# Filling a Gap in Tk with Tablelist

by

**Csaba Nemethi**

*csaba.nemethi@t-online.de*

## Contents

# 1 Looking for a Multi-Column Listbox

## 1.1 The requirements

Every really professional toolkit for building graphical user interfaces should contain a multi-column listbox widget fulfilling the following conditions:

- It should be easy to install and use;
- It should behave just like a Tk core listbox;
- It should support left- and right-aligned as well as centered columns;
- It should have support for both static- and dynamic-width columns;
- A single mouse click should suffice to make the width of a column dynamic, i.e., just large enough to hold all the elements in the column, including the header;
- The columns should be, per default, resizable (with the aid of the left mouse button);
- It should have built-in support for sorting the items both globally and by an arbitrary column;
- The sort operation should preserve the selection information and the active item;
- When sorting the items by a column, the sorting order should, per default, be visualized with the aid of an arrow placed into the corresponding column label;
- It should support programmatic updating of row and cell contents;
- It should be possible to configure the colors and fonts of the columns, rows, cells, and header labels individually;
- It should provide support for inserting images into the cells and header labels;
- It should support stretching of an arbitrary set of columns in order to eliminate the blank space that might appear at the right of the widget;
- It should be possible to define callbacks for the `activate`, `selection set`, and `selection clear` commands;
- It should support vertical separators and horizontal stripes, to improve the readability of the items.

## 1.2 The candidates

Unfortunately, the Tk core doesn’t include a multi-column listbox. The well-known **table** widget of the **TkTable** compiled extension by Jeffrey Hobbs behaves more like a spreadsheet than like a listbox, and fails to fulfil several of the above conditions. Bryan Oakley’s **mclistbox** is based on normal listbox widgets, which prevents it from supporting right-aligned or centered columns as well as images. The **BLT** compiled extension by George A. Hewlett contains a **hiertable** widget, which has many of the above features, but still misses some of them and is not quite easy to install and use. Also the **rtl_mlistbox** widget of the **Runtime Library** by Patzschke + Rasp Software AG and the **listcontrol** component of the **mkWidgets** package by Michael Kraus fail to fulfil several of the conditions mentioned above.

These facts determined me nearly two years ago to develop a new multi-column listbox, called **tablelist**. It has all of the above features and is really very easy to install and use, as confirmed in e-mails received from dozens of satisfied users. Many of them have also given me valuable suggestions for improving the look & feel, behavior, and performance of the widget. This feedback has been the main motivation for my efforts aimed to make this widget a professional Tk extension, useful to everyone looking for a versatile and user-friendly multi-column listbox.

# 2 Tablelist Overview

This chapter is taken from the *Tablelist Programmer's Guide*, which is part of the detailed documentation belonging to the distribution of the current version 2.5 of the Tablelist package.

## 2.1 What is Tablelist?

Tablelist is a library package for Tcl/Tk version 8.0 or higher, written in pure Tcl/Tk code. It contains:

- the implementation of the tablelist mega-widget, including a general utility module for mega-widgets;
- a demo script containing a useful procedure that displays the configuration options of an arbitrary widget in a tablelist;
- a second demo script, containing a simple widget browser based on a tablelist;
- a third demo script, showing several ways to improve the appearance of a tablelist widget;
- this tutorial;
- reference pages in HTML format.

A tablelist widget is a multi-column listbox. The width of each column can be dynamic (i.e., just large enough to hold all its elements, including the header) or static (specified in characters or pixels). The columns are, per default, resizable. The alignment of each column can be specified as `left`, `right`, or `center`.

The columns, rows, and cells can be configured individually. Several of the global and column-specific options refer to the headers, implemented as label widgets. For instance, the `-labelcommand` option specifies a Tcl command to be invoked when mouse button 1 is released over a label. The most common value of this option is `tablelist::sortByColumn`, which sorts the items based on the respective column.

The Tcl command corresponding to a tablelist widget is very similar to the one associated with a normal listbox. There are column-, row-, and cell-specific counterparts of the `configure` and `cget` subcommands (`columnconfigure`, `columncget`, ...). They can be used, among others, to insert images into the cells and the header labels. The `index` and `nearest` command options refer to the rows, but similar subcommands are provided for the columns and cells (`columnindex`, `nearestcolumn`, ...). The items can be sorted with the `sort` and `sortbycolumn` command options.

The bindings defined for the body of a tablelist widget make it behave just like a normal listbox. This includes the support for the virtual event `<<ListboxSelect>>`, when using Tk version 8.1 or higher. In addition, version 2.3 or higher of the widget callback package Wcb (written in pure Tcl/Tk code as well) can be used to define callbacks for the `activate`, `selection set`, and `selection clear` commands. The download location of Wcb is

*http://www.nemethi.de*

## 2.2 How to get it?

Tablelist is available for free download from the same URL as Wcb. The distribution file is `tablelist2.5.tar.gz` for UNIX and `tablelist2_5.zip` for Windows. These files contain the same information, except for the additional carriage return character preceding the linefeed at the end of each line in the text files for Windows.

## 2.3 How to install it?

Install the package as a subdirectory of one of the directories given by the `auto_path` variable. For example, you can install it as a directory at the same level as the Tcl and Tk script libraries. The locations of these library directories are given by the `tcl_library` and `tk_library` variables, respectively.

To install Tablelist *on UNIX*, `cd` to the desired directory and unpack the distribution file `tablelist2.5.tar.gz`:

```
gunzip -c tablelist2.5.tar.gz | tar -xf -
```

This command will create a directory named `tablelist2.5`, with the subdirectories `demos`, `doc`, and `scripts`.

*On Windows*, use WinZip or some other program capable of unpacking the distribution file `tablelist2_5.zip` into the directory `tablelist2.5`, with the subdirectories `demos`, `doc`, and `scripts`.

Now you should check the exact version number of your Tcl/Tk distribution, given by the `tcl_patchLevel` and `tk_patchLevel` variables. If you are using TcloTk version 8.2.X, 8.3.0 - 8.3.2, or 8.4a1, then you should upgrade your Tcl/Tk distribution to a higher release. This is because a bug in these Tcl versions (fixed in Tcl 8.3.3 and 8.4a2) causes excessive memory use when calling `info exists` on non-existent array elements, and Tablelist makes a lot of invocations of this command.

If for some reason you cannot upgrade your Tcl/Tk version, then you should execute the Tcl script `repair.tcl` in the directory `scripts`. This script makes a backup copy of the file `tablelistWidget.tcl`, and then creates a new version of it by replacing all invocations of `info exists` for array elements with a call to the helper procedure `arrElemExists`. The patched file works with all Tcl/Tk releases starting with 8.0, but the original version has a better performance.

## 2.4 How to use it?

To be able to access the commands and variables defined in the package Tablelist, your scripts must contain one of the lines

```
package require Tablelist
package require tablelist
```

You can use either one of the above two statements because the file `tablelist.tcl` contains both lines

```
package provide Tablelist ...
package provide tablelist ...
```

You are free to remove one of these two lines from `tablelist.tcl` if you want to prevent the package from making itself known under two different names. Of course, by doing so you restrict the argument of `package require` to a single name. Notice that the examples below use the statement `package require Tablelist`.

Since the package Tablelist is implemented in its own namespace called `tablelist`, you must either invoke the

```
namespace import tablelist::pattern ?tablelist::pattern ...?
```

command to import the *procedures* you need, or use qualified names like `tablelist::tablelist`. In the examples below we have chosen the latter approach.

To access Tablelist *variables*, you *must* use qualified names. There are only two Tablelist variables that are designed to be accessed outside the namespace `tablelist`:

- The variable `tablelist::version` holds the current version number of the Tablelist package.
- The variable `tablelist::library` holds the location of the Tablelist installation directory.

# 3 Tablelist Examples

Also this chapter is part of the *Tablelist Programmer's Guide*.

## 3.1 A tablelist widget displaying configuration options

The file `config.tcl` in the `demos` directory contains a procedure `demo::displayConfig` that displays the configuration options of an arbitrary widget in a tablelist contained in a newly created top-level widget. This procedure can prove to be quite useful during interactive GUI development. To test it, start `wish` and evaluate the file by using the `source` command as follows:

- If `wish` was started in the `demos` directory then it is sufficient to enter

  ```
  source config.tcl
  ```

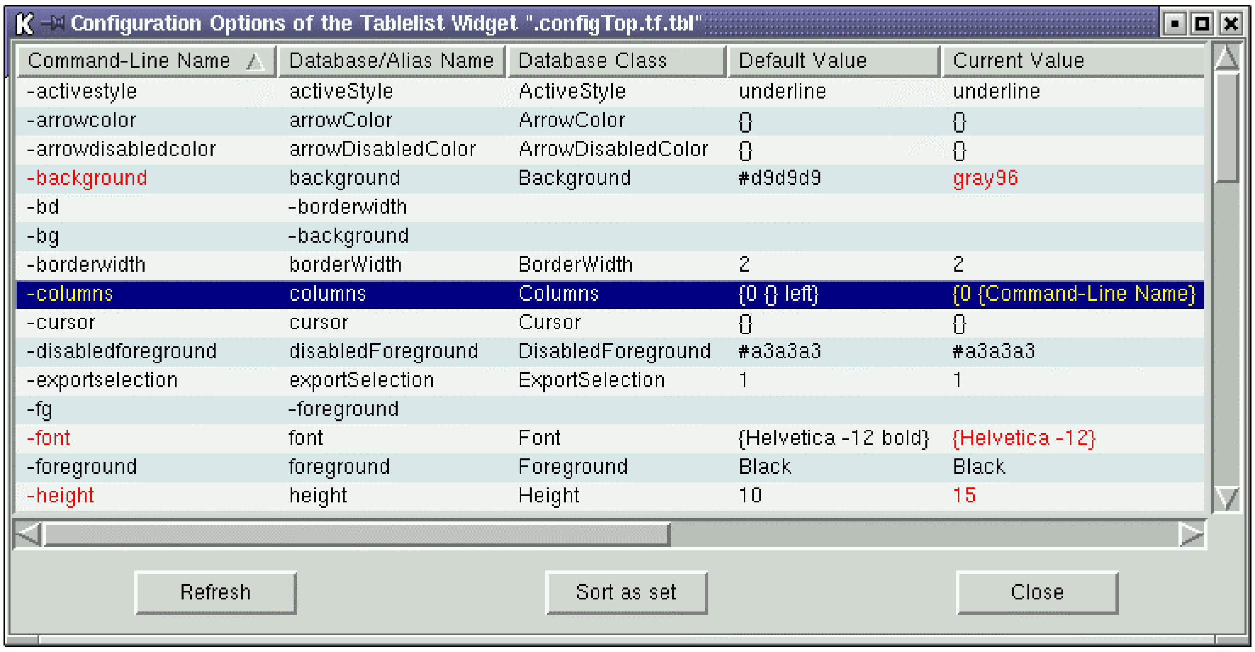
- If `wish` was started in some other directory then you can use the `tablelist::library` variable to find the location of the file:

  ```
  package require Tablelist
  source [file join $tablelist::library demos config.tcl]
  ```

In both cases, the script will print the following message to `stdout`:

```
To display the configuration options of an arbitrary widget, enter

        demo::displayConfig <widgetName>
```


Configuration Options of the Tablelist Widget ".configTop.tf.tbl"
Refresh
Sort as set
Close

| Command-Line Name | Database/Alias Name | Database Class | Default Value | Current Value |
|---|---|---|---|---|
| -activestyle | activeStyle | ActiveStyle | underline | underline |
| -arrowcolor | arrowColor | ArrowColor | {} | {} |
| -arrowdisabledcolor | arrowDisabledColor | ArrowDisabledColor | {} | {} |
| -background | background | Background | #d9d9d9 | gray96 |
| -bd | -borderwidth | | | |
| -bg | -background | | | |
| -borderwidth | borderWidth | BorderWidth | 2 | 2 |
| -columns | columns | Columns | {0 {} left} | {0 {Command-Line Name} |
| -cursor | cursor | Cursor | {} | {} |
| -disabledforeground | disabledForeground | DisabledForeground | #a3a3a3 | #a3a3a3 |
| -exportselection | exportSelection | ExportSelection | 1 | 1 |
| -fg | -foreground | | | |
| -font | font | Font | {Helvetica -12 bold} | {Helvetica -12} |
| -foreground | foreground | Foreground | Black | Black |
| -height | height | Height | 10 | 15 |

It is assumed that the Tcl command associated with the widget specified by `<widgetName>` has a `configure` subcommand which, when invoked without any argument, returns a list describing all of the available configuration options for the widget, in the common format known from the standard Tk widgets. The `demo::displayConfig` procedure inserts the items of this list into a scrolled tablelist with 5 dynamic-width columns and interactive sort capability, and returns the name of the newly created tablelist widget:

```
package require Tablelist

namespace eval demo {
    #
    # Add some entries to the Tk option database for the following
    # widget hierarchy within a toplevel widget of the class DemoTop:
    #
    # Name              Class
    # -----------------------------
    # tf                Frame
    #   tbl               Tabellist
    #   vsb, hsb          Scrollbar
    # bf                Frame
    #   b1, b2, b3        Button
    #
    switch $::tcl_platform(platform) {
        unix {
            option add *DemoTop*Font                            "Helvetica -12"
        }
        windows {
            option add *DemoTop.tf.borderWidth                  2
            option add *DemoTop.tf.relief                       sunken
            option add *DemoTop.tf.tbl.borderWidth              0
            option add *DemoTop.tf.tbl.highlightThickness       0
        }
        macintosh {
            option add *DemoTop*Background                      #dedede

            option add *DemoTop.tf.borderWidth                  2
            option add *DemoTop.tf.relief                       sunken
            option add *DemoTop.tf.tbl.borderWidth              0
            option add *DemoTop.tf.tbl.highlightThickness       0
        }
    }
    option add *DemoTop.tf.tbl.background                       gray96
    option add *DemoTop.tf.tbl.stripeBackground                 #e0e8f0
    option add *DemoTop.tf.tbl.selectBackground                 navy
    option add *DemoTop.tf.tbl.selectForeground                 white
    option add *DemoTop.tf.tbl.setGrid                          yes
    option add *DemoTop.bf.Button.width                         10
}

#------------------------------------------------------------------------------
# demo::displayConfig
#
# Displays the configuration options of the widget w in a tablelist widget
# contained in a newly created top-level widget.  Returns the name of the
# tablelist widget.
#------------------------------------------------------------------------------
proc demo::displayConfig w {
    if {![winfo exists $w]} {
        bell
        tk_messageBox -icon error -message "Bad window path name \"$w\"" \
                      -type ok
        return ""
    }

    #
    # Create a top-level widget of the class DemoTop
    #
    set top .configTop
    for {set n 2} {[winfo exists $top]} {incr n} {
        set top .configTop$n
    }
    toplevel $top -class DemoTop
    wm title $top "Configuration Options of the [winfo class $w] Widget \"$w\""

    #
    # Create a scrolled tablelist widget with 5 dynamic-width
    # columns and interactive sort capability within the top-level
    #
    set tf $top.tf
    frame $tf
    set tbl $tf.tbl
    set vsb $tf.vsb
    set hsb $tf.hsb
    tablelist::tablelist $tbl \
        -columns {0 "Command-Line Name"
                  0 "Database/Alias Name"
                  0 "Database Class"
                  0 "Default Value"
                  0 "Current Value"} \
        -labelcommand tablelist::sortByColumn -sortcommand demo::compareAsSet \
        -xscrollcommand [list $hsb set] -yscrollcommand [list $vsb set] \
        -height 15 -width 100 -stretch all
    scrollbar $vsb -orient vertical   -command [list $tbl yview]
    scrollbar $hsb -orient horizontal -command [list $tbl xview]

    #
    # Create three buttons within a frame child of the top-level widget
    #
    set bf $top.bf
    frame $bf
    set b1 $bf.b1
    set b2 $bf.b2
    set b3 $bf.b3
    button $b1 -text "Refresh"     -command [list demo::putConfig $w $tbl]
    button $b2 -text "Sort as set" -command [list $tbl sort]
    button $b3 -text "Close"       -command [list destroy $top]

    #
    # Manage the widgets
    #
    . . .

    #
    # Fill the tablelist with the configuration options of the given widget
    #
    putConfig $w $tbl
    return $tbl
}
```

The procedure invokes the `tablelist::tablelist` command to create a tablelist widget. The value of the `-columns` option passed to this command specifies the widths, titles, and alignments of the 5 columns. The width of each column is given as `0`, specifying that the column's width is to be made just large enough to hold all the elements in the column, including its title, which is the string following the width. We have omitted the alignment specifications (which can optionally follow the titles), because the columns shall all be left-justified.

The command `tablelist::sortByColumn`, specified as the value of the `-labelcommand` option, will be invoked whenever mouse button 1 is released over one of the labels. This command sorts the items based on the column corresponding to that label, in the right order, by invoking the `sortbycolumn` subcommand of the Tcl command associated with the tablelist widget.

As seen from the creation of the button displaying the text `"Sort as set"`, the items will also be sorted by invoking the `sort` subcommand. This makes it necessary to specify a command to be used for the comparison of the items, as the value of the `-sortcommand` option. In our example this is the `demo::compareAsSet` procedure shown below.

By specifying the value `all` for the `-stretch` configuration option we make sure that all columns will be stretched by the same number of pixels to eliminate the blank space that might appear at the right of the table.

Besides the options given on the command line, our tablelist widget will automatically inherit the ones contained in the Tk option database entries specified in the namespace initialization preceding the `demo::displayConfig` procedure. The only non-standard name given here is `stripeBackground`, corresponding to the `-stripebackground` configuration option. Due to this entry, every other row of the tablelist widget will be displayed in the background color `#e0e8f0`, which improves the readability of the items and gives the widget a nice appearance.

We fill the tablelist by invoking the `demo::putConfig` procedure discussed below. The same script is associated with the `Refresh` button, as the value of its `-command` configuration option. This procedure is implemented as follows:

```
#------------------------------------------------------------------------------
# demo::putConfig
#
# Outputs the configuration options of the widget w into the tablelist widget
# tbl.
#------------------------------------------------------------------------------
proc demo::putConfig {w tbl} {
    if {![winfo exists $w]} {
        bell
        tk_messageBox -icon error -message "Bad window path name \"$w\"" \
                      -parent [winfo toplevel $tbl] -type ok
        return ""
    }

    #
    # Display the configuration options of w in the tablelist widget tbl
    #
    $tbl delete 0 end
    foreach configSet [$w configure] {
        #
        # Enclose the default and current values between braces if necessary
        #
        . . .

        #
        # Insert the list configSet into the tablelist widget
        #
        $tbl insert end $configSet

        #
        # Change the colors of the first and last cell of the row
        # if the current value is different from the default one
        #
        if {[llength $configSet] != 2 &&
            [string compare $default $current] != 0} {
            foreach col {0 4} {
                $tbl cellconfigure end,$col \
                     -foreground red -selectforeground yellow
            }
        }
    }

    $tbl sortbycolumn 0
}
```

After deleting the current items of the tablelist widget `tbl`, the procedure inserts the items of the list returned by the `configure` subcommand of the Tcl command associated with the widget `w`. For each option whose current value is different from the default one, it invokes the `cellconfigure` tablelist operation to change the values of the `-foreground` and `-selectforeground` options of the cells no. 0 and 4, containing the command-line name of the option and its current value.

Finally, here is the implementation of the comparison command `demo::compareAsSet` mentioned above:

```
#------------------------------------------------------------------------------
# demo::compareAsSet
#
# Compares two items of a tablelist widget used to display the configuration
# options of an arbitrary widget.  The item in which the current value is
# different from the default one is considered to be less than the other; if
# both items fulfil this condition or its negation then string comparison is
# applied to the two option names.
#------------------------------------------------------------------------------
proc demo::compareAsSet {item1 item2} {
    foreach {opt1 dbName1 dbClass1 default1 current1} $item1 \
            {opt2 dbName2 dbClass2 default2 current2} $item2 {
        set changed1 [expr {[string compare $default1 $current1] != 0}]
        set changed2 [expr {[string compare $default2 $current2] != 0}]
        if {$changed1 == $changed2} {
            return [string compare $opt1 $opt2]
        } elseif {$changed1} {
            return -1
        } else {
            return 1
        }
    }
}
```

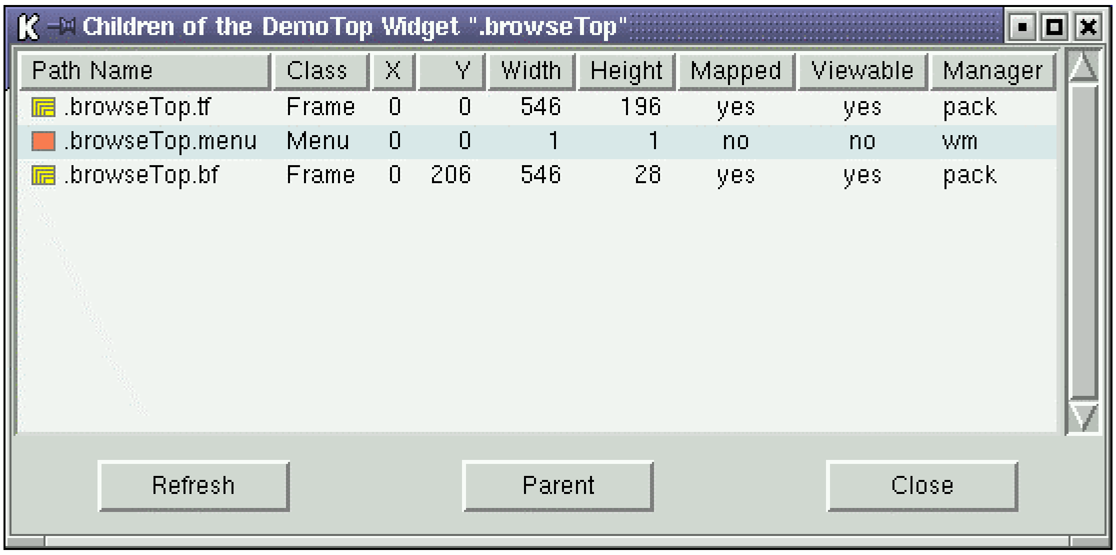
## 3.2 A simple widget browser based on a tablelist

The file `browse.tcl` in the `demos` directory contains a procedure `demo::displayChildren` that displays information about the children of an arbitrary widget in a tablelist contained in a newly created top-level widget. To test it, start `wish` and evaluate the file by using the `source` command, in a similar way as in the case of the previous example.

The script will print the following message to `stdout`:

```
To display information about the children of an arbitrary widget, enter

        demo::displayChildren <widgetName>
```



| Path Name | Class | X | Y | Width | Height | Mapped | Viewable | Manager |
|---|---|---|---|---|---|---|---|---|
| .browseTop.tf | Frame | 0 | 0 | 546 | 196 | yes | yes | pack |
| .browseTop.menu | Menu | 0 | 0 | 1 | 1 | no | no | wm |
| .browseTop.bf | Frame | 0 | 206 | 546 | 28 | yes | yes | pack |

The `demo::displayChildren` command inserts some data of the children of the widget specified by `<widgetName>` into a vertically scrolled tablelist with 9 dynamic-width columns and interactive sort capability, and returns the name of the newly created tablelist widget. By double-clicking on an item or invoking the first entry of a pop-up menu within the body of the tablelist, you can display the data of the children of the widget corresponding to the selected item, and with the second menu entry you can display its configuration options (see the previous example for details). To go one level up, click on the `Parent` button.

```
package require Tablelist

namespace eval demo {
    set dir [file join $tablelist::library demos]

    #
    # Create two images, needed in the procedure putChildren
    #
    set leafImg [image create bitmap -file [file join $dir leaf.bmp] \
                 -background coral -foreground gray50]
    set compImg [image create bitmap -file [file join $dir comp.bmp] \
                 -background yellow -foreground gray50]
}

source [file join $demo::dir config.tcl]

#------------------------------------------------------------------------------
# demo::displayChildren
#
# Displays information on the children of the widget w in a tablelist widget
# contained in a newly created top-level widget.  Returns the name of the
# tablelist widget.
#------------------------------------------------------------------------------
proc demo::displayChildren w {
    if {![winfo exists $w]} {
        bell
        tk_messageBox -icon error -message "Bad window path name \"$w\"" \
                      -type ok
        return ""
    }

    #
    # Create a top-level widget of the class DemoTop
    #
    set top .browseTop
    for {set n 2} {[winfo exists $top]} {incr n} {
        set top .browseTop$n
    }
    toplevel $top -class DemoTop

    #
    # Create a vertically scrolled tablelist widget with 9 dynamic-width
    # columns and interactive sort capability within the top-level
    #
    set tf $top.tf
    frame $tf
    set tbl $tf.tbl
    set vsb $tf.vsb
    tablelist::tablelist $tbl \
        -columns {0 "Path Name" left
                  0 "Class"     left
                  0 "X"         right
                  0 "Y"         right
                  0 "Width"     right
                  0 "Height"    right
                  0 "Mapped"    center
                  0 "Viewable"  center
                  0 "Manager"   left} \
        -labelcommand demo::labelCmd -yscrollcommand [list $vsb set] -width 0
    foreach col {2 3 4 5} {
        $tbl columnconfigure $col -sortmode integer
    }
    foreach col {6 7} {
        $tbl columnconfigure $col -formatcommand demo::formatBoolean
    }
    scrollbar $vsb -orient vertical -command [list $tbl yview]

    #
    # When displaying the information about the children of any
    # ancestor of the label widgets, the widths of some of the
    # labels and thus also the widths and x coordinates of some
    # children may change.  For this reason, make sure the items
    # will be updated after any change in the sizes of the labels
    #
    foreach l [$tbl labels] {
        bind $l <Configure> [list demo::updateItemsDelayed $tbl]
    }
    bind $tbl <Configure> [list demo::updateItemsDelayed $tbl]

    #
    # Create a pop-up menu with two command entries; bind the script
    # associated with its first entry to the <Double-1> event, too
    #
    set menu $top.menu
    menu $menu -tearoff no
    $menu add command -label "Display children" \
                      -command [list demo::putChildrenOfSelWidget $tbl]
    $menu add command -label "Display config" \
                      -command [list demo::dispConfigOfSelWidget $tbl]
    set body [$tbl bodypath]
    bind $body <<Button3>>  [bind TablelistBody <Button-1>]
    bind $body <<Button3>> +[list demo::postPopupMenu $top %X %Y]
    bind $body <Double-1>   [list demo::putChildrenOfSelWidget $tbl]

    #
    # Create three buttons within a frame child of the top-level widget
    #
    set bf $top.bf
    frame $bf
    set b1 $bf.b1
    set b2 $bf.b2
    set b3 $bf.b3
    button $b1 -text "Refresh"
    button $b2 -text "Parent"
    button $b3 -text "Close" -command [list destroy $top]

    #
    # Manage the widgets
    #
    . . .

    #
    # Fill the tablelist with the data of the given widget’s children
    #
    putChildren $w $tbl
    return $tbl
}
```

The procedure invokes the `tablelist::tablelist` command to create a tablelist widget. The value of the `-columns` option passed to this command specifies the widths, titles, and alignments of the 9 columns. The width of each column is given as `0`, specifying that the column’s width is to be made just large enough to hold all the elements in the column, including its title, which is the string following the width. Each of the titles is followed by an alignment, which indicates how to justify both the elements and the title of the respective column.

The command `demo::labelCmd`, specified as the value of the `-labelcommand` option, will be invoked whenever mouse button 1 is released over one of the labels. We will discuss this procedure a little later.

We specify the value `0` for the widget’s `-width` option, meaning that the tablelist’s width shall be made just large enough to hold all its columns.

After creating the tablelist widget, we make sure that the elements of its columns 2, 3, 4, and 5 (displaying the x and y coordinates as well as the widths and heights of the children) will be compared as integers when sorting the items based on one of these columns. We do this with the aid of the `columnconfigure` tablelist operation.

The same `columnconfigure` subcommand enables us to specify that, when displaying the elements of columns 6 and 7 (having the titles `"Mapped"` and `"Viewable"`, respectively), the boolean values `1` and `0` will be replaced with the strings `"yes"` and `"no"`, returned by the `demo::formatBoolean` command shown below.

After creating the vertical scrollbar, we iterate over the elements of the list containing the path names of all header labels of the tablelist widget, returned by the `labels` subcommand of the Tcl command corresponding to the widget. For each element of the list, we bind the procedure `demo::updateItemsDelayed` to the `<Configure>` event. In this way we make sure the procedure will be invoked whenever the header label indicated by that list element changes size.

The three invocations of the `bind` command following the creation of the pop-up menu use the path name of the tablelist’s body, returned by the `bodypath` subcommand of the Tcl command associated with the tablelist widget. Both the `<<Button3>>` virtual event (used in the first two `bind` commands) and the `TablelistBody` binding tag (used in the first binding script) are created by the Tablelist package. The first two `bind` commands make sure that a

`<<Button3>>` virtual event will select the nearest item and will post a pop-up menu with two command entries that refer to the widget described by that item.

We fill the tablelist by invoking the `demo::putChildren` procedure, implemented as follows:

```
#------------------------------------------------------------------------------
# demo::putChildren
#
# Outputs the data of the children of the widget w into the tablelist widget
# tbl.
#------------------------------------------------------------------------------
proc demo::putChildren {w tbl} {
    #
    # The following check is necessary because this procedure
    # is also invoked by the "Refresh" and "Parent" buttons
    #
    if {![winfo exists $w]} {
        . . .
    }

    set top [winfo toplevel $tbl]
    wm title $top "Children of‘the [winfo class $w] Widget \"$w\""

    #
    # Display the data of the children of the
    # widget w in the tablelist widget tbl
    #
    variable leafImg
    variable compImg
    $tbl resetsortinfo
    $tbl delete 0 end
    foreach c [winfo children $w] {
        #
        # Insert the data of the current child into the tablelist widget
        #
        set item {}
        lappend item $c [winfo class $c] [winfo x $c] [winfo y $c] \
                     [winfo width $c] [winfo height $c] [winfo ismapped $c] \
                     [winfo viewable $c] [winfo manager $c]
        $tbl insert end $item

        #
        # Insert an image into the first cell of the row
        #
        if {[llength [winfo children $c]] == 0} {
            $tbl cellconfigure end,0 -image $leafImg
        } else {
            $tbl cellconfigure end,0 -image $compImg
        }
    }

    #
    # Configure the "Refresh" and "Parent" buttons
    #
    $top.bf.b1 configure -command [list demo::putChildren $w $tbl]
    set b2 $top.bf.b2
    set p [winfo parent $w]
    if {[string compare $p ""] == 0} {
        $b2 configure -state disabled
    } else {
        $b2 configure -state normal -command [list demo::putChildren $p $tbl]
    }
}
```

After resetting the sorting information by invoking the `resetsortinfo` subcommand and deleting the current items of the tablelist widget `tbl`, the procedure iterates over the children of the specified widget and inserts the items built from some data retrieved by using the `winfo` command. For each child, it invokes the `cellconfigure` tablelist operation to set the value of the `-image` option of the first cell, containing the path name of the child. In this way, the procedure inserts the image `$leafImg` or `$compImg` into the first cell, depending upon whether the child in question is a leaf or a composite widget. Remember that both images were created outside this procedure, within the initialization of the `demo` namespace.

The `demo::formatBoolean` and `demo::labelCmd` procedures mentioned above are trivial:

```
#------------------------------------------------------------------------------
# demo::formatBoolean
#
# Returns "yes" or "no", according to the specified boolean value.
#------------------------------------------------------------------------------
proc demo::formatBoolean val {
    if {$val} {
        return yes
    } else {
        return no
    }
}

#------------------------------------------------------------------------------
# demo::labelCmd
#
# Sorts the contents of the tablelist widget tbl by its col’th column and makes
# sure the items will be updated 500 ms later (because one of the items might
# refer to the canvas containing the arrow that displays the sorting order).
#------------------------------------------------------------------------------
proc demo::labelCmd {tbl col} {
    tablelist::sortByColumn $tbl $col
    updateItemsDelayed $tbl
}
```

The command `tablelist::sortByColumn` sorts the items of the tablelist widget by the specified column in the right order, by invoking the `sortbycolumn` subcommand of the Tcl command associated with the tablelist widget.

The implementation of the `demo::updateItemsDelayed` command, invoked in this procedure and already encountered in the `demo::displayChildren` procedure above, is quite simple:

```
#------------------------------------------------------------------------------
# demo::updateItemsDelayed
#
# Arranges for the items of the tablelist widget tbl to be updated 500 ms
# later.
#------------------------------------------------------------------------------
proc demo::updateItemsDelayed tbl {
    #
    # Schedule the demo::updateItems command for execution
    # 500 ms later, but only if it is not yet pending
    #
    if {[string compare [$tbl attrib afterId] ""] == 0} {
        $tbl attrib afterId [after 500 [list demo::updateItems $tbl]]
    }
}

#------------------------------------------------------------------------------
# demo::updateItems
#
# Updates the items of the tablelist widget tbl.
#------------------------------------------------------------------------------
proc demo::updateItems tbl {
    #
    # Reset the tablelist’s "afterId" attribute
    #
    $tbl attrib afterId ""

    #
    # Update the items
    #
    set rowCount [$tbl size]
    for {set row 0} {$row < $rowCount} {incr row} {
        set c [$tbl cellcget $row,0 -text]
        if {![winfo exists $c]} {
            continue
        }

        set item {}
        lappend item $c [winfo class $c] [winfo x $c] [winfo y $c] \
                     [winfo width $c] [winfo height $c] [winfo ismapped $c] \
                     [winfo viewable $c] [winfo manager $c]
        $tbl rowconfigure $row -text $item
    }
}
```

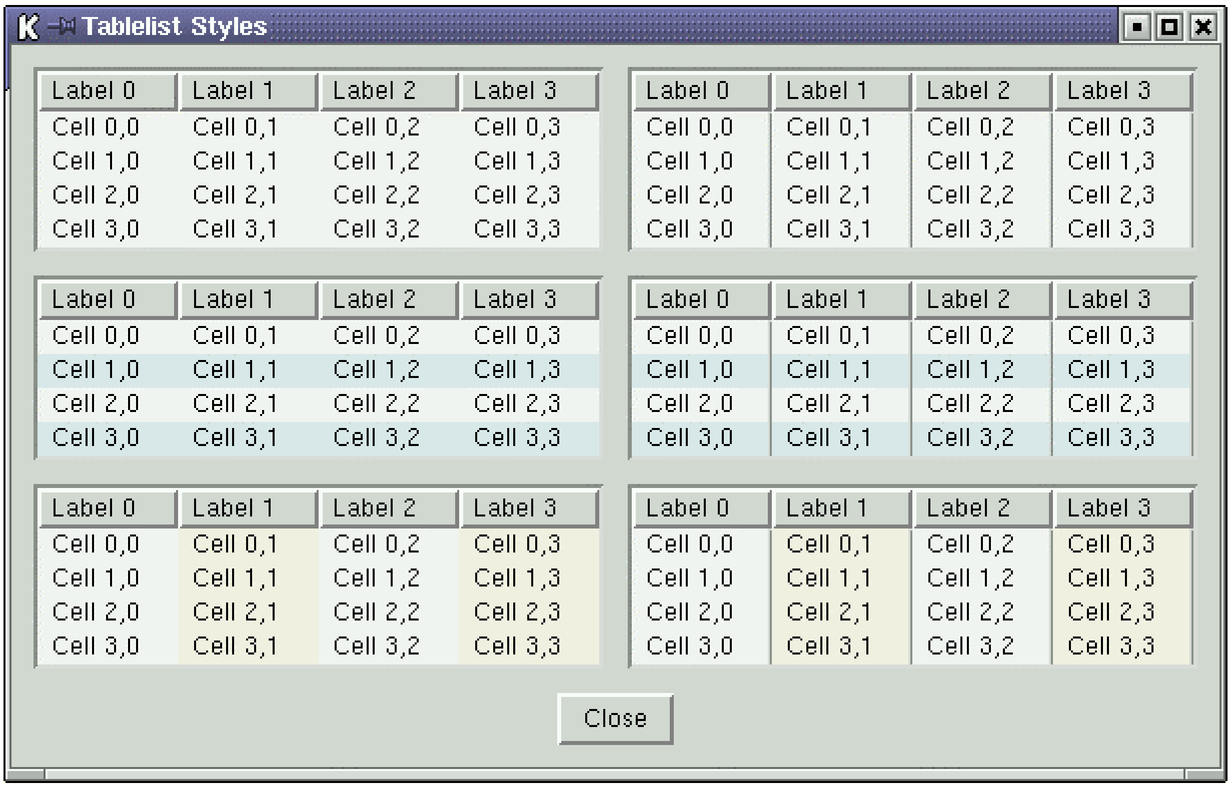
Each tablelist widget may have any number of private **attributes**, which can be set and retrieved with the aid of the `attrib` subcommand of the Tcl procedure corresponding to the widget. As shown above, the `afterId` attribute is set by the `demo::updateItemsDelayed` procedure when sheduling the `demo::updateItems` command for execution 500 ms later, but only if its value is an empty string. For this reason, the `demo::updateItems` procedure resets this attribute. It also makes use of the `cellcget` tablelist command to get the path names contained in the first cell of each row, and updates the data of the children with the aid of the `rowconfigure` command.

The remaining three procedures are also straight-forward. For example, the `demo::putChildrenOfSelWidget` command shown below makes use of the `curselection` subcommand to get the index of the selected row. More precisely, `curselection` returns a list, but in our case this list will have exactly one element, hence it can be used directly as the first component of a cell index.

```
#------------------------------------------------------------------------------
# demo::putChildrenOfSelWidget
#
# Outputs the data of the children of the selected widget into the tablelist
# widget tbl.
#------------------------------------------------------------------------------
proc demo::putChildrenOfSelWidget tbl {
    set w [$tbl cellcget [$tbl curselection],0 -text]
    if {![winfo exists $w]} {
        bell
        tk_messageBox -icon error -message "Bad window path name \"$w\"" \
                      -parent [winfo toplevel $tbl] -type ok
        return ""
    }
    if {[llength [winfo children $w]] == 0} {
        bell
    } else {
        putChildren $w $tbl
    }
}
```

## 3.3 Improving the look & feel of a tablelist widget

The script `styles.tcl` in the `demos` directory demonstrates some ways of making tablelist widgets smarter and improving the readability of their items. It creates 6 tablelist widgets, shown in the following figure:



| Label 0 | Label 1 | Label 2 | Label 3 |
|---|---|---|---|
| Cell 0,0 | Cell 0,1 | Cell 0,2 | Cell 0,3 |
| Cell 1,0 | Cell 1,1 | Cell 1,2 | Cell 1,3 |
| Cell 2,0 | Cell 2,1 | Cell 2,2 | Cell 2,3 |
| Cell 3,0 | Cell 3,1 | Cell 3,2 | Cell 3,3 |

| Label 0 | Label 1 | Label 2 | Label 3 |
|---|---|---|---|
| Cell 0,0 | Cell 0,1 | Cell 0,2 | Cell 0,3 |
| Cell 1,0 | Cell 1,1 | Cell 1,2 | Cell 1,3 |
| Cell 2,0 | Cell 2,1 | Cell 2,2 | Cell 2,3 |
| Cell 3,0 | Cell 3,1 | Cell 3,2 | Cell 3,3 |

| Label 0 | Label 1 | Label 2 | Label 3 |
|---|---|---|---|
| Cell 0,0 | Cell 0,1 | Cell 0,2 | Cell 0,3 |
| Cell 1,0 | Cell 1,1 | Cell 1,2 | Cell 1,3 |
| Cell 2,0 | Cell 2,1 | Cell 2,2 | Cell 2,3 |
| Cell 3,0 | Cell 3,1 | Cell 3,2 | Cell 3,3 |

| Label 0 | Label 1 | Label 2 | Label 3 |
|---|---|---|---|
| Cell 0,0 | Cell 0,1 | Cell 0,2 | Cell 0,3 |
| Cell 1,0 | Cell 1,1 | Cell 1,2 | Cell 1,3 |
| Cell 2,0 | Cell 2,1 | Cell 2,2 | Cell 2,3 |
| Cell 3,0 | Cell 3,1 | Cell 3,2 | Cell 3,3 |

| Label 0 | Label 1 | Label 2 | Label 3 |
|---|---|---|---|
| Cell 0,0 | Cell 0,1 | Cell 0,2 | Cell 0,3 |
| Cell 1,0 | Cell 1,1 | Cell 1,2 | Cell 1,3 |
| Cell 2,0 | Cell 2,1 | Cell 2,2 | Cell 2,3 |
| Cell 3,0 | Cell 3,1 | Cell 3,2 | Cell 3,3 |

| Label 0 | Label 1 | Label 2 | Label 3 |
|---|---|---|---|
| Cell 0,0 | Cell 0,1 | Cell 0,2 | Cell 0,3 |
| Cell 1,0 | Cell 1,1 | Cell 1,2 | Cell 1,3 |
| Cell 2,0 | Cell 2,1 | Cell 2,2 | Cell 2,3 |
| Cell 3,0 | Cell 3,1 | Cell 3,2 | Cell 3,3 |

Here is the relevant code segment:

```
#
# Create, configure, and populate 6 tablelist widgets
#
frame .f
for {set n 0} { $n < 6} {incr n} {
    set tbl .f.tbl$n
    tablelist::tablelist $tbl \
        -columns {0 "Label 0"  0 "Label 1"  0 "Label 2"  0 "Label 3"} \
        -background gray96 -selectbackground navy -selectforeground white \
        -height 4 -width 40 -stretch all

    switch $n {
        1 {                                     ;# top right tablelist
            $tbl configure -showseparators yes
        }
        2 {                                     ;# middle left tablelist
            $tbl configure -stripebackground #e0e8f0
        }
        3 {                                     ;# middle right tablelist
```

```
            $tbl configure -stripebackground #e0e8f0 -showseparators yes
        }
        4 {                                     ;# bottom left tablelist
            foreach col {1 3} {
                $tbl columnconfigure $col -background linen
            }
        }
        5 {                                     ;# bottom right tablelist
            $tbl configure -showseparators yes
            foreach col {1 3} {
                $tbl columnconfigure $col -background linen
            }
        }
    }

    foreach row {0 1 2 3} {
        $tbl insert end \
             [list "Cell $row,0" "Cell $row,1" "Cell $row,2" "Cell $row,3"]
    }
}
```

The only configuration option used here but not encountered in the first two examples is `-showseparators`. The visual effect it produces looks nice both by itself and combined with horizontal or vertical stripes, created by using the `-stripebackground` option and the `columnconfigure` command, respectively. On the other hand, it is no good idea to mix horizontal stripes with vertical ones.

# 4 Tablelist Quick Reference

## 4.1 The `tablelist::tablelist` command

**NAME**

`tablelist::tablelist` - Create and manipulate tablelist widgets

**SYNOPSIS**

**`tablelist::tablelist`** *`pathName`* `?`*`options`*`?`

**STANDARD OPTIONS**

```
-borderwidth          -highlightthickness  -setgrid
-cursor               -relief              -xscrollcommand
-exportselection      -selectbackground    -yscrollcommand
-highlightbackground  -selectborderwidth
-highlightcolor       -selectforeground
```

**OPTIONS FOR THE BODY COMPONENT OF THE WIDGET**

```
-background  -disabledforeground  -font  -foreground
```

**WIDGET-SPECIFIC OPTIONS**

```
-activestyle underline|frame
-arrowcolor color
-arrowdisabledcolor color
-columns {width title ?left|right|center? \
         ?width title ?left|right|center? ...?}
-height lines
-incrarrowtype up|down
-labelbackground color or -labelbg color
-labelborderwidth pixels or -labelbd pixels
-labelcommand command
-labeldisabledforeground color
-labelfont fontName
-labelforeground color or -labelfg color
-labelheight lines
-labelpady pixels
-labelrelief relief
-listvariable variable
-resizablecolumns boolean
-resizecursor cursor
-selectmode mode (single|browse|multiple|extended)
-showarrow boolean
-showlabels boolean
-showseparators boolean
-sortcommand command
-state normal|disabled
-stretch all|list
-stripebackground color or -stripebg color
-stripeforeground color or -stripefg color
-stripeheight lines
-takefocus 0|1|""|command
-width characters
```

**COLUMN CONFIGURATION OPTIONS**

```
-align left|right|center
-background color or -bg color
```

**-font** *font*
**-foreground** *color* or **-fg** *color*
**-formatcommand** *command*
**-hide** *boolean*
**-labelalign left|right|center**
**-labelbackground** *color* or **-labelbg** *color*
**-labelborderwidth** *pixels* or **-labelbd** *pixels*
**-labelcommand** *command*
**-labelfont** *fontName*
**-labelforeground** *color* or **-labelfg** *color*
**-labelheight** *lines*
**-labelpady** *pixels*
**-labelrelief** *relief*
**-labelimage** *image*
**-resizable** *boolean*
**-selectbackground** *color*
**-selectforeground** *color*
**-showarrow** *boolean*
**-sortcommand** *command*
**-sortmode ascii|command|dictionary|integer|real**
**-title** *title*
**-width** *width*

**ROW CONFIGURATION OPTIONS**

**-background** *color* or **-bg** *color*
**-font** *font*
**-foreground** *color* or **-fg** *color*
**-selectable** *boolean*
**-selectbackground** *color*
**-selectforeground** *color*
**-text** *value*

**CELL CONFIGURATION OPTIONS**

**-background** *color* or **-bg** *color*
**-font** *font*
**-foreground** *color* or **-fg** *color*
**-image** *image*
**-selectbackground** *color*
**-selectforeground** *color*
**-text** *value*

**ROW INDICES**

*number* **active** **anchor** **end** *@x,y*

**COLUMN INDICES**

*number* **end** *@x,y*

**CELL INDICES**

*row,col* **end** *@x,y*
    *row* : *number* **active** **anchor** **end**
    *col* : *number* **end**

**WIDGET COMMAND**

*pathName* **activate** *index*
*pathName* **attrib** ?*name*? ?*value name value* ...?
*pathName* **bbox** *index*
*pathName* **bodypath**

```
pathName cellcget cellIndex option
pathName cellconfigure cellIndex ?option? ?value option value ...?
pathName cellindex cellIndex
pathName cget option
pathName columncget columnIndex option
pathName columnconfigure columnIndex ?option? ?value option value ...?
pathName columncount
pathName columnindex columnIndex
pathName configure ?option? ?value option value ...?
pathName curselection
pathName delete first ?last?
pathName get first ?last?
pathName index index
pathName insert index ?item item ...?
pathName insertlist index itemList
pathName labelpath columnIndex
pathName labels
pathName nearest y
pathName nearestcell x y
pathName nearestcolumn x
pathName resetsortinfo
pathName rowcget index option
pathName rowconfigure index ?option? ?value option value ...?
pathName scan mark|dragto x y
pathName see index
pathName selection option args
      pathName selection anchor index
      pathName selection clear first last
      pathName selection includes index
      pathName selection set first last
pathName separatorpath columnIndex
pathName separators
pathName size
pathName sort ?-increasing|-decreasing?
pathName sortbycolumn columnIndex ?-increasing|-decreasing?
pathName sortcolumn
pathName sortorder
pathName xview args
      pathName xview
      pathName xview units
      pathName xview moveto fraction
      pathName xview scroll number what
pathName yview args
      pathName yview
      pathName yview index
      pathName yview moveto fraction
      pathName yview scroll number what
```

## 4.2 The `tablelist::sortByColumn` command

**NAME**

`tablelist::sortByColumn` - Sort the items of a tablelist widget based on one of its columns

**SYNOPSIS**

**`tablelist::sortByColumn`** *`pathName columnIndex`*