Tablelist as Multi-Column Tree Widget

by

Csaba Nemethi

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

INFOSYS GmbH, Unterhaching, Germany

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:

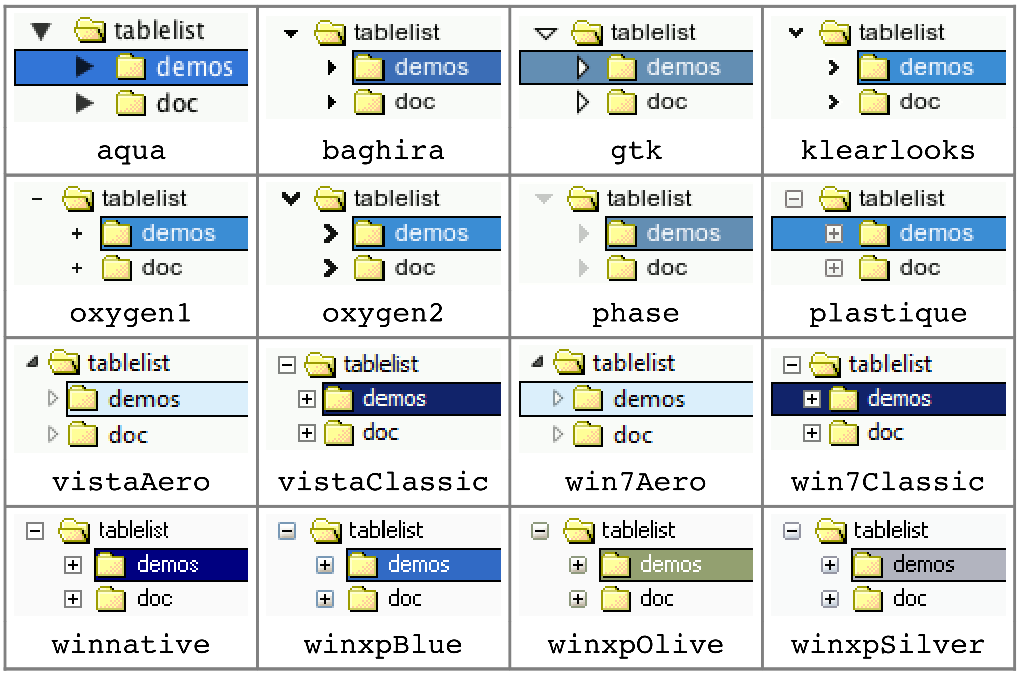
- the implementation of the tablelist mega-widget (in pure Tcl/Tk code), including a general utility module for mega-widgets;
- 9 demo scripts that create tablelist widgets in classical look;
- 10 demo-scripts that create tablelist widgets in tile look (9 of them being tile-based counterparts of the above);
- a comprehensive Tablelist Programmer's Guide in HTML format;
- detailed reference pages in HTML format.

A tablelist is a multi-column listbox and tree widget, 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.
- 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 displaying the contents of individual columns in word-wrapped multi-line rather than snipped form.
- Support for embedded images in the cells and header labels, as well as for embedded windows in the cells.
- Built-in multi-column sort capability.
- Support for moving a column or row programmatically or interactively (with the left mouse button).
- Full tile support.
- Support for interactive editing with a great variety of widgets from the Tk core and the packages tile, BWidget, Iwidgets, combobox, and Mentry.
- Support for cell- and column label-specific balloon help.
- Support for arbitrary attributes at widget, column, row, and cell levels.

2. Using a Tablelist as Multi-Column Tree Widget

When a tablelist is used as a tree widget, one of its columns (specified by the `-treecolumn` option) will display the tree hierarchy with the aid of indentations and expand/collapse controls. The look & feel of that column is controlled by the `-treestyle` option, which includes, among others, the images used for displaying the expand/collapse controls, the indentation width, and whether expand/collapse controls and indentations are to be protected when selecting a row or cell. The Tablelist package provides a great variety of tree styles, and chooses the correct default style depending on the windowing system, operating system version, and tile theme:



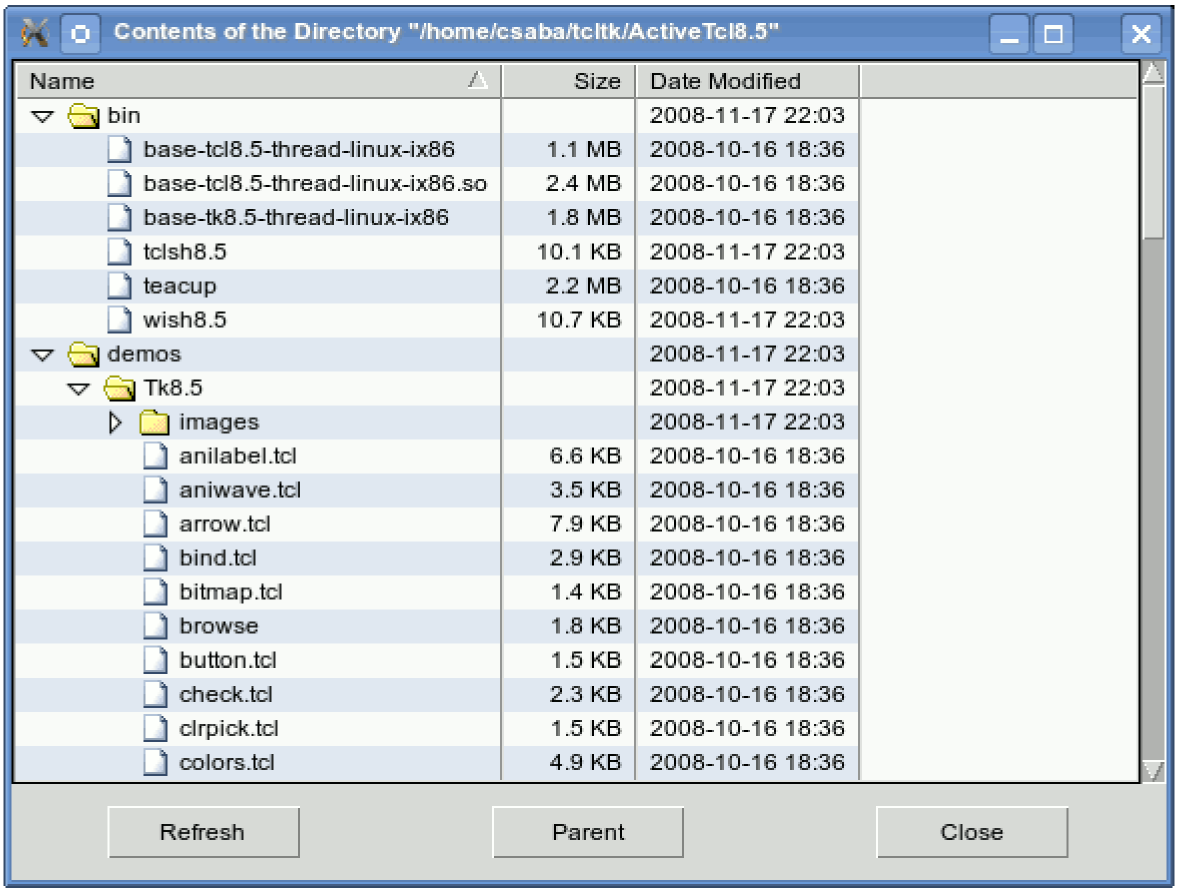
In a tablelist used as a multi-column tree widget, every row is at the same time a tree **node**, having exactly one **parent** node and any number of **child** nodes. The tree's origin is the invisible **root** node, which has no parent itself and whose children are the **top-level** nodes.

Child nodes can be inserted by invoking the `insertchild(ren)` or `insertchildlist` subcommand of the Tcl command associated with a tablelist widget. It is common practice to do this from within the command specified as the value of the `-expandcommand` option if the node being expanded (with the aid of the `expand` or `expandall` subcommand) has no children yet. There is also a `-collapsecommand` option, specifying the command to be invoked when collapsing a node (with the aid of the `collapse` or `collapseall` subcommand).

Some of the other frequently used tree-related subcommands are `childcount`, `childkeys`, and `expandedkeys`. The latter enables you to restore the expanded states of the items after redisplaying the tablelist widget's contents.

3. Example: a Directory Viewer Based on a Tablelist

The script `dirViewer.tcl` in the `demos` directory displays the contents of the volumes mounted on the system (e.g., the root `/` on UNIX and the local drives on Windows) in a tablelist used as multi-column tree widget:



Name	Size	Date Modified
bin		2008-11-17 22:03
base-tcl8.5-thread-linux-ix86	1.1 MB	2008-10-16 18:36
base-tcl8.5-thread-linux-ix86.so	2.4 MB	2008-10-16 18:36
base-tk8.5-thread-linux-ix86	1.8 MB	2008-10-16 18:36
tclsh8.5	10.1 KB	2008-11-17 22:03
teacup	2.2 MB	2008-10-16 18:36
wish8.5	10.7 KB	2008-11-17 22:03
demos		2008-11-17 22:03
Tk8.5		2008-11-17 22:03
images		2008-11-17 22:03
anilabel.tcl	6.6 KB	2008-10-16 18:36
aniwave.tcl	3.5 KB	2008-10-16 18:36
arrow.tcl	7.9 KB	2008-10-16 18:36
bind.tcl	2.9 KB	2008-10-16 18:36
bitmap.tcl	1.4 KB	2008-10-16 18:36
browse	1.8 KB	2008-10-16 18:36
button.tcl	1.5 KB	2008-10-16 18:36
check.tcl	2.3 KB	2008-10-16 18:36
clrpick.tcl	1.5 KB	2008-10-16 18:36
colors.tcl	4.9 KB	2008-10-16 18:36

By double-clicking an item or invoking the single entry of a pop-up menu within the body of the tablelist, you can display the contents of the folder corresponding to the selected item. To go one level up, click on the **Parent** button.

In the following code fragments the tablelist options and subcommands introduced in version 5.0 of the package are shown in **red** color.

```
package require Tk 8.3
package require tablelist 5.0

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

#
# Create three images
#
set clsdFolderImg [image create photo -file [file join $dir clsdFolder.gif]]
set openFolderImg [image create photo -file [file join $dir openFolder.gif]]
set fileImg       [image create photo -file [file join $dir file.gif]]
```

```
#------------------------------------------------------------------------------
# displayContents
#
# Displays the contents of the directory dir in a tablelist widget.
#------------------------------------------------------------------------------
proc displayContents dir {
    #
    # Create a vertically scrolled tablelist widget with 3
    # dynamic-width columns and interactive sort capability
    #
    set tf .tf
    frame $tf
    set tbl $tf.tbl
    set vsb $tf.vsb
    tablelist::tablelist $tbl \
        -columns {0 "Name"          left
                  0 "Size"          right
                  0 "Date Modified" left} \
        -expandcommand expandCmd -collapsecommand collapseCmd \
        -yscrollcommand [list $vsb set] -height 20 -width 80
    if {[$tbl cget -selectborderwidth] == 0} {
        $tbl configure -spacing 1
    }
    $tbl columnconfigure 0 -formatcommand formatString -sortmode dictionary
    $tbl columnconfigure 1 -formatcommand formatSize -sortmode integer
    $tbl columnconfigure 2 -formatcommand formatString
    scrollbar $vsb -orient vertical -command [list $tbl yview]

    . . .

    #
    # Create three buttons within a frame child of the main widget
    #
    set bf .bf
    frame $bf
    set b1 $bf.b1
    set b2 $bf.b2
    set b3 $bf.b3
    button $b1 -width 10 -text "Refresh"
    button $b2 -width 10 -text "Parent"
    button $b3 -width 10 -text "Close" -command exit

    . . .

    #
    # Populate the tablelist with the contents of the given directory
    #
    $tbl sortbycolumn 0
    putContents $dir $tbl root
}
```

The command to be invoked whenever an item corresponding to a nonempty folder gets expanded is specified as the value of the `-expandcommand` option. As discussed later, the `expandCmd` procedure will insert the children of the row that is about to be expanded, if it has no children yet. Similarly, the command specified by the `-collapsecommand` option will be invoked automatically when collapsing an item. As shown below, it will merely restore the image shown in the first column to the one displaying a closed folder.

```
#------------------------------------------------------------------------------
# putContents
#
# Outputs the contents of the directory dir into the tablelist widget tbl, as
# child items of the one identified by nodeIdx.
#------------------------------------------------------------------------------
proc putContents {dir tbl nodeIdx} {
    . . .

    if {[string compare $nodeIdx "root"] == 0} {
        if {[string compare $dir ""] == 0} {
            wm title . "Contents of the Workspace"
        } else {
            wm title . "Contents of the Directory \"[file nativename $dir]\""
        }

        $tbl delete 0 end
        set row 0
    } else {
        set row [expr {$nodeIdx + 1}]
    }

    #
    # Build a list from the data of the subdirectories and
    # files of the directory dir.  Prepend a "D" or "F" to
    # each entry's name and modification date & time, for
    # sorting purposes (it will be removed by formatString).
    #
    set itemList {}
    if {[string compare $dir ""] == 0} {
        foreach volume [file volumes] {
            lappend itemList [list D[file nativename $volume] -1 D $volume]
        }
    } else {
        foreach entry [glob -nocomplain -types {d f} -directory $dir *] {
            if {[catch {file mtime $entry} modTime] != 0} {
                continue
            }

            if {[file isdirectory $entry]} {
                lappend itemList [list D[file tail $entry] -1 \
                    D[clock format $modTime -format "%Y-%m-%d %H:%M"] $entry]
            } else {
                lappend itemList [list F[file tail $entry] [file size $entry] \
                    F[clock format $modTime -format "%Y-%m-%d %H:%M"] ""]
            }
        }
    }

    #
    # Sort the above list and insert it into the tablelist widget
    # tbl as list of children of the row identified by nodeIdx
    #
    set itemList [$tbl applysorting $itemList]
    $tbl insertchildlist $nodeIdx end $itemList

    #
    # Insert an image into the first cell of each newly inserted row
    #
    global clsdFolderImg fileImg
    foreach item $itemList {
        set name [lindex $item end]
        if {[string compare $name ""] == 0} {                   ;# file
            $tbl cellconfigure $row,0 -image $fileImg
        } else {                                                ;# subdirectory
            $tbl cellconfigure $row,0 -image $clsdFolderImg
            $tbl rowattrib $row pathName $name

            #
            # Mark the row as collapsed if the subdirectory is non-empty
            #
            if {[file readable $name] && [llength \
                [glob -nocomplain -types {d f} -directory $name *]] != 0} {
                $tbl collapse $row
            }
        }

        incr row
    }

    if {[string compare $nodeIdx "root"] == 0} {
        #
        # Configure the "Refresh" and "Parent" buttons
        #
        .bf.b1 configure -command [list refreshView $dir $tbl]
        set b2 .bf.b2
        if {[string compare $dir ""] == 0} {
            $b2 configure -state disabled
        } else {
            $b2 configure -state normal
            set p [file dirname $dir]
            if {[string compare $p $dir] == 0} {
                $b2 configure -command [list putContents "" $tbl root]
            } else {
                $b2 configure -command [list putContents $p $tbl root]
            }
        }
    }
}
```

The last argument of this procedure indicates the tree node to become the parent of the items displaying the contents of the directory passed as first argument. If this parent is the invisible `root` node then we first delete the current items of the tablelist widget `tbl`.

Instead of inserting the child items individually with the aid of the new `insertchild(ren)` tablelist subcommand, here we add the relevant data to a list of items and then invoke the much more performant `insertchildlist` subcommand. Also, instead of first inserting the items and then sorting them via `refreshsorting` (which is another new tablelist subcommand), we first perform the necessary sortings on the above-mentioned list of items by invoking the `applysorting` subcommand. Again, this is much faster than sorting the already inserted child items.

We mark every newly created row corresponding to a non-empty subdirectory as collapsed by invoking the `collapse` subcommand. This will prepend an expand/collapse control to the contents of the first column, whose column index `0` is the default value of the `-treecolumn` configuration option.

This procedure also illustrates an effective technique based on the `-formatcommand` column configuration option: In the tablelist widget's internal list, the names and modification times of the directories and files are preceded by a `D` and `F`, respectively. This makes sure that the directories will sort before the files (when sorting in ascending order). When displaying the items, the Tablelist code will automatically invoke the `formatString` procedure, which removes the first character. Similarly, in the widget's internal list, the size of a directory is set to `-1`, which sorts before the sizes of the files. The `formatSize` procedure, invoked automatically when displaying the items, replaces this value with an empty string:

```
#------------------------------------------------------------------------------
# formatString
#
# Returns the substring obtained from the specified value by removing its first
# character.
#------------------------------------------------------------------------------
proc formatString val {
    return [string range $val 1 end]
}

#------------------------------------------------------------------------------
# formatSize
#
# Returns an empty string if the specified value is negative and the value
# itself in user-friendly format otherwise.
#------------------------------------------------------------------------------
proc formatSize val {
    if {$val < 0} {
        return ""
    } elseif {$val < 1024} {
        return "$val bytes"
    } elseif {$val < 1048576} {
        return [format "%.1f KB" [expr {$val / 1024.0}]]
    } elseif {$val < 1073741824} {
        return [format "%.1f MB" [expr {$val / 1048576.0}]]
    } else {
        return [format "%.1f GB" [expr {$val / 1073741824.0}]]
    }
}
```

Besides its common task of inserting the children of the row to be expanded, the `expandCmd` procedure shown below also changes the image contained in the first column to the one displaying an open folder. The `collapseCmd` procedure restores the image to the one displaying a closed folder:

```
#------------------------------------------------------------------------------
# expandCmd
#
# Outputs the contents of the directory whose leaf name is displayed in the
# first cell of the specified row of the tablelist widget tbl, as child items
# of the one identified by row, and updates the image displayed in that cell.
#------------------------------------------------------------------------------
proc expandCmd {tbl row} {
    if {[$tbl childcount $row] == 0} {
        set dir [$tbl rowattrib $row pathName]
        putContents $dir $tbl $row
    }

    if {[$tbl childcount $row] != 0} {
        global openFolderImg
        $tbl cellconfigure $row,0 -image $openFolderImg
    }
}

#------------------------------------------------------------------------------
# collapseCmd
#
# Updates the image displayed in the first cell of the specified row of the
# tablelist widget tbl.
#------------------------------------------------------------------------------
proc collapseCmd {tbl row} {
    global clsdFolderImg
    $tbl cellconfigure $row,0 -image $clsdFolderImg
}
```

The procedure `refreshView`, associated with the **Refresh** button, is implemented as follows:

```
#------------------------------------------------------------------------------
# refreshView
#
# Redisplays the contents of the directory dir in the tablelist widget tbl and
# restores the expanded states of the folders as well as the vertical view.
#------------------------------------------------------------------------------
proc refreshView {dir tbl} {
    #
    # Save the vertical view and get the path names
    # of the folders displayed in the expanded rows
    #
    set yView [$tbl yview]
    foreach key [$tbl expandedkeys] {
        set pathName [$tbl rowattrib $key pathName]
        set expandedFolders($pathName) 1
    }

    #
    # Redisplay the directory's (possibly changed) contents and restore
    # the expanded states of the folders, along with the vertical view
    #
    putContents $dir $tbl root
    restoreExpandedStates $tbl root expandedFolders
    $tbl yview moveto [lindex $yView 0]
}
```

Before redisplaying the tablelist's contents via `putContents`, we get the full keys of the currently expanded items with the aid of the `expandedkeys` tablelist subcommand and insert the correspondig subdirectory paths into the array `expandedFolders`. After redisplaying the (possibly changed) contents of the directory given as first argument, we pass this array to the `restoreExpandedStates` procedure shown below:

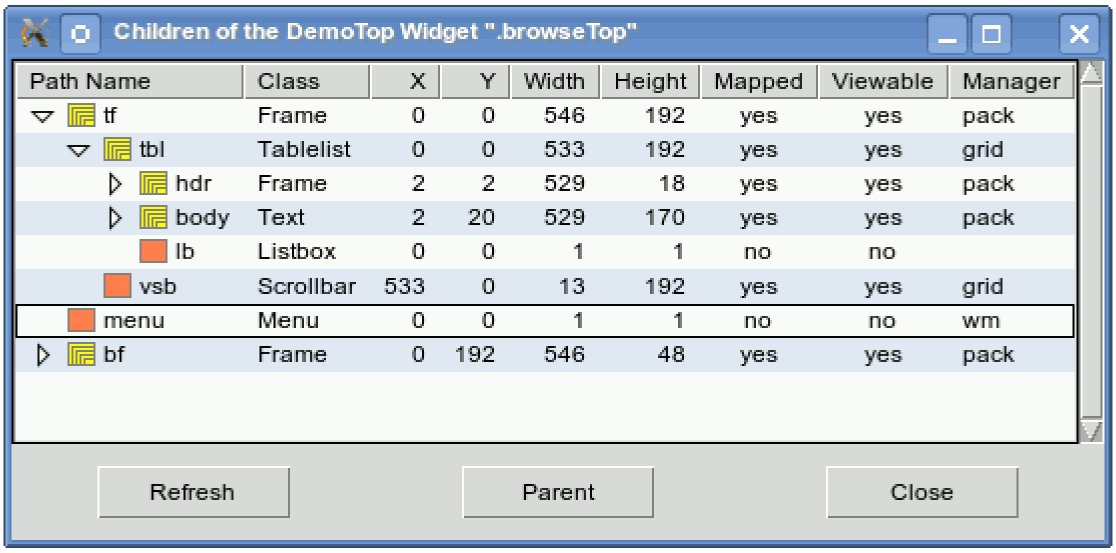
```
#------------------------------------------------------------------------------
# restoreExpandedStates
#
# Expands those children of the parent identified by nodeIdx that display
# folders whose path names are the names of the elements of the array specified
# by the last argument.
#------------------------------------------------------------------------------
proc restoreExpandedStates {tbl nodeIdx expandedFoldersName} {
    upvar $expandedFoldersName expandedFolders

    foreach key [$tbl childkeys $nodeIdx] {
        set pathName [$tbl rowattrib $key pathName]
        if {[string compare $pathName ""] != 0 &&
            [info exists expandedFolders($pathName)]} {
            $tbl expand $key -partly
            restoreExpandedStates $tbl $key expandedFolders
        }
    }
}

displayContents ""
```

The procedure retrieves the list of full keys of the children of the parent node indicated by `nodeIdx`, by means of the `childkeys` tablelist subcommand. It then loops over this list, and for each key for which the corresponding row was previously expanded, it invokes the `expand` tablelist subcommand and then calls itself recursively to restore the expanded states of that row's children.

The last line of the script invokes the procedure `displayContents` with an empty string as argument, i.e., displays the volumes mounted on the system.

4. Example: a Widget Browser Based on a Tablelist

Just like the file `browse.tcl` in the `demos` directory, the new script `browseTree.tcl` in the same distribution 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. While the tablelist created by the procedure `demo::displayChildren` in the file `browse.tcl` is a multi-column listbox, the one created by the procedure of the same name in the file `browseTree.tcl` is a multi-column tree widget:



Path Name	Class	X	Y	Width	Height	Mapped	Viewable	Manager
tf	Frame	0	0	546	192	yes	yes	pack
tbl	Tablelist	0	0	533	192	yes	yes	grid
hdr	Frame	2	2	529	18	yes	yes	pack
body	Text	2	20	529	170	yes	yes	pack
lb	Listbox	0	0	1	1	no	no	
vsb	Scrollbar	533	0	13	192	yes	yes	grid
menu	Menu	0	0	1	1	no	no	wm
bf	Frame	0	192	546	48	yes	yes	pack

The script `browseTree.tcl` is discussed in detail in Tablelist Programmer's Guide included in the Tablelist 5.0 release. It is quite similar to the one discussed in the previous section. The main difference is related to the way child items are inserted and sorted: Since the performance is not as critical as in the case of the directory viewer, this script inserts the child items individually with the aid of the `insertchild(ren)` subcommand and then sorts them (if needed) via `refreshsorting`.