Tablelist in KDE Look

by

Csaba Nemethi

csaba.nemethi@t-online.de
http://www.nemethi.de

INFOSYS GmbH, Unterhaching

csaba.nemethi@infosys-online.de
http://www.infosys-online.de

Contents

1. A Quick Tablelist Overview

The Tablelist package (see *http://www.nemethi.de*) contains:

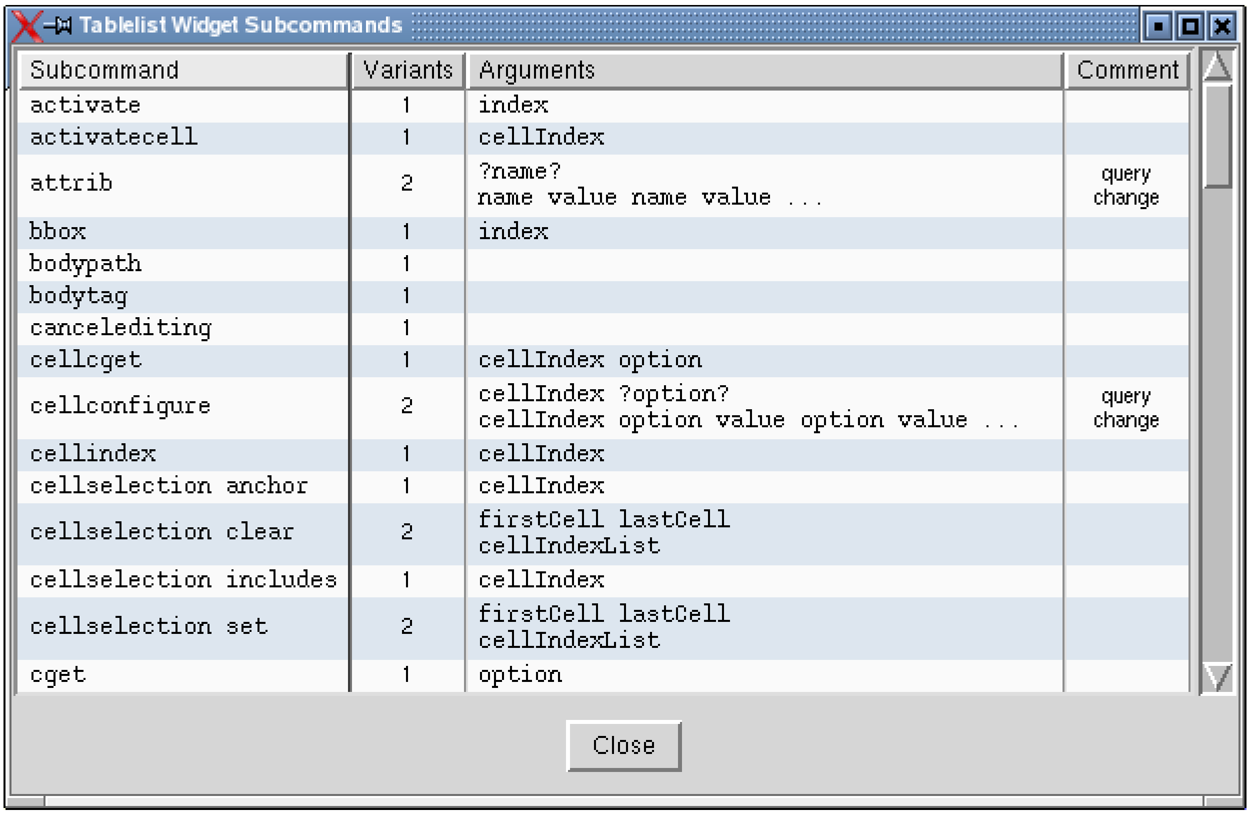
- the implementation of the tablelist mega-widget (in pure Tcl/Tk code), including a general utility module for mega-widgets;
- seven demo scripts that create tablelist widgets in classical look;
- eight demo-scripts that create tablelist widgets in tile look (seven of them being tile-based counterparts of the above);
- a comprehensive Tablelist Programmer's Guide in HTML format;
- detailed reference pages in HTML format.

A tablelist widget is a multi-column listbox, supporting a large number of options and widget subcommands. Here are just a few of them:

- Static- and dynamic-width columns.
- The columns are, per default, resizable.
- Interactive switching between static and dynamic column widths via `Button-3` and `Shift-Button-3`.
- Supported column alignments: `left`, `right`, and `center`.
- The font, colors, text, and other options can be set individually for the columns (and their titles), rows, and cells.
- Support for column separators and row stripes.
- Support for hidden columns and rows.
- Newline characters in the elements give rise to multi-line cells.
- Support for embedded images in the cells and header labels, as well as for embedded windows in the cells.
- Built-in sort capability, with the aid of the `sort`, `sortbycolumn`, and `sortbycolumnlist` subcommands (the latter does multi-column sorting).
- Support for interactive (multi-column) sorting via `Button-1` (and `Shift-Button-1`).
- Besides the `-selectmode` option (with the values `single`, `browse`, `multiple`, and `extended`) there is also a `-selecttype` option, with the values `row` and `cell`.
- Support for moving a column or row programmatically or interactively (with the left mouse button).
- Support for interactive editing with a great variety of widgets from the Tk core and the packages tile, BWidget, Iwidgets, combobox, and Mentry.

2. Tablelist Widgets in Classical Look

Our first example displays the tablelist widget subcommands in a tablelist with single- and multi-line cells.



Tablelist Widget Subcommands

Subcommand	Variants	Arguments	Comment
activate	1	index	
activatecell	1	cellIndex	
attrib	2	?name? name value name value ...	query change
bbox	1	index	
bodypath	1		
bodytag	1		
cancelediting	1		
cellcget	1	cellIndex option	
cellconfigure	2	cellIndex ?option? cellIndex option value option value ...	query change
cellindex	1	cellIndex	
cellselection anchor	1	cellIndex	
cellselection clear	2	firstCell lastCell cellIndexList	
cellselection includes	1	cellIndex	
cellselection set	2	firstCell lastCell cellIndexList	
cget	1	option	

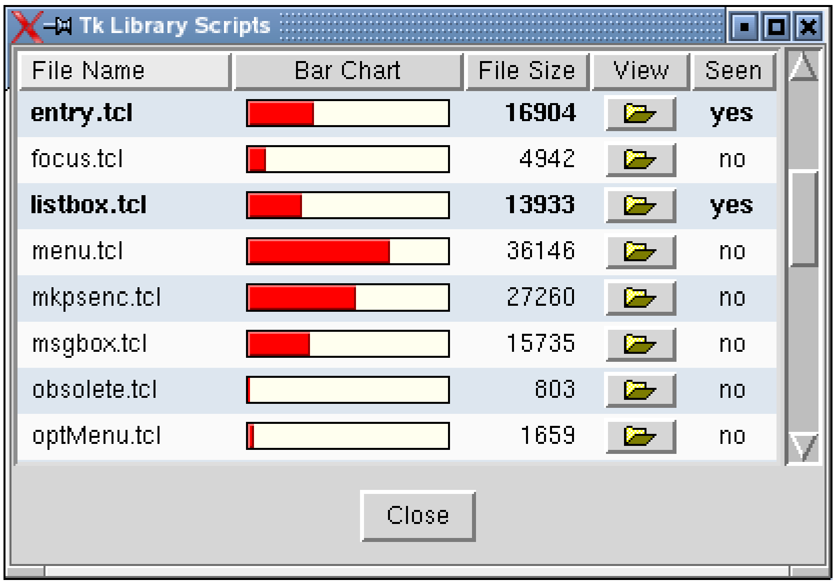
Close

The code is quite simple. When populating the widget, we insert newline characters into the strings that are to be displayed in two lines:

```
package require Tablelist

#
# The following optional command makes the use of a
# text widget as edit window much more user-friendly:
#
package require Wcb

wm title . "Tablelist Widget Subcommands"

#
# Add some entries to the Tk option database
#
set dirName [file dirname [info script]]
source [file join $dirName option.tcl]

#
# Create a vertically scrolled tablelist widget with 4
# dynamic-width columns and interactive sort capability
#
set tbl .tbl
set vsb .vsb
tablelist::tablelist $tbl \
    -columns {0 Subcommand left
```

```
              0 Variants   center
              0 Arguments  left
              0 Comment    center} \
    -font "Helvetica -10" -height 20 -setgrid no -showseparators yes \
    -titlecolumns 1 -protecttitlecolumns yes \
    -yscrollcommand [list $vsb set] -width 0
if {[$tbl cget -selectborderwidth] == 0} {
    $tbl configure -spacing 1
}
$tbl columnconfigure 0 -font "Courier -12"
$tbl columnconfigure 2 -font "Courier -12"
$tbl columnconfigure 3 -editable yes -editwindow text
scrollbar $vsb -orient vertical -command [list $tbl yview]

#
# Populate the tablelist widget
#
$tbl insert end \
    [list "activate" 1 "index"] \
    [list "activatecell" 1 "cellIndex"] \
    [list "attrib" 2 "?name?\nname value name value ..." "query\nchange"] \

    . . .

    [list "insertcolumnlist" 1 "columnIndex {width title ?alignment? \\
             width title ?alignment? ...}"] \

    . . .

set btn [button .btn -text "Close" -command exit]

. . .
```

Our second example is a slightly modified version of a demo script contained in the Tablelist distribution. It creates a tablelist widget whose items correspond to the Tk library scripts. The size of each file (in bytes) is not only displayed as a number, but is also illustrated with the aid of a frame with red background, created as a child of an embedded frame with ivory background. The files can be viewed by clicking on the corresponding embedded button widgets.


Tk Library Scripts

File Name	Bar Chart	File Size	View	Seen
entry.tcl		**16904**		**yes**
focus.tcl		4942		no
listbox.tcl		**13933**		**yes**
menu.tcl		36146		no
mkpsenc.tcl		27260		no
msgbox.tcl		15735		no
obsolete.tcl		803		no
optMenu.tcl		1659		no

First, we create and populate the tablelist widget:

```
package require Tablelist

wm title . "Tk Library Scripts"

#
# Add some entries to the Tk option database
#
set dirName [file dirname [info script]]
source [file join $dirName option.tcl]

#
# Create an image to be displayed in buttons embedded in a tablelist widget
#
set openImg [image create photo -file [file join $dirName open.gif]]

#
# Create a vertically scrolled tablelist widget with 5
# dynamic-width columns and interactive sort capability
#
set tbl .tbl
set vsb .vsb
tablelist::tablelist $tbl \
    -columns {0 "File Name" left
              0 "Bar Chart" center
              0 "File Size" right
              0 "View"      center
              0 "Seen"      center} \
    -font "Helvetica -12" -setgrid no -yscrollcommand [list $vsb set] -width 0
if {[$tbl cget -selectborderwidth] == 0} {
    $tbl configure -spacing 1
}
$tbl columnconfigure 0 -name fileName
$tbl columnconfigure 1 -formatcommand emptyStr -sortmode integer
$tbl columnconfigure 2 -name fileSize -sortmode integer
$tbl columnconfigure 4 -name seen
scrollbar $vsb -orient vertical -command [list $tbl yview]

proc emptyStr val { return "" }

#
# Populate the tablelist widget
#
cd $tk_library
set maxFileSize 0
foreach fileName [lsort [glob *.tcl]] {
    set fileSize [file size $fileName]
    $tbl insert end [list $fileName $fileSize $fileSize "" no]

    if {$fileSize > $maxFileSize} {
        set maxFileSize $fileSize
    }
}
```

In order to create the embedded windows, we have to implement the creation scripts for them:

```
#------------------------------------------------------------------------------
# createFrame
#
# Creates a frame widget w to be embedded into the specified cell of the
# tablelist widget tbl, as well as a child frame representing the size of the
# file whose name is diplayed in the first column of the cell's row.
#------------------------------------------------------------------------------
```

```
proc createFrame {tbl row col w} {

    . . .

}

#------------------------------------------------------------------------------
# createButton
#
# Creates a button widget w to be embedded into the specified cell of the
# tablelist widget tbl.
#------------------------------------------------------------------------------
proc createButton {tbl row col w} {
    set key [$tbl getkeys $row]
    button $w -image $::openImg -highlightthickness 0 -takefocus 0 \
              -command [list viewFile $tbl $key]
}

#------------------------------------------------------------------------------
# viewFile
#
# Displays the contents of the file whose name is contained in the row with the
# given key of the tablelist widget tbl.
#------------------------------------------------------------------------------
proc viewFile {tbl key} {
    set top .top$key
    if {[winfo exists $top]} {
        raise $top
        return ""
    }

    toplevel $top
    set fileName [$tbl cellcget k$key,fileName -text]
    wm title $top "File \"$fileName\""

    #
    # Create a vertically scrolled text widget as a child of the toplevel
    #
    set txt $top.txt
    set vsb $top.vsb
    text $txt -font "Courier -12" -setgrid yes -yscrollcommand [list $vsb set]
    scrollbar $vsb -orient vertical -command [list $txt yview]

    #
    # Insert the file's contents into the text widget
    #
    set chan [open $fileName]
    $txt insert end [read $chan]
    close $chan

    . . .

    #
    # Mark the file as seen
    #
    $tbl rowconfigure k$key -font "Helvetica -12 bold"
    $tbl cellconfigure k$key,seen -text yes
}
```

Having implemented the creation scripts for the frames and buttons, we can now use the `cellconfigure` subcommand to create these widgets as embedded windows:

```
#
# Create embedded windows in the columns no. 1 and 3
#
set rowCount [$tbl size]
for {set row 0} {$row < $rowCount} {incr row} {
    $tbl cellconfigure $row,1 -window createFrame
    $tbl cellconfigure $row,3 -window createButton
}
```

Now let's take a look at the file `option.tcl`, which is `source`d into both example scripts above:

```
#
# Get the current windowing system ("x11", "win32", or
# "aqua") and add some entries to the Tk option database
#
if {[tk windowingsystem] eq "x11"} {
    option add *Font                        "Helvetica -12"
    option add *selectBackground            #447bcd
    option add *selectForeground            white
    option add *Text.background             white
    option add *Text.foreground             black
}
option add *Tablelist.background        gray98
option add *Tablelist.foreground        black
option add *Tablelist.stripeBackground  #e0e8f0
option add *Tablelist.activeStyle       frame
option add *Tablelist.setFocus          yes
option add *Tablelist.setGrid           yes
option add *Tablelist.movableColumns    yes
option add *Tablelist.labelCommand      tablelist::sortByColumn
option add *Tablelist.labelCommand2     tablelist::addToSortColumns
```

The main goal of the option database settings contained in this file is to improve the overall appearance and behavior of the widgets used in both example scripts.

The `*Font` option is specified with a capital "F", because we want to set both the `-font` and `-labelfont` tablelist options, both belonging to the database class `Font`. We specify this option as well as some others not only for tablelist widgets, because they are common widget options, whose values should be the same for all widgets if not explicitly overridden in the application.

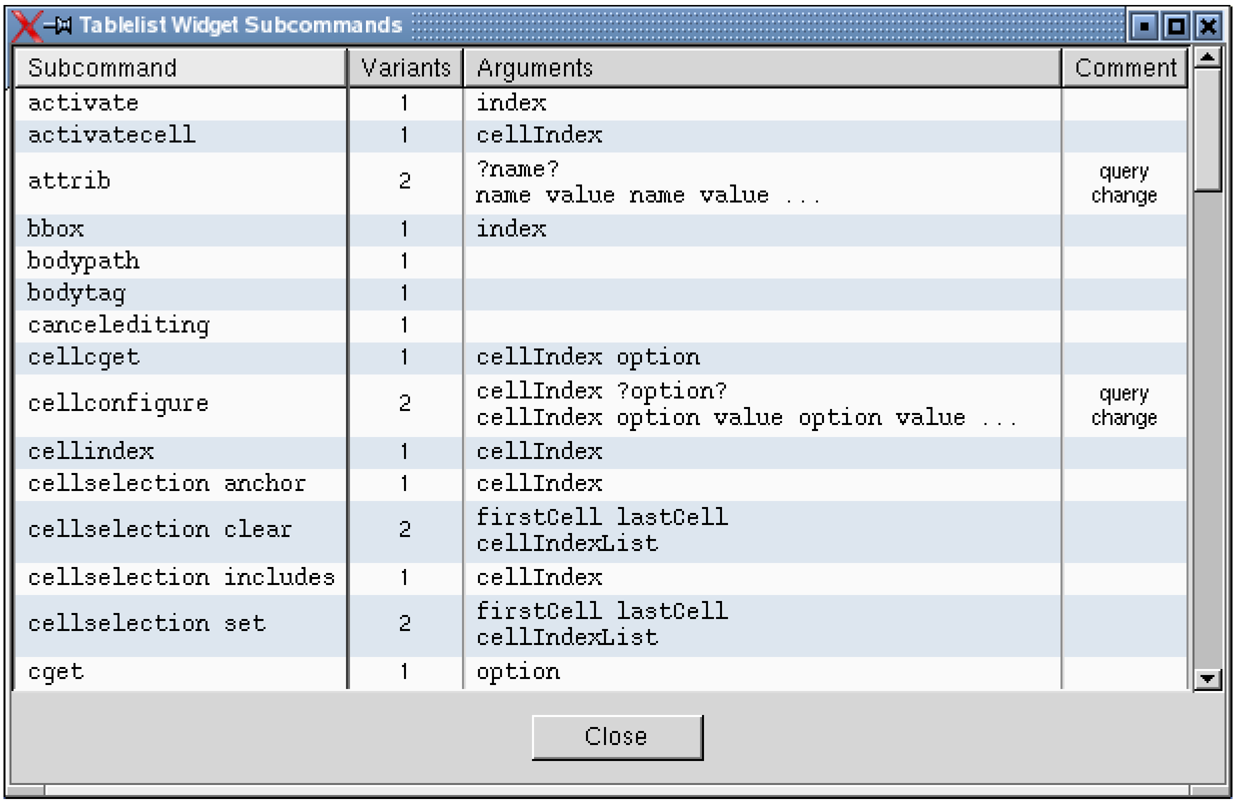
The procedures specified in the last two option database settings enable the interactive multi-column sorting via `Button-1` and `Shift-Button-1`; they invoke the `sortbycolumn` and `sortbycolumnlist` subcommands, mentioned in Section 1.

3. Tablelist Widgets in Tile Look

Here is the tile-based version of our first example script:

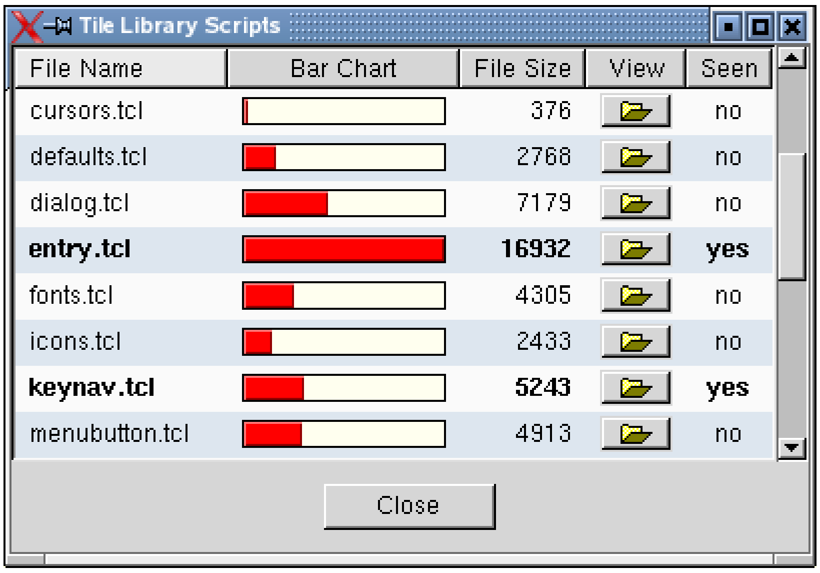
```
package require Tablelist_tile

#
# The following optional command makes the use of a
# text widget as edit window much more user-friendly:
#
package require Wcb

wm title . "Tablelist Widget Subcommands"

#
# Add some entries to the Tk option database
#
set dirName [file dirname [info script]]
source [file join $dirName option_tile.tcl]

#
# Work around the improper appearance of the tile scrollbars in the aqua theme
#
if {$tile::currentTheme eq "aqua"} {
    interp alias {} ttk::scrollbar {} ::scrollbar
}

#
# Improve the window's appearance by using a tile frame as a
# container for the other widgets; set -padding -2 to work
# around a tile bug in the themes winnative and xpnative
#
set f [ttk::frame .f -padding -2]

#
# Create a vertically scrolled tablelist widget with 4
# dynamic-width columns and interactive sort capability
#
set tbl $f.tbl
set vsb $f.vsb
tablelist::tablelist $tbl \
    -columns {0 Subcommand left
              0 Variants   center
              0 Arguments  left
              0 Comment    center} \
    -font "Helvetica -10" -height 20 -setgrid no -showseparators yes \
    -titlecolumns 1 -protecttitlecolumns yes \
    -yscrollcommand [list $vsb set] -width 0
if {[$tbl cget -selectborderwidth] == 0} {
    $tbl configure -spacing 1
}
$tbl columnconfigure 0 -font "Courier -12"
$tbl columnconfigure 2 -font "Courier -12"
$tbl columnconfigure 3 -editable yes -editwindow text
ttk::scrollbar $vsb -orient vertical -command [list $tbl yview]

#
# Populate the tablelist widget
#
$tbl insert end \
    [list "activate" 1 "index"] \
    [list "activatecell" 1 "cellIndex"] \
    [list "attrib" 2 "?name?\nname value name value ..." "query\nchange"] \

    . . .

    [list "insertcolumnlist" 1 "columnIndex {width title ?alignment? \\
             width title ?alignment? ...}"] \

    . . .

set btn [ttk::button $f.btn -text "Close" -command exit]

. . .
```

In the file `option_tile.tcl`, which is `source`d into our example script, the theme is set to `alt`. The resulting window has a nice theme-specific appearance:



Subcommand	Variants	Arguments	Comment
activate	1	index	
activatecell	1	cellIndex	
attrib	2	?name? name value name value ...	query change
bbox	1	index	
bodypath	1		
bodytag	1		
cancelediting	1		
cellcget	1	cellIndex option	
cellconfigure	2	cellIndex ?option? cellIndex option value option value ...	query change
cellindex	1	cellIndex	
cellselection anchor	1	cellIndex	
cellselection clear	2	firstCell lastCell cellIndexList	
cellselection includes	1	cellIndex	
cellselection set	2	firstCell lastCell cellIndexList	
cget	1	option	

The tile-based version of our second example script contains similar changes. In addition, it displays the tile library scripts instead of the Tk ones. Here is a screenshot of the resulting window:



File Name	Bar Chart	File Size	View	Seen
cursors.tcl		376	📂	no
defaults.tcl		2768	📂	no
dialog.tcl		7179	📂	no
entry.tcl		**16932**	📂	**yes**
fonts.tcl		4305	📂	no
icons.tcl		2433	📂	no
keynav.tcl		**5243**	📂	**yes**
menubutton.tcl		4913	📂	no

The tile-based versions of both example scripts make use of the option database settings contained in the file `option_tile.tcl`:

```
#
# Get the current windowing system ("x11", "win32",
# or "aqua") and set the theme if needed
#
if {[tk windowingsystem] eq "x11"} {
    lappend auto_path ~/tcltk/lib       ;# contains all non-built-in themes
    tile::setTheme alt
}

#
# Populate the array tablelist::themeDefaults
# and add some entries to the Tk option database
#
tablelist::setThemeDefaults
if {$tile::currentTheme ne "aqua"} {
    option add *selectBackground  $tablelist::themeDefaults(-selectbackground)
    option add *selectForeground  $tablelist::themeDefaults(-selectforeground)
    option add *selectBorderWidth $tablelist::themeDefaults(-selectborderwidth)
    option add *Text.background   $tablelist::themeDefaults(-background)
    option add *Text.foreground   $tablelist::themeDefaults(-foreground)
}
if {$tablelist::themeDefaults(-stripebackground) eq ""} {
    option add *Tablelist.background            gray98
    option add *Tablelist.foreground            black
    option add *Tablelist.stripeBackground      #e0e8f0
}
option add *Tablelist.activeStyle       frame
option add *Tablelist.setFocus          yes
option add *Tablelist.setGrid           yes
option add *Tablelist.movableColumns    yes
option add *Tablelist.labelCommand      tablelist::sortByColumn
option add *Tablelist.labelCommand2     tablelist::addToSortColumns
```

The procedure `tablelist::setThemeDefaults` populates the array `tablelist::themeDefaults` with the theme-specific default values of a series of Tablelist configuration options. This procedure is invoked by the Tablelist code automatically when the first tablelist widget is createad or a `<<ThemeChanged>>` virtual event is received by a tablelist widget. In the code above we call it explicitly and use the values written by it into the array `tablelist::themeDefaults` for some common option database settings, in order to make sure that classical Tk widgets, e.g., text, will have a theme-specific appearance, just like the tile widgets. We exclude the `aqua` theme, because on Mac OS X Aqua the default values of the `-selectbackground` and `-selectforeground` options for tablelist widgets are different from those for Tk core widgets.

Notice that we didn't remove the three lines containing the option database settings for the tablelist `background`, `foreground`, and `stripeBackground` options. The reason is that we want to have stripes in our tablelist widgets, but in most themes the default value of the `-stripebackground` option is an empty string. For this case we define reasonably looking values for both that alternate color and the ones used for the background and foreground of the tablelist widgets. Currently the only theme providing a nonempty default value for the `-stripebackground` option is `tileqt`.

4. The TileQt Extension for the KDE Desktop

Instead of `alt` we can take any other available theme. The following simple script lists all of them:

```
package require tile

#
# Print the list of the available themes; the non-built-in
# ones are located in the directory ~/tcltk/lib
#
lappend auto_path ~/tcltk/lib
eval [package unknown] Tcl [package provide Tcl]
puts "Available themes:  [lsort -dictionary [tile::availableThemes]]"

exit
```

On my Linux notebook I get the the following output:

```
Available themes:  alt Aquativo blue clam classic default keramik kroc plastik
sriv srivlg step tileqt winxpblue
```

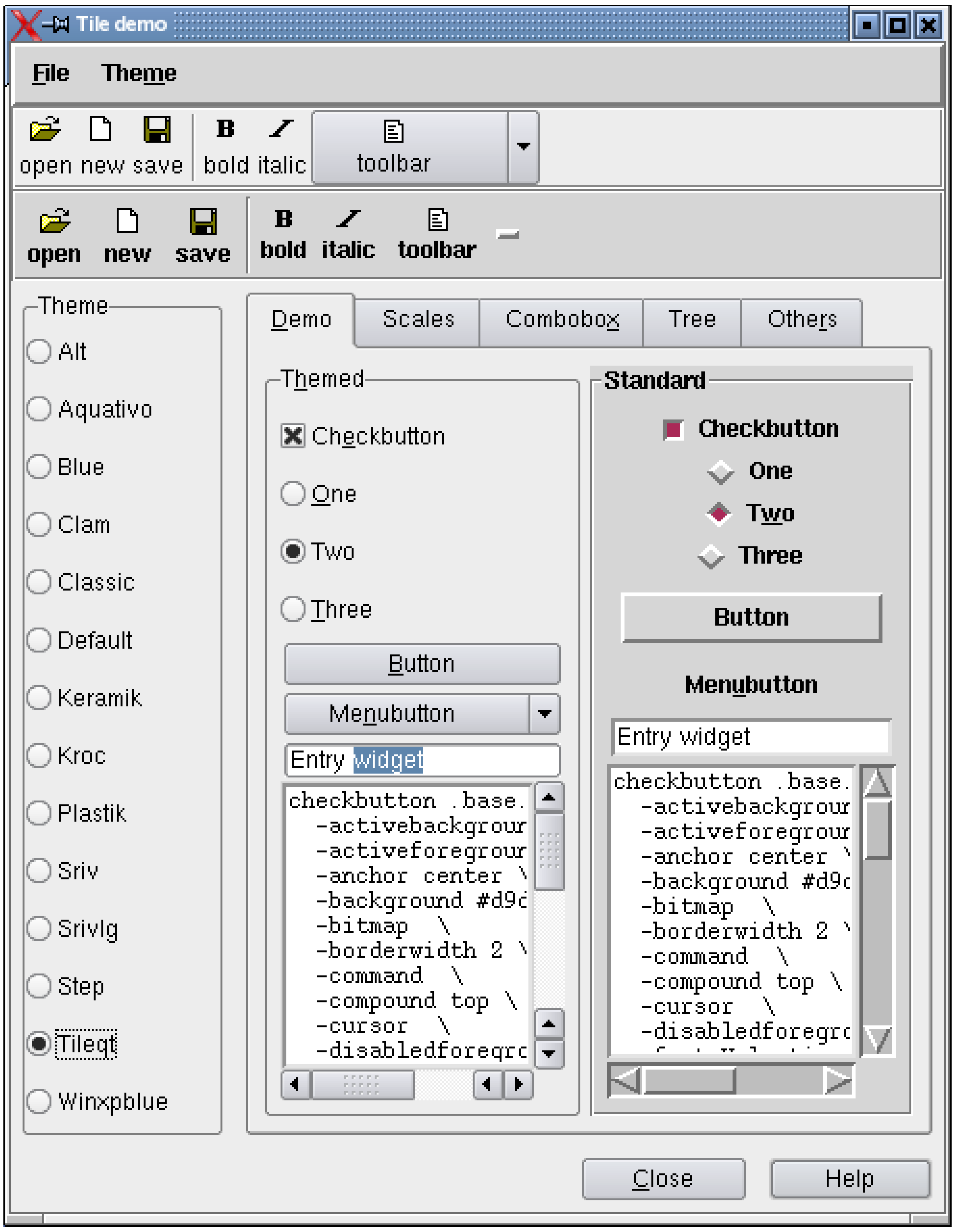
By experimenting with various widgets using these themes, it quickly turns out that `tileqt` is the only theme that gives rise to a GUI appearance that fits well into a KDE environment. This theme, developed by Georgios Petasis, makes use of the Qt class library to draw the tile widgets, just like the KDE components. The resulting widgets have both the layouts and colors corresponding to the current Qt style and KDE color scheme. The download location of the excellent TileQt extension is:

http://www.ellogon.org/petasis/index.php?option=com_content&task=view&id=24&Itemid=40

Although the Qt class library is a multi-platform toolkit, TileQt was explicitly developed for the KDE system. It makes use of the Inter-Client Communication Conventions (ICCC) protocol to communicate with KDE. When loaded into an interpreter, TileQt creates an invisible `QWidget` object and sets the `"KDE_DESKTOP_WINDOW"` property on it, to make sure that it will be notified about Qt style and color scheme changes (which are usually done with the aid of the KDE Control Center). Whenever such a change takes place, KDE sends a `ClientMessage` event to this window, having the `message_type` member set to the atom corresponding to the

"KIPC_COMM_ATOM" property. TileQt receives this event with the aid of a Tk generic handler registered for the interpreter in question, and performs the necessary actions to redraw the tile widgets. As a result, the widget layouts and colors are adapted to the new style and color scheme, respectively.

Here is a screenshot displaying the window produced by a slightly modified version of the well-known tile distribution script `demo.tcl`, where the theme is set to `tileqt` and both the style and color scheme have the value `Plastik` (which is the default in newer KDE versions):

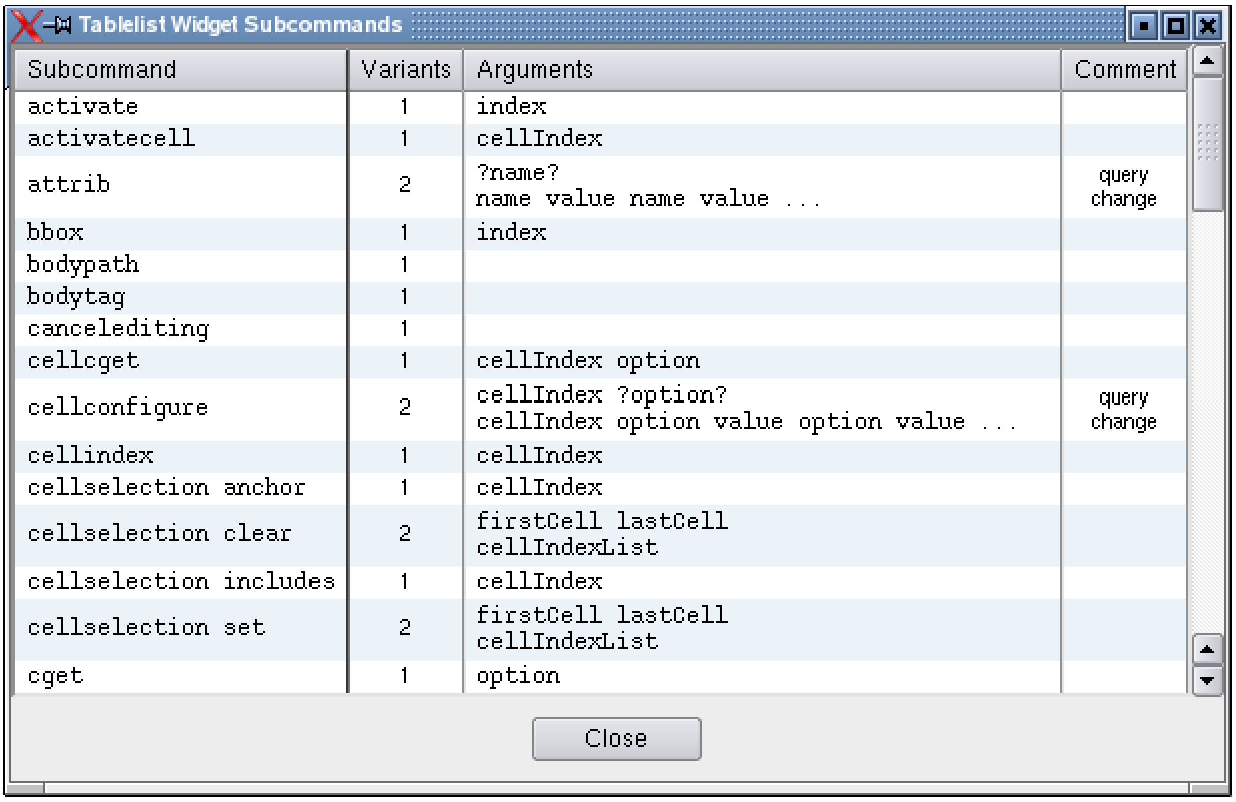


```
checkbutton .base.
  -activebackgrour
  -activeforegrour
  -anchor center \
  -background #d9c
  -bitmap  \
  -borderwidth 2 \
  -command  \
  -compound top \
  -cursor  \
  -disabledforegrc
```

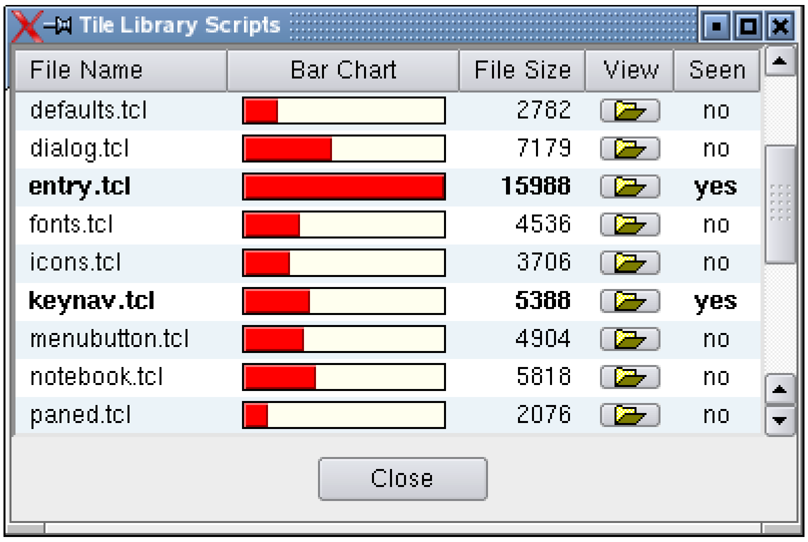
```
checkbutton .base.
  -activebackgrour
  -activeforegrour
  -anchor center \
  -background #d9c
  -bitmap  \
  -borderwidth 2 \
  -command  \
  -compound top \
  -cursor  \
  -disabledforegrc
```

5. Tablelist Widgets in KDE Look

The two pictures below are screenshots displaying the windows created by the two demo scripts discussed in the preceding sections, where the theme is set to `tileqt` and both the Qt style and color scheme have the value `Plastik`:



Subcommand	Variants	Arguments	Comment
activate	1	index	
activatecell	1	cellIndex	
attrib	2	?name? name value name value ...	query change
bbox	1	index	
bodypath	1		
bodytag	1		
cancelediting	1		
cellcget	1	cellIndex option	
cellconfigure	2	cellIndex ?option? cellIndex option value option value ...	query change
cellindex	1	cellIndex	
cellselection anchor	1	cellIndex	
cellselection clear	2	firstCell lastCell cellIndexList	
cellselection includes	1	cellIndex	
cellselection set	2	firstCell lastCell cellIndexList	
cget	1	option	



File Name	Bar Chart	File Size	View	Seen
defaults.tcl	🟥	2782	📂	no
dialog.tcl	🟥	7179	📂	no
entry.tcl	🟥	**15988**	📂	**yes**
fonts.tcl	🟥	4536	📂	no
icons.tcl	🟥	3706	📂	no
keynav.tcl	🟥	**5388**	📂	**yes**
menubutton.tcl	🟥	4904	📂	no
notebook.tcl	🟥	5818	📂	no
paned.tcl	🟥	2076	📂	no

By changing the Qt style and/or the color scheme, the appearance of the above windows will change accordingly. This is achieved by using some commands provided by the TileQt extension to adapt the default values of a series of configuration options to the currently used style and color scheme, and by a lot of additional work put into Tablelist to cope with the many special cases where the means provided by TileQt are not sufficient for all purposes. The rest of the paper describes the main aspects of the implementation of Tablelist's support for the `tileqt` theme.

As already mentioned in Section 3, Tablelist invokes the procedure `tablelist::setThemeDefaults` when the first tablelist widget is createad or a `<<ThemeChanged>>` virtual event is received by a tablelist widget. This virtual event is generated by the tile widget engine whenever the theme is set via `style theme use` (which in turn is invoked by `tile::setTheme`). However, if the current theme is `tileqt` and the Qt style or KDE color scheme is changed then tile doesn't send this event. As described in the preceding section, the TileQt extension is notified of such changes and handles them by redrawing the tile widgets. Besides doing this, TileQt also generates a `<<ThemeChanged>>` virtual event, which in turn will fire an invocation of the `tablelist::setThemeDefaults` procedure.

This procedure populates the array `tablelist::themeDefaults` with theme-specific default values of the following Tablelist configuration options: `-background`, `-foreground`, `-disabledforeground`, `-stripebackground`, `-selectbackground`, `-selectforeground`, `-selectborderwidth`, `-font`, `-labelbackground`, `-labelforeground`, `-labelfont`, `-labelborderwidth`, `-labelpady`, `-arrowcolor`, and `-arrowstyle`. In addition, it sets some other array elements to theme-specific default values of the label background and foreground colors in `disabled`, `active`, and `pressed` states. The label colors are needed for handling sort arrows and images with transparent background placed into the column labels.

When a tablelist widget receives the `<<ThemeChanged>>` virtual event, Tablelist invokes the procedure `tablelist::setThemeDefaults` to retrieve the new default values of the above configuration options, and then reconfigures the widget by updating those options that were not set explicitly to values different from the corresponding defaults.

If the current theme is `tileqt` then the default values mentioned above are not only theme-specific, but depend also on the Qt style and KDE color scheme. Tablelist retrieves most default color values by invoking the TileQt command `tile::theme::tileqt::currentThemeColour`, which accepts various arguments specifying which color is meant. While this works as expected for most colors, there are quite a few styles (like "Baghira", "BlueCurve", "HighColor Classic", "Keramik", "Light Style, 3rd revision", "Phase", "Plastik", "Platinum", "QtCurve", "ThinKeramik", etc.) in which the background color of the labels in `normal` and/or `pressed` states cannot be retrieved by calling any TileQt command. For these styles, Tablelist uses empirically measured values, which depend not only on the Qt style but also on the KDE color scheme. In some other styles (like "Acqua", "KDE_XP", "Lipstik", "Marble", "RISC OS", "System-Series", "System++", etc.), the label colors are independent of the color scheme, but, again, Tablelist can only use empirically measured values for them. Paul Obermeier's excellent `poImgview` application (see *http://www.poSoft.de*) helped me a lot getting these colors, which in many cases are just median pixel values.

The default value of the `-stripebackground` option cannot be retrieved by calling any TileQt command either, but, fortunately, KDE saves this color in a `kdeglobals` configuration file, and Tablelist reads it from that file, which is located in one of the standard KDE directories.

Finally, determining the default values of the `-arrowcolor` and `-arrowstyle` options, which, again, depend on the currently used Qt style, was made by looking at the sort arrows displayed by standard KDE applications in various styles. Georgios Petasis used the same method to determine the scrollbar layouts, which are style-specific, too.

From all the above it follows that it is nearly impossible to cope with the great variety of available Qt styles and KDE color schemes. Nevertheless, Tablelist supports all the built-in ones, along with a number of additional styles and color schemes. The following code snippet, taken from the Tablelist distribution file `tablelistThemes.tcl`, lists the currently supported Qt styles and KDE color schemes:

```
#------------------------------------------------------------------------------
# tablelist::tileqtTheme
#
# Tested with the following Qt styles:
#
#   Acqua              KDE_XP                     Motif Plus     SGI
#   B3/KDE             Keramik                    MS Windows 9x  System-Series
#   Baghira            Light Style, 2nd revision  Phase          System++
#   CDE                Light Style, 3rd revision  Plastik        ThinKeramik
#   HighColor Classic  Lipstik                    Platinum
#   HighContrast       Marble                     QtCurve
#   KDE Classic        Motif                      RISC OS
#
# Supported color schemes:
#
#   Aqua Blue                     Ice (FreddyK)      Point Reyes Green
#   Aqua Graphite                 KDE 1              Pumpkin
#   Atlas Green                   KDE 2              Redmond 2000
#   BeOS                          Keramik            Redmond 95
#   Blue Slate                    Keramik Emerald    Redmond XP
#   CDE                           Keramik White      Solaris
#   Dark Blue                     Lipstik Noble      Storm
#   Desert Red                    Lipstik Standard   SuSE, old & new
#   Digital CDE                   Lipstik White      SUSE-kdm
#   EveX                          Media Peach        System
#   High Contrast Black Text      Next               Thin Keramik, old & new
#   High Contrast Yellow on Blue  Pale Gray          Thin Keramik II
#   High Contrast White Text      Plastik
#------------------------------------------------------------------------------
```