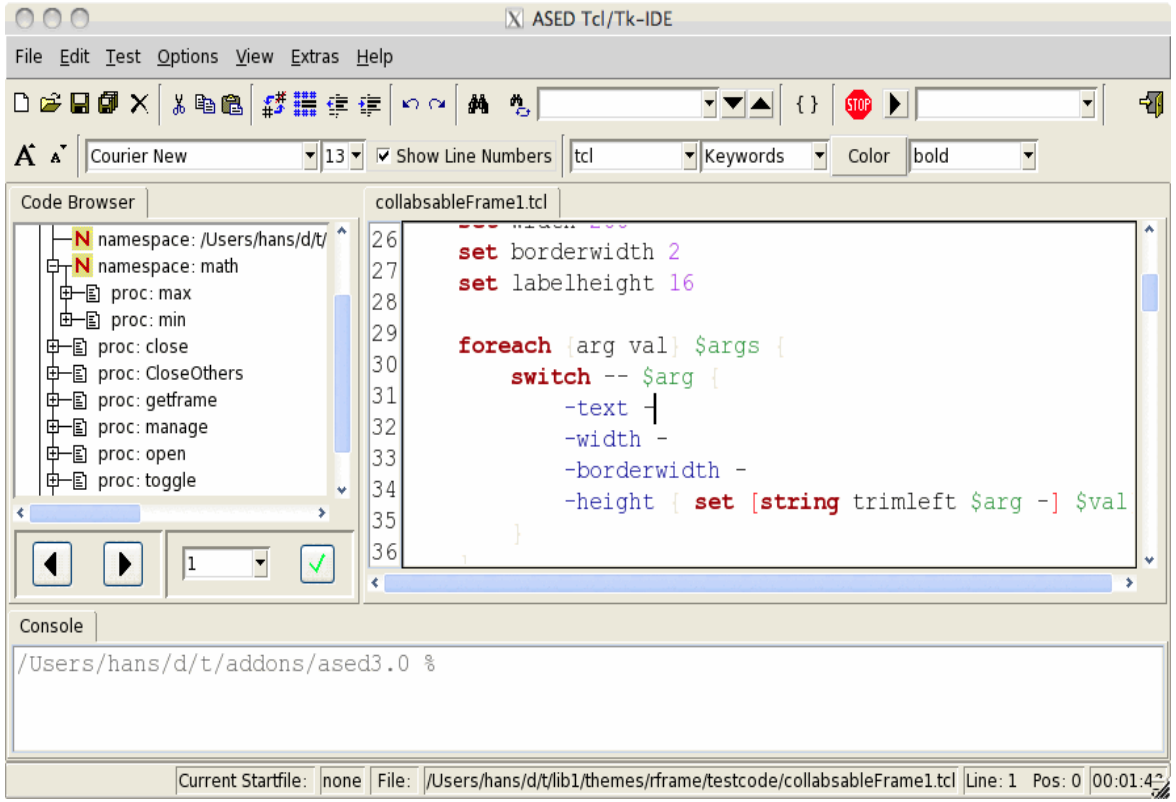
Migrating to tile – a practical example:

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:



Migrating to tile – a practical example:

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 occurences 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}
```

frame versus ttk::frame ?

As long as a frame doesn't have a decoration (rounded frame or whatever,...) the only difference is that a ttk::frame changes it's background color immediately when a ttk style change takes place.

Using both types of frame objects at the same tame (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!

What min. Tile version do we need?

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:

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

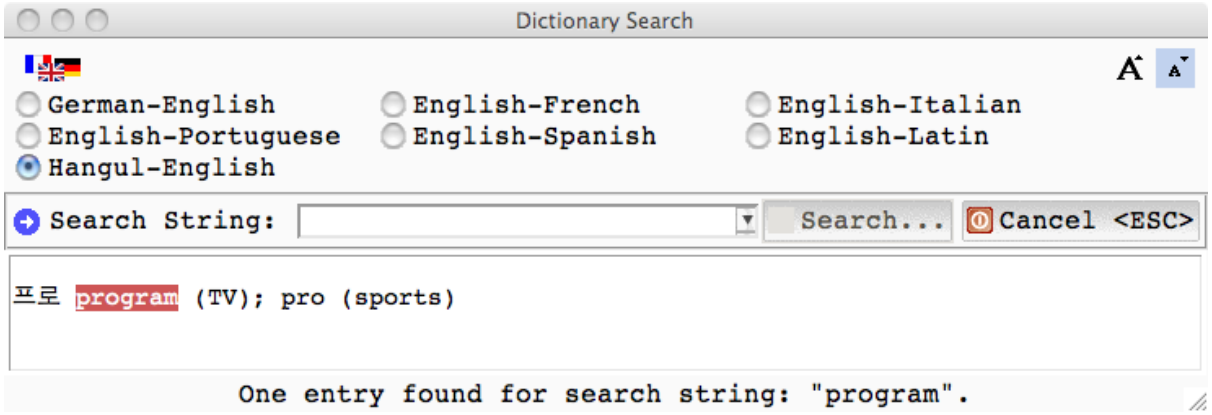
it is required to have at least tile 0.8!

Simpler is easier: Update to tile 0.8, otherwise you won't get it!

Finally – What comes next?

Regarding Bwidgets future developement, what is planned so far:

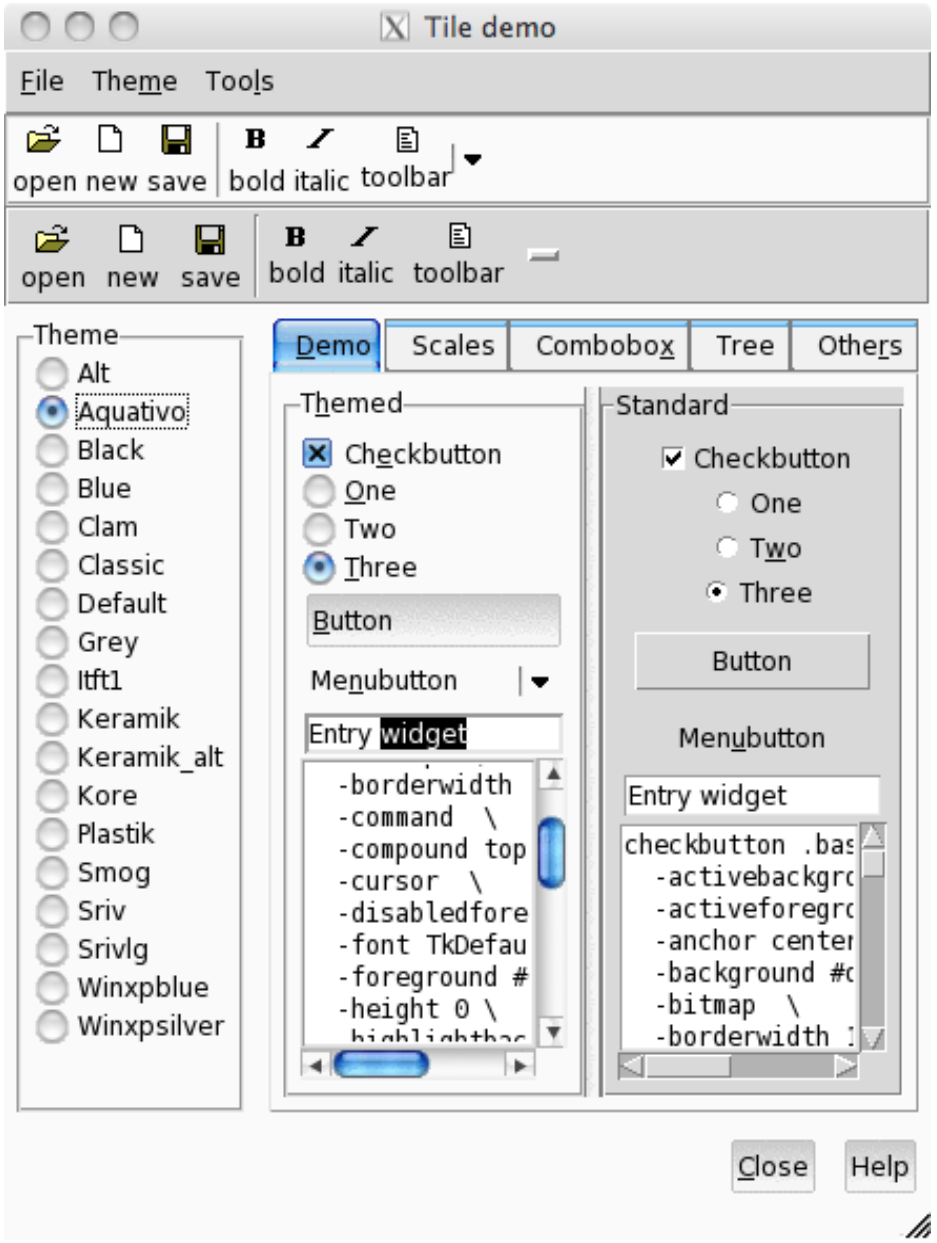
Font management:
Additional functionality allowing to “register” user defined fonts which can then be easily increased/decreased by the user (like in modern browsers “ctrl+/ctrl-”).



Additional widgets:
Imageflow ?

Additonal demo applications + improved theme support.

Extra – unified theme packages:



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/24595
themes.tar.gz

End of the presentation.

Thank you very much for your attention and the idealism
to contribute to the open source idea!

Johann Oberdorfer

OBERDORFER
Konstruktion | CAD
Software-Entwicklung