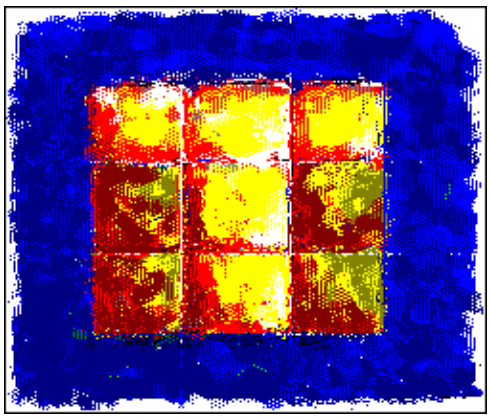

About this Manual

Table of Contents

About this Manual **1**

Legal Notices **2**

Chapter 1: Introduction to the Tix Library **4**
- TixIntro − Introduction to the Tix library 4

Chapter 2: Standard Widgets **9**
- tixGrid − Create and manipulate Tix Grid widgets 9
- tixHList − Create and manipulate Tix Hierarchial List widgets 13
- tixInputOnly − Create and manipulate TIX InputOnly widgets 27
- tixNBFrame − Create and manipulate Tix NoteBook Frame widgets 28
- tixTList − Create and manipulate Tix Tabular List widgets 30

Chapter 3: Mega Widgets **38**
- tixBalloon − Create and manipulate tixBalloon widgets 38
- tixButtonBox − Create and manipulate Tix ButtonBox widgets 40
- tixCheckList − Create and manipulate tixCheckList widgets 43
- tixComboBox − Create and manipulate tixComboBox widgets 45
- tixControl − Create and manipulate tixControl widgets 52
- tixDirList − Create and manipulate tixDirList widgets 56
- tixDirSelectDialog − Create and manipulate directory selection dialogs 60
- tixDirTree − Create and manipulate tixDirTree widgets 62
- tixExFileSelectBox − Create and manipulate tixExFileSelectBox widgets 64
- tixExFileSelectDialog − Create and manipulate tixExFileSelectDialog widgets 68
- tixFileEntry − Create and manipulate tixFileEntry widgets 70
- tixFileSelectBox − Create and manipulate Tix FileSelectBox widgets 74
- tixFileSelectDialog − Create and manipulate tixFileSelectDialog widgets 77
- tixLabelEntry − Create and manipulate tixLabelEntry widgets 79
- tixLabelFrame − Create and manipulate tixLabelFrame widgets 81
- tixListNoteBook − Create and manipulate tixListNoteBook widgets 83
- tixMeter − Create and manipulate Tix Meter widgets 86
- tixNoteBook − Create and manipulate tixNoteBook widgets 88
- tixOptionMenu − Create and manipulate tixOptionMenu widgets 92
- tixPanedWindow − Create and manipulate tixPanedWindow widgets 95
- tixPopupMenu − Create and manipulate tixPopupMenu widgets 99
- tixScrolledHList − Create and manipulate Tix ScrolledHList widgets 102
- tixScrolledListBox − Create and manipulate Tix ScrolledListBox widgets 104
- tixScrolledText − Create and manipulate Tix ScrolledText widgets 107
- tixScrolledWindow − Create and manipulate Tix ScrolledWindow widgets 109
- tixSelect − Create and manipulate tixSelect widgets 112
- tixStdButtonBox − Create and manipulate Tix StdButtonBox widgets 116
- tixTree − Create and manipulate tixTree widgets 119

Chapter 4: Display Items **123**
- tixDisplayStyle − Create style object for Tix display items 123

Chapter 5: Image Types **129**

Table of Contents

compound − multi−line compound image type.....129
pixmap − image type for the XPM file format.....133

Chapter 6: Other Commands.....135
tix − Manipulate internal states of the Tix library.....135
tixDestroy − Destroy Tix Objects.....138
tixForm − Geometry manager based on attachment rules.....138
tixGetBoolean − Get the boolean value of a string.....144
tixGetInt − Get the integer value of a string.....144
tixMwm − Communicate with the Motif(tm) window manager.....145
tixUtils − Utility commands in Tix.....146

Chapter 7: Executable Programs.....148
tixwish − Windowing shell for interpreting Tix commands.....148

Appendix 1: Tk Commands.....151
frame − Create and manipulate frame widgets.....151
image − Create and manipulate images.....153
options − Standard options supported by widgets.....155

Appendix 2: Tk Library References.....163
Tk_ConfigureWidget, Tk_Offset, Tk_ConfigureInfo, Tk_ConfigureValue, Tk_FreeOptions − process configuration options for widgets.....163
Tk_AllocBitmapFromObj, Tk_GetBitmap, Tk_GetBitmapFromObj, Tk_DefineBitmap, Tk_NameOfBitmap, Tk_SizeOfBitmap, Tk_FreeBitmapFromObj, Tk_FreeBitmap, Tk_GetBitmapFromData − maintain database of single−plane pixmaps.....172
Tk_GetPixelsFromObj, Tk_GetPixels, Tk_GetMMFromObj, Tk_GetScreenMM − translate between strings and screen units.....177

Appendix 3: Keywords.....180
A.....180
B.....180
C.....180
D.....181
F.....181
G.....181
H.....182
I.....182
J.....182
L.....182
M.....182
N.....183
O.....183
P.....183
R.....183
S.....183
T.....184
U.....184

Table of Contents

W.....184
X.....184

About this Manual

This manual is originally written in nroff format. It is converted to HTML format using the `tcltk-man2html.tcl` tool written by Roger E. Critchlow Jr. The PDF version of this manual is generated from HTML files using the HTMLDOC tool available from http://www.easysw.com/htmldoc .

The nroff sources to the Tix portion of this manual may be downloaded from http://tixlibrary.sourceforge.net/ as part of the Tix source distribution. The sources to the Tcl/Tk portion of this manual may be downloaded from http://dev.scriptics.com/ as part of the Tcl/Tk source distribution.

Tix and Tcl/Tk are open-source software. Please see the the Legal Notices section for licensing terms on the documentation included in this manual.

Legal Notices

(1) The Tix portion of this Reference Manual is subject to the following licensing terms.

(2) The Tcl/Tk portion of this Reference Manual is subject to the following licensing terms.

described here, provided that the new terms are clearly indicated on the first page of each file where they apply.

IN NO EVENT SHALL THE AUTHORS OR DISTRIBUTORS BE LIABLE TO ANY PARTY FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OF THIS SOFTWARE, ITS DOCUMENTATION, OR ANY DERIVATIVES THEREOF, EVEN IF THE AUTHORS HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGE.

THE AUTHORS AND DISTRIBUTORS SPECIFICALLY DISCLAIM ANY WARRANTIES, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE, AND NON−INFRINGEMENT. THIS SOFTWARE IS PROVIDED ON AN "AS IS" BASIS, AND THE AUTHORS AND DISTRIBUTORS HAVE NO OBLIGATION TO PROVIDE MAINTENANCE, SUPPORT, UPDATES, ENHANCEMENTS, OR MODIFICATIONS.

GOVERNMENT USE: If you are acquiring this software on behalf of the U.S. government, the Government shall have only "Restricted Rights" in the software and related documentation as defined in the Federal Acquisition Regulations (FARs) in Clause 52.227.19 (c) (2). If you are acquiring the software on behalf of the Department of Defense, the software shall be classified as "Commercial Computer Software" and the Government shall have only "Restricted Rights" as defined in Clause 252.227−7013 (c) (1) of DFARs. Notwithstanding the foregoing, the authors grant the U.S. Government and others acting in its behalf permission to use and distribute the software in accordance with the terms specified in this license.

Chapter 1: Introduction to the Tix Library

TixIntro − Introduction to the Tix library

DESCRIPTION

Tix, which stands for Tk Interface Extension, is an extension library for Tcl/Tk. Tix adds many new widgets, image types and other commands that allows you to create compelling Tcl/Tk−based GUI applications.

One advantage of Tix over other Tk widget libraries is many of the Tix standard widgets are implemented in native code. This enhances performance and provides native look−and−feel for your applications.

This version of Tix works with Tcl/Tk version 8.0 or above. If Tix has been installed properly on your system, you can load the package into any Tk script by invoking the command

```
package require Tix
```

After this command has successfully returned, you can start using the functionalities of the Tix library. See the **EXAMPLES** section below for example scripts that use Tix.

If the "package require Tix" command fails, you probably need to install a new copy of Tix on your system. You can download the latest version of Tix from the web site **http://tixlibrary.sourceforge.net/**.

STANDARD WIDGETS

Tix includes the following standard widgets which, like their counterparts in Tk, are implemented in native code to achieve high performance and native look−and−feel.

tixGrid
: The **tixGrid** widget displays items in a spread−sheet format.

tixHList
: Hierarchical listbox widget. This widget display entries in a tree−like format.

tixInputOnly
: A transparent window that can be used to cover another widget so as to disable mouse input.

tixNBFrame
: The **tixNBFrame** widget is used internally by the **tixNoteBook** widget to display choices among a set of overlapping pages.

tixTList
Tabular listbox widget. This widget is similar to the built−in Tk **listbox** widget but provides more flexibility in displaying the list entries.

MEGA WIDGETS

Tix provides many new types of *mega widgets*: these are widgets that are composed of built−in Tk widgets and the Tix standard widgets mentioned above.

tixBalloon
The **tixBalloon** widget provides context−sensitive on−screen help.

tixButtonBox
A convenience class for creating a box of **button** widgets.

tixCheckList
A subclass of **tixTree** that presents single− or multiple choices to the user in a tree−like format.

tixComboBox
A combination of the **listbox** and **entry** widgets that allows the user to input an item by typing or by selecting from a list of items.

tixControl
The **tixControl** widget allows the user to adjust a value using arrow buttons.

tixDirList
A directory selection widget that displays the file system as a cascading list.

tixDirSelectDialog
A dialog for selecting a directory. *This widget is deprecated.* Use **tk_chooseDirectory** instead.

tixDirTree
A directory selection widget that displays the file system in a tree format.

tixExFileSelectBox
A widget for selecting a file. *This widget is deprecated.* Use **tk_getOpenFile** instead.

tixExFileSelectDialog
A dialog for selecting a file. *This widget is deprecated.* Use **tk_getOpenFile** instead.

tixFileEntry
A special entry widget that allows the user to choose a file by typing in its name or by selecting from a file dialog.

tixFileSelectBox
A widget for selecting a file. *This widget is deprecated.* Use **tk_getOpenFile** instead.

tixFileSelectDialog
A dialog for selecting a file. *This widget is deprecated.* Use **tk_getOpenFile** instead.

tixLabelEntry
A convenience class for creating an **entry** with an associated **label** widget.

tixLabelFrame
A labeled **frame** widget for grouping together a set of related widgets.

tixListNoteBook
The **tixListNoteBook** widget allows the user to choose from a set of over−lapping pages by selecting from a list box.

tixMeter
The **tixMeter** widget displays a bar graph to indicate progress.

tixNoteBook
The **tixNoteBook** widget allows the user to choose from a set of over−lapping pages with a tabbed notebook metaphor.

tixOptionMenu
The **tixOptionMenu** widget allows the user to choose a value from a pop−up menu.

tixPanedWindow
The **tixPanedWindow** widgets allows the user to adjust the visible size of several **frame** widgets with handle bars.

tixPopupMenu
The **tixPopupMenu** widget can be used to implement context−sensitive pop−up menu.

tixScrolledHList
A **tixHList** widget with smart scrollbars. Like other Tix scrolled widgets, the scroll bars can be displayed on an as−needed basis.

tixScrolledListBox
A Tk **listbox** widget with smart scrollbars.

tixScrolledText
A Tk **text** widget with smart scrollbars.

tixScrolledWindow
A Tk **frame** widget with smart scrollbars.

tixSelect
The **tixSelect** widget uses a set of buttons to present single− or multiple selection options to the user.

tixStdButtonBox
A subclass of **tixButtonBox** that provides four standard buttons (OK, Apply, Cancel Help) for Motif(TM)−like dialog boxes.

tixTree
A subclass of **tixScrolledHList** that supports expanding and collapsing of entries in a hierarchical list.

DISPLAY ITEMS

Three Tix standard widgets **tixGrid**, **tixHList** and **tixTList** support the *Display Items* mechanism. Display items are created by the widget command of these standard widgets. You can customize the items using *styles* objects.

tixDisplayStyle
: Create style objects to customize display items.

IMAGE TYPES

Tix provides two additional image types to the Tk **image** framework.

compound
: The **compound** image type allows you to combine images, widgets, and text strings into a single Tk **image** object.

pixmap
: The **pixmap** image type supports the XPM format.

OTHER COMMANDS

The following Tcl command are also included in the Tix library to perform various functions.

tixDestroy
: Destroy Tix objects instances of classes defined by **tixClass** construct.

tixForm
: A geometry manager based on attachment rules. *This geometry manager is deprecated*. Use the Tk **grid** geometry manager instead.

tixMwm
: A command to communicate with the Mwm, the Motif(TM) Window Manager. Works on Unix only.

tix
: The **tix** command controls the Tix application context.

tixGetBoolean
: The **tixGetBoolean** command converts a string to a boolean value.

tixGetInt
: The **tixGetInt** command converts a string to a integer value.

tixUtils
: The **tixUtils** manual page describes some utility Tix commands.

EXECUTABLE PROGRAM

tixwish
: The **tixwish** program can be used to execute Tix−based applications. **tixwish** *is*

deprecated. You should use the standard **wish** program from Tk and access Tix via the "package require Tix" command.

EXAMPLES

The following is an example script that uses a **tixTree** widget.

```
set tree [tixTree .t]
pack $tree -expand yes -fill both
for {set i 0} {$i < 2} {incr i} {
    $tree subwidget hlist add $i -itemtype imagetext \
        -text "Folder $i" -image [tix getimage folder]
    for {set j 0} {$j < 5} {incr j} {
        $tree subwidget hlist add $i.$j -itemtype imagetext \
            -text "File $i.$j" -image [tix getimage textfile]
    }
}
$tree autosetmode
```

KEYWORDS

Tix, mega widgets

Chapter 2: Standard Widgets

tixGrid − Create and manipulate Tix Grid widgets

SYNOPSIS

tixGrid *pathName* ?*options*?

STANDARD OPTIONS

−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−font, font, Font
−foreground or −fg, foreground, Foreground
−height
−highlightColor
−highlightThickness
−relief, relief, Relief
−selectBackground
−selectForeground
−width
−xScrollCommand
−yScrollCommand

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−editdonecmd***
Database Name: ***editDoneCmd***
Database Class: ***EditDoneCmd***

If non−empty, gives a Tcl command to be executed when the user has edited grid cell. When this command is called, it is passed with two additional parameters: *x y*, where (*x*,*y*) is the location of the cell that has just been edited.

Command−Line Name: ***−editnotifycmd***
Database Name: ***editNotifyCmd***
Database Class: ***EditNotifyCmd***

If non−empty, gives a Tcl command to be executed when the user tries to edit a grid cell. When this command is called, it is passed with two additional parameters: *x y*, where (*x*,*y*,) is the location of the cell. This command should return a boolean value: **true** indicates that the cells is editable and **false** otherwise.

Command−Line Name: ***−formatcmd***
Database Name: ***formatCmd***

Database Class: ***FormatCmd***

If non−empty, gives a Tcl command to be executed when the grid cells need to be formatted on the screen. Normally, this command calls the **format** widget command (see below). When this command is called, it is passed with five additional parameters: *type x1 y1 x2 y2*. *type* gives the logical type of the region in the grid. It may be one of the following. **x−region**: the horizontal margin; **y−region**: the vertical margin; **s−region**, the area where the the horizontal and vertical margins are joined; **main**: all the cells that do not fall into the above three types. *x1 y1 x2 y2* gives the extent of the region that needs formatting.

Command−Line Name: ***−leftmargin***
Database Name: ***leftMargin***
Database Class: ***LeftMargin***

In the number of cells, gives the width of vertical margin. A zero indicates that no vertical should be drawn.

Command−Line Name: ***−selectmode***
Database Name: ***selectMode***
Database Class: ***SelectMode***

Specifies one of several styles for manipulating the selection. The value of the option may be arbitrary, but the default bindings expect it to be either **single**, **browse**, **multiple**, or **extended**; the default value is **single**.

Command−Line Name: ***−selectunit***
Database Name: ***selectUnit***
Database Class: ***SelectUnit***

Specifies the selection unit. Valid values are **cell**, **column** or **row**.

Command−Line Name: ***−topmargin***
Database Name: ***topMargin***
Database Class: ***TopMargin***

In the number of cells, gives the height of horizontal margin. A zero indicates that no horizontal should be drawn.

DESCRIPTION

The **tixGrid** command creates a new window (given by the *pathName* argument) and makes it into a **tixGrid** widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the **tixGrid** widget such as its cursor and relief.

A Grid widget displays its contents in a two dimensional grid of cells. Each cell may contain one Tix **display item**, which may be in text, graphics or other formats. See the **tixDisplayStyle** manual page for more information about Tix display items. Individual cells, or groups of cells, can be formatted with a wide range of attributes, such as its color, relief and border.

WIDGET COMMAND

The **tixGrid** command creates a new Tcl command whose name is the same as the path name of the **tixGrid** widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the **tixGrid** widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for **tixGrid** widgets:

pathName ***anchor*** *option ?args ...?*

Manipulates the **anchor cell** of the **tixGrid** widget. The anchor cell is the end of the selection that is fixed while the user is dragging out a selection with the mouse.

pathName ***bdtype***

TODO place holder

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixGrid** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list.) If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixGrid** command.

pathName ***delete*** *dim from ?to?*

Dim may be **row** or **column**. If *to* is not given, deletes a single row (or column) at the position *from*. If *to* is given, deletes the range of rows (or columns) from position *from* through *to*.

pathName ***edit apply***

If any cell is being edited, de−highlight the cell and applies the changes.

pathName ***edit set*** *x y*

Highlights the cell at (*x*,*y*) for editing, if the **−editnotify** command returns true for this cell.

pathName ***entrycget*** *x y option*

Returns the current value of the configuration option given by *option* of the cell at (*x*,*y*). *Option* may have any of the values accepted by the **set** widget command.

pathName ***entryconfigure*** *x y ?option? ?value option value ...?*

Query or modify the configuration options of the cell at (*x*,*y*). If no *option* is specified, returns a list describing all of the available options for the cell (see **Tk_ConfigureInfo** for information on the format of this list.) If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified.) If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by

the **set** widget command.

pathName ***format***
TODO place holder

pathName ***index***
TODO place holder

pathName ***move*** *dim from to offset*
Dim may be **row** or **column**. Moves the the range of rows (or columns) from position *from* through *to* by the distance indicated by *offset*. For example, **move row 2 4 1** moves the rows 2,3,4 to rows 3,4,5.

pathName ***set*** *x y ?−itemtype type? ?option value...?*
Creates a new display item at the cell at (*x*,*y*). The optional **−itemtype** parameter gives the type of the display item. An additional list of *option−value* pairs specify options of the display item. If a display item already exists at this cell, the old item will be deleted automatically.

pathName ***size*** *dim index ?option value ...?*
Queries or sets the size of the row or column given by *dim* and *index*. *Dim* may be **row** or **column**. *Index* may be any non−negative integer that gives the position of a given row (or column). *Index* can also be the string **default**; in this case, this command queries or sets the default size of all rows (or columns).

When no *option−value* pair is given, this command returns a list containing the current size setting of the given row (or column). When *option−value* pairs are given, the corresponding options of the size setting of the given row are changed.
Option may be one of the follwing:

−pad0 *pixels*
Specifies the paddings to the left or a column or the top of a row.

−pad1 *pixels*
Specifies the paddings to the right or a column or the bottom of a row.

−size *val*
Specifies the width of a column or the height of a row. *Val* may be: **auto** −− the width of the column is set the the widest cell in the column; a valid Tk screen distance unit (see **Tk_GetPixels**); or a real number following by the word **chars** (e.g. **3.4chars**) that sets the width of the column to the given number of characters.

pathName ***unset*** *x y*
Clears the cell at (*x*,*y*) by removing its display item.

pathName ***xview***
TODO place holder

pathName ***yview***
TODO place holder

KEYWORDS

grid, spread sheet, table

tixHList − Create and manipulate Tix Hierarchial List widgets

SYNOPSIS

tixHList *pathName* ?*options*?

STANDARD OPTIONS

−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−font, font, Font
−foreground or −fg, foreground, Foreground
−height
−highlightColor −highlightThickness
−relief, relief, Relief
−selectBackground
−selectForeground
−width
−xScrollCommand
−yScrollCommand

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browsecmd***
Database Class: ***BrowseCmd***

Specifies a TCL command to be executed when the user browses through the entries in the HList widget.

Command−Line Name: ***−columns***
Database Name: ***columns***
Database Class: ***Columns***

Specifies the number of columns in this HList widget. This option can only be set during the creation of the HList widget and cannot be changed subsequently.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the TCL command to be executed when the user invokes a list entry in the HList widget. Normally the user invokes a list entry by double−clicking it or pressing the Return key.

Command−Line Name: ***−drawbranch***
Database Name: ***drawBranch***
Database Class: ***DrawBranch***
A Boolean value to specify whether branch line should be drawn to connect list entries to their parents.

Command−Line Name: ***−foreground or −fg***
Database Name: ***foreground***
Database Class: ***Foreground***
[OBSOLETE] Specifies the default foreground color for the list entries.

Command−Line Name: ***−gap***
Database Name: ***gap***
Database Class: ***Gap***
[OBSOLETE] The default distance between the bitmap/image and the text in list entries.

Command−Line Name: ***−header***
Database Name: ***header***
Database Class: ***Header***
A Boolean value specifying whether headers should be displayed for this HList widget (see the **header** widget command below).

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***
Specifies the desired height for the window in number of characters.

Command−Line Name: ***−indent***
Database Name: ***indent***
Database Class: ***Indent***
Specifies the amount of horizontal indentation between a list entry and its children. Must be a valid screen distance value.

Command−Line Name: ***−indicator***
Database Name: ***indicator***
Database Class: ***Indicator***
Specifies whether the indicators should be displayed inside the HList widget. See the **indicator** widget command below.

Command−Line Name: ***−indicatorcmd***
Database Name: ***indicatorCmd***
Database Class: ***IndicatorCmd***
Specifies a TCL command to be executed when the user manipulates the indicator of an HList entry. The **−indicatorcmd** is triggered when the user press or releases the mouse button over the indicator in an HList entry. By default the TCL command specified by **−indicatorcmd** is executed with one additional argument, the entryPath of the entry whose indicator has been triggered. Additional information about the event can be obtained by the **tixEvent** command.

Command−Line Name: ***−itemtype***
Database Name: ***itemType***
Database Class: ***ItemType***
Specifies the default type of display item for this HList widget. When you call the add and addchild widget commands, display items of this type will be created if the **−itemtype** option is not specified .

Command−Line Name: ***−padx***
Database Name: ***padX***
Database Class: ***Pad***
[OBSOLETE] The default horizontal padding for list entries.

Command−Line Name: ***−padx***
Database Name: ***padY***
Database Class: ***Pad***
[OBSOLETE] The default vertical padding for list entries.

Command−Line Name: ***−selectbackground***
Database Name: ***selectBackground***
Database Class: ***SelectBackground***
Specifies the background color for the selected list entries.

Command−Line Name: ***−selectborderwidth***
Database Name: ***selectBorderWidth***
Database Class: ***BorderWidth***
Specifies a non−negative value indicating the width of the 3−D border to draw around selected items. The value may have any of the forms acceptable to **Tk_GetPixels**.

Command−Line Name: ***−selectforeground***
Database Name: ***selectForeground***
Database Class: ***SelectForeground***
Specifies the foreground color for the selected list entries.

Command−Line Name: ***−selectmode***
Database Name: ***selectMode***
Database Class: ***SelectMode***
Specifies one of several styles for manipulating the selection. The value of the option may be arbitrary, but the default bindings expect it to be either **single**, **browse**, **multiple**, or **extended**; the default value is **single**.

Command−Line Name: ***−sizecmd***
Database Name: ***sizeCmd***
Database Class: ***SizeCmd***
Specifies a TCL script to be called whenever the HList widget changes its size. This command can be useful to implement "user scroll bars when needed" features.

Command−Line Name: ***−separator***
Database Name: ***separator***
Database Class: ***Separator***
Specifies the character to used as the separator character when intepreting the path−names of list entries. By default the character "." is used.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window in characters.

DESCRIPTION

The **tixHList** command creates a new window (given by the *pathName* argument) and makes it into a HList widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the HList widget such as its cursor and relief.

The HList widget can be used to display any data that have a hierarchical structure, for example, file system directory trees. The list entries are indented and connected by branch lines according to their places in the hierachy.

Each list entry is identified by an **entryPath**. The entryPath is a sequence of **entry names** separated by the separator charactor (specified by the **−separator** option). An **entry name** can be any string that does not contain the separator charactor, or it can be the a string that contains only one separator charactor.

For example, when "." is used as the separator charactor, "one.two.three" is the entryPath for a list entry whose parent is "one.two", whose parent is "one", which is a toplevel entry (has no parents).

Another examples: ".two.three" is the entryPath for a list entry whose parent is ".two", whose parent is ".", which is a toplevel entry.

DISPLAY ITEMS

Each list entry in an HList widget is associated with a **display item**. The display item determines what visual information should be displayed for this list entry. Please see the **tixDisplayStyle** manual page for a list of all display items. When a list entry is created by the **add** or **addchild** widget commands, the type of its display item is determined by the **−itemtype** option passed to these commands. If the **−itemtype** is omitted, then by default the type specified by this HList widget's **−itemtype** option is used.

WIDGET COMMAND

The **tixHList** command creates a new Tcl command whose name is the same as the path name of the HList widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the HList widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for HList widgets:

pathName ***add*** *entryPath ?option value ...?*
Creates a new list entry with the pathname *entryPath*. A list entry must be created after its parent is created (unless this entry is a top−level entry, which has no parent). This command returns the entryPath of the newly created list entry. The following

configuration options can be given to configure the list entry:

−at *position*
Insert the new list at the position given by *position*. *position* must be a valid integer. the Position **0** indicates the first position, **1** indicates the second position, and so on.

−after *afterWhich*
Insert the new list entry after the entry identified by *afterWhich*. *afterWhich* must be a valid list entry and it mush have the same parent as the new list entry

−before *beforeWhich*
Insert the new list entry before the entry identified by *beforeWhich*. *beforeWhich* must be a valid list entry and it mush have the same parent as the new list entry

−data *string*
Specifies a string to associate with this list entry. This string can be queried by the **info** widget command. The application programmer can use the **−data** option to associate the list entry with the data it represents.

−itemtype *type*
Specifies the type of display item to be display for the new list entry. **type** must be a valid display item type. Currently the available display item types are **imagetext**, **text**, and **window**. If this option is not specified, then by default the type specified by this HList widget's **−itemtype** option is used.

−state
Specifies whether this entry can be selected or invoked by the user. Must be either **normal** or **disabled**.

The **add** widget command accepts additional configuration options to configure the display item associated with this list entry. The set of additional configuration options depends on the type of the display item given by the **−itemtype** option. Please see the **tixDisplayStyle** manual page for a list of the configuration options for each of the display item types.

pathName ***addchild*** *parentPath ?option value ... ?*
Adds a new child entry to the children list of the list entry identified by *parentPath*. Or, if *parentPath* is set to be the empty string, then creates a new toplevel entry. The name of the new list entry will be a unique name automatically generated by the HList widget. Usually if *parentPath* is **foo**, then the entryPath of the new entry will be **foo.1**, **foo.2**, ... etc. This command returns the entryPath of the newly created list entry. *option* can be any option for the **add** widget command.

pathName ***anchor set*** *entryPath*
Sets the anchor to the list entry identified by *entryPath*. The anchor is the end of the selection that is fixed while the user is dragging out a selection with the mouse.

pathName ***anchor clear***

Removes the anchor, if any, from this HList widget. This only removes the surrounding highlights of the anchor entry and does not affect its selection status.

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixHList** command.

pathName ***column width*** *col ?−char? ?width?*
Querys or sets the width of a the column *col* in the HList widget. The value of *col* is zero−based: 0 stands for the first column, 1 stands for the second, and so on. If no further parameters are given, returns the current width of this column (in number of pixels). Additional parameters can be given to set the width of this column:

pathName ***column width*** *col* ***{}***
An empty string indicates that the width of the column should be just wide enough to display the widest element in this column. In this case, the width of this column may change as a result of the elements in this column changing their sizes.

pathName ***column width*** *col width*
width must be in a form accepted by **Tk_GetPixels**.

pathName ***column width*** *col* ***−char*** *nChars*
The width is set to be the average width occupied by *nChars* number of characters of the font specified by the **−font** option of this HList widget.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixHList** command.

pathName ***delete*** *option ?entryPath?*
Delete one or more list entries. *option* may be one of the following:

all
Delete all entries in the HList. In this case the *entryPath* does not need to be specified.

entry
Delete the entry specified by *entryPath* and all its offsprings, if any.

offsprings
Delete all the offsprings, if any, of the entry specified by *entryPath*. However, *entryPath* itself is not deleted.

siblings
Delete all the list entries that share the same parent with the entry specified by *entryPath*. However, *entryPath* itself is not deleted.

pathName ***dragsite set*** *entryPath*

Sets the dragsite to the list entry identified by *entryPath*. The dragsite is used to indicate the source of a drag−and−drop action. Currently drag−and−drop functionality has not been implemented in Tix yet.

pathName ***dragsite clear***

Remove the dragsite, if any, from the this HList widget. This only removes the surrounding highlights of the dragsite entry and does not affect its selection status.

pathName ***dropsite set*** *entryPath*

Sets the dropsite to the list entry identified by *entryPath*. The dropsite is used to indicate the target of a grag−and−drop action. Currently drag−and−drop functionality has not been implemented in Tix yet.

pathName ***dropsite clear***

Remove the dropsite, if any, from the this HList widget. This only removes the surrounding highlights of the dropsite entry and does not affect its selection status.

pathName ***entrycget*** *entryPath option*

Returns the current value of the configuration option given by *option* for the entry indentfied by *entryPath*. *Option* may have any of the values accepted by the **add** widget command.

pathName ***entryconfigure*** *entryPath ?option? ?value option value ...?*

Query or modify the configuration options of the list entry indentfied by *entryPath*. If no *option* is specified, returns a list describing all of the available options for *entryPath* (see **Tk_ConfigureInfo** for information on the format of this list.) If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **add** or **addchild** widget command. The exact set of options depends on the value of the **−itemtype** option passed to the the **add** or **addchild** widget command when this list entry is created.

pathName ***header*** *option col ?args ...?*

Manipulates the header items of this HList widget. If the **−header** option of this HList widget is set to true, then a header item is displayed at the top of each column. The *col* argument for this command must be a valid integer. 0 indicates the first column, 1 the second column, ... and so on. This command supports the following options:

pathName ***header cget*** *col option*

If the *col*−th column has a header display item, returns the value of the specified *option* of the header item. If the header doesn't exist, returns an error.

pathName ***header configure*** *col ?option? ?value option value ...?*

Query or modify the configuration options of the header display item of the *col*−th column. The header item must exist, or an error will result. If no *option* is specified, returns a list describing all of the available options for the header display item (see **Tk_ConfigureInfo** for information on the

format of this list.) If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **header create** widget command. The exact set of options depends on the value of the **−itemtype** option passed to the the **header create** widget command when this display item was created.

pathName ***header create*** *col ?−itemtype type? ?option value ...?*
Creates a new display item as the header for the *col*−th column. If an header display item already exists for this column, it will be replaced by the new item. An optional parameter *−itemtype* can be used to specify what type of display item should be created. If the *−itemtype* is not given, then by default the type specified by this HList widget's **−itemtype** option is used. Additional parameters, in *option−value* pairs, can be passed to configure the appearance of the display item. Each *option−value* pair must be a valid option for this type of display item or one of the following:

−borderwidth
Specifies the border width of this header item.

−headerbackground
Specifies the background color of this header item.

−relief
Specifies the relief type of the border of this header item.

pathName ***header delete*** *col*
Deletes the header display item for the *col*−th column.

pathName ***header exists*** *col*
Return true if an header display item exists for the *col*−th column; return false otherwise.

pathName ***header size*** *entryPath*
If an header display item exists for the *col*−th column , returns its size in a two element list of the form {*width height*}; returns an error if the header display item does not exist.

pathName ***hide*** *option ?entryPath?*
Makes some of entries invisible invisible without deleting them. *Option* can be one of the following:

entry
Hides the list entry identified by *entryPath*.

Currently only the **entry** option is supported. Other options will be added in the next release.

pathName ***indicator*** *option entryPath ?args ...?*

Manipulates the indicator on the list entries. An indicator is usually a small display item (such as an image) that is displayed to the left to an entry to indicate the status of the entry. For example, it may be used to indicator whether a directory is opened or closed. *option* can be one of the following:

pathName ***indicator cget*** *entryPath option*
: If the list entry given by *entryPath* has an indicator, returns the value of the specified *option* of the indicator. If the indicator doesn't exist, returns an error.

pathName ***indicator configure*** *entryPath ?option? ?value option value ...?*
: Query or modify the configuration options of the indicator display item of the entry specified by *entryPath*. The indicator item must exist, or an error will result. If no *option* is specified, returns a list describing all of the available options for the indicator display item (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **indicator create** widget command. The exact set of options depends on the value of the **−itemtype** option passed to the the **indicator create** widget command when this display item was created.

pathName ***indicator create*** *entryPath ?−itemtype type? ?option value ...?*
: Creates a new display item as the indicator for the entry specified by *entryPath*. If an indicator display item already exists for this entry, it will be replaced by the new item. An optional parameter *−itemtype* can be used to specify what type of display item should be created. If the *−itemtype* is not given, then by default the type specified by this HList widget's **−itemtype** option is used. Additional parameters, in *option−value* pairs, can be passed to configure the appearance of the display item. Each *option−value* pair must be a valid option for this type of display item.

pathName ***indicator delete*** *entryPath*
: Deletes the indicator display item for the entry given by *entryPath*.

pathName ***indicator exists*** *entryPath*
: Return true if an indicator display item exists for the entry given by *entryPath*; return false otherwise.

pathName ***indicator size*** *entryPath*
: If an indicator display item exists for the entry given by *entryPath*, returns its size in a two element list of the form {*width height*}; returns an error if the indicator display item does not exist.

pathName ***info*** *option arg ...*
: Query information about the HList widget. *option* can be one of the following:

pathName ***info anchor***
: Returns the entryPath of the current anchor, if any, of the HList widget. If the anchor is not set, returns the empty string.

pathName ***info bbox*** *entryPath*

Returns a list of four numbers describing the visible bounding box of the entry given *entryPath*. The first two elements of the list give the x and y coordinates of the upper−left corner of the screen area covered by the entry (specified in pixels relative to the widget) and the last two elements give the lower−right corner of the area, in pixels. If no part of the entry given by index is visible on the screen then the result is an empty string; if the entry is partially visible, the result gives the only the visible area of the entry.

pathName ***info children*** *?entryPath?*

If *entrpyPath* is given, returns a list of the entryPath's of its children entries. Otherwise returns a list of the toplevel entryPath's.

pathName ***info data*** *?entryPath?*

Returns the data associated with *entryPath*.

pathName ***info dragsite***

Returns the entryPath of the current dragsite, if any, of the HList widget. If the dragsite is not set, returns the empty string.

pathName ***info dropsite***

Returns the entryPath of the current dropsite, if any, of the HList widget. If the dropsite is not set, returns the empty string.

pathName ***info exists*** *entryPath*

Returns a boolean value indicating whether the list entry *entrpyPath* exists.

pathName ***info hidden*** *entryPath*

Returns a boolean value indicating whether the list entry **entrpyPath** is hidden or not.

pathName ***info next*** *entryPath*

Returns the entryPath of the list entry, if any, immediately below this list entry. If this entry is already at the bottom of the HList widget, returns an empty string.

pathName ***info parent*** *entryPath*

Returns the name of the parent of the list entry identified by *entrpyPath*. If *entrpyPath* is a toplevel list entry, returns the empty string.

pathName ***info prev*** *entryPath*

Returns the entryPath of the list entry, if any, immediately above this list entry. If this entry is already at the top of the HList widget, returns an empty string.

pathName ***info selection***

Returns a list of selected entries in the HList widget. If no entries are selectd, returns an empty string.

pathName ***item*** *option ?args ...?*

Creates and configures the display items at individual columns the entries. The form of additional of arguments depends on the choice of *option*:

pathName ***item cget*** *entryPath col option*

Returns the current value of the configure *option* of the display item at the column designated by *col* of the entry specified by *entryPath*.

pathName ***item configure*** *entryPath col ?option? ?value option value ...?*

Query or modify the configuration options of the display item at the column designated by *col* of the entry specified by *entryPath*. If no *option* is specified, returns a list describing all of the available options for *entryPath* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **item create** widget command. The exact set of options depends on the value of the **−itemtype** option passed to the the **item create** widget command when this display item was created.

pathName ***item create*** *entryPath col ?−itemtype type? ?option value ...?*

Creates a new display item at the column designated by *col* of the entry specified by *entryPath*. An optional parameter *−itemtype* can be used to specify what type of display items should be created. If the *−itemtype* is not specified, then by default the type specified by this HList widget's **−itemtype** option is used. Additional parameters, in *option−value* pairs, can be passed to configure the appearance of the display item. Each *option−value* pair must be a valid option for this type of display item.

pathName ***item delete*** *entryPath col*

Deletes the display item at the column designated by *col* of the entry specified by *entryPath*.

pathName ***item exists*** *entryPath col*

Returns true if there is a display item at the column designated by *col* of the entry specified by *entryPath*; returns false otherwise.

pathName ***nearest*** *y*

Given a y−coordinate within the HList window, this command returns the entryPath of the (visible) HList element nearest to that y−coordinate.

pathName ***see*** *entryPath*

Adjust the view in the HList so that the entry given by *entryPath* is visible. If the entry is already visible then the command has no effect; if the entry is near one edge of the window then the HList scrolls to bring the element into view at the edge; otherwise the HList widget scrolls to center the entry.

pathName ***selection*** *option arg ...*

This command is used to adjust the selection within a HList widget. It has several forms, depending on *option*:

pathName ***selection clear*** *?from? ?to?*

When no extra arguments are given, deselects all of the list entrie(s) in this HList widget. When only *from* is given, only the list entry identified by

from is deselected. When both *from* and *to* are given, deselects all of the list entrie(s) between between *from* and *to*, inclusive, without affecting the selection state of entries outside that range.

pathName ***selection get***
This is an alias for the **info selection** widget command. ,

pathName ***selection includes*** *entryPath*
Returns 1 if the list entry indicated by *entryPath* is currently selected; returns 0 otherwise.

pathName ***selection set*** *from ?to?*
Selects all of the list entrie(s) between between *from* and *to*, inclusive, without affecting the selection state of entries outside that range. When only *from* is given, only the list entry identified by *from* is selected.

pathName ***show*** *option ?entryPath?*
Show the entries that are hidden by the **hide** command, *option* can be one of the following:

entry
Shows the list entry identified by *entryPath*.

Currently only the **entry** option is supported. Other options will be added in future releases.

pathName ***xview*** *args*
This command is used to query and change the horizontal position of the information in the widget's window. It can take any of the following forms:

pathName ***xview***
Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the HList entry is off−screen to the left, the middle 40% is visible in the window, and 40% of the entry is off−screen to the right. These are the same values passed to scrollbars via the **−xscrollcommand** option.

pathName ***xview*** *entryPath*
Adjusts the view in the window so that the list entry identified by *entryPath* is aligned to the left edge of the window.

pathName ***xview moveto*** *fraction*
Adjusts the view in the window so that *fraction* of the total width of the HList is off−screen to the left. *fraction* must be a fraction between 0 and 1.

pathName ***xview scroll*** *number what*
This command shifts the view in the window left or right according to *number* and *what*. *Number* must be an integer. *What* must be either **units** or **pages** or an abbreviation of one of these. If *what* is **units**, the view adjusts left or right by *number* character units (the width of the **0** character) on the display; if it is **pages** then the view adjusts by *number* screenfuls. If *number* is negative then characters farther to the left become visible; if it is

positive then characters farther to the right become visible.

pathName ***yview*** *?args?*

This command is used to query and change the vertical position of the entries in the widget's window. It can take any of the following forms:

pathName ***yview***

Returns a list containing two elements, both of which are real fractions between 0 and 1. The first element gives the position of the list element at the top of the window, relative to the HList as a whole (0.5 means it is halfway through the HList, for example). The second element gives the position of the list entry just after the last one in the window, relative to the HList as a whole. These are the same values passed to scrollbars via the **−yscrollcommand** option.

pathName ***yview*** *entryPath*

Adjusts the view in the window so that the list entry given by *entryPath* is displayed at the top of the window.

pathName ***yview moveto*** *fraction*

Adjusts the view in the window so that the list entry given by *fraction* appears at the top of the window. *Fraction* is a fraction between 0 and 1; 0 indicates the first entry in the HList, 0.33 indicates the entry one−third the way through the HList, and so on.

pathName ***yview scroll*** *number what*

This command adjust the view in the window up or down according to *number* and *what*. *Number* must be an integer. *What* must be either **units** or **pages**. If *what* is **units**, the view adjusts up or down by *number* lines; if it is **pages** then the view adjusts by *number* screenfuls. If *number* is negative then earlier entries become visible; if it is positive then later entries become visible.

BINDINGS

[1]

If the **−selectmode** is "browse", when the user drags the mouse pointer over the list entries, the entry under the pointer will be highlighted and the **−browsecmd** procedure will be called with one parameter, the entryPath of the highlighted entry. Only one entry can be highlighted at a time. The **−command** procedure will be called when the user double−clicks on a list entry.

[2]

If the **−selectmode** is "single", the entries will only be highlighted by mouse <ButtonRelease−1> events. When a new list entry is highlighted, the **−browsecmd** procedure will be called with one parameter indicating the highlighted list entry. The **−command** procedure will be called when the user double−clicks on a list entry.

[3]

If the **−selectmode** is "multiple", when the user drags the mouse pointer over the list entries, all the entries under the pointer will be highlighted. However, only a

contiguous region of list entries can be selected. When the highlighted area is changed, the **−browsecmd** procedure will be called with an undefined parameter. It is the responsibility of the **−browsecmd** procedure to find out the exact highlighted selection in the HList. The **−command** procedure will be called when the user double−clicks on a list entry.

[4]

If the **−selectmode** is "extended", when the user drags the mouse pointer over the list entries, all the entries under the pointer will be highlighted. The user can also make disjointed selections using <Control−ButtonPress−1>. When the highlighted area is changed, the **−browsecmd** procedure will be called with an undefined parameter. It is the responsibility of the **−browsecmd** procedure to find out the exact highlighted selection in the HList. The **−command** procedure will be called when the user double−clicks on a list entry.

[5]

Arrow key bindings: <Up> arrow key moves the anchor point to the item right on top of the current anchor item. <Down> arrow key moves the anchor point to the item right below the current anchor item. <Left> arrow key moves the anchor to the parent item of the current anchor item. <Right> moves the anchor to the first child of the current anchor item. If the current anchor item does not have any children, moves the anchor to the item right below the current anchor item.

EXAMPLE

This example demonstrates how to use an HList to store a file directory structure and respond to the user's browse events:

```
set h [tixHList .h −separator "/" −browsecmd browse \
    −selectmode single −itemtype text]
$h add /         −text /
$h add /home     −text /home
$h add /home/ioi −text /home/ioi
$h add /home/foo −text /home/foo
$h add /usr      −text /usr
$h add /usr/lib  −text /usr/lib
pack $h

proc browse {file} {
    puts "$file browsed"
}
```

BUGS

The fact that the display item at column 0 is implicitly associated with the whole entry is probably a design bug. This was done for backward compatibility purposes. The result is that there is a large overlap between the **item** command and the **add**, **addchild**, **entrycget** and **entryconfigure** commands. Whenever multiple columns exist, the programmer should use ONLY the **item** command to create and configure the display items in each column; the **add**, **addchild**, **entrycget** and **entryconfigure** should be used ONLY to create and configure entries.

KEYWORDS

hierarchical listbox, widget

tixInputOnly − Create and manipulate TIX InputOnly widgets

SYNOPSIS

tixInputOnly *pathName* ?*options*?

STANDARD OPTIONS

−cursor −width
−height

WIDGET−SPECIFIC OPTIONS

TixInputOnly does not have any widget specific options.

DESCRIPTION

The **tixInputOnly** command creates a new window (given by the *pathName* argument) and makes it into a **tixInputOnly** widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the **tixInputOnly** such as its cursor or width.

TixInputOnly widgets are not visible to the user. The only purpose of **TixInputOnly** widgets are to accept inputs from the user, which can be done with the **bind** command.

WIDGET COMMAND

The **tixInputOnly** command creates a new Tcl command whose name is the same as the path name of the **tixInputOnly**'s window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the InputOnly widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for **tixInputOnly** widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixInputOnly** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixInputOnly** command.

BINDINGS

tixInputOnly widgets have no default bindings.

BUGS

tixInputOnly is currently implemented for the Unix version of Tix only.

KEYWORDS

input only, invisible, widget

tixNBFrame − Create and manipulate Tix NoteBook Frame widgets

SYNOPSIS

tixNBFrame *pathName* ?*options*?

STANDARD OPTIONS

−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−disabledForeground
−font, font, Font
−foreground or −fg, foreground, Foreground
−height
−highlightColor
−highlightThickness
−relief, relief, Relief
−takeFocus
−width

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−backpagecolor***
Database Name: ***backPageColor***
Database Class: ***BackPageColor***
Specifies the color for the extra space on the row of tabs which is not covered by any page tabs.

Command−Line Name: ***−focuscolor***
Database Name: ***focusColor***
Database Class: ***FocusColor***
Specifies the color for the focus highlight.

Command−Line Name: ***−inactivebackground***
Database Name: ***inactiveBackground***
Database Class: ***InactiveBackground***
Specifies the color for the inactive tabs (the active tab always have the same background color as the notebook).

Command−Line Name: ***−tabpadx***
Database Name: ***tabPadX***
Database Class: ***Pad***
The horizontal padding around the text labels on the page tabs.

Command−Line Name: ***−tabpady***
Database Name: ***tabPadY***
Database Class: ***Pad***
The vertical padding around the text labels on the page tabs.

DESCRIPTION

The NBFrame widget is used privately inside the **TixNoteBook** widget to display the page tabs. The application programmer should never create a NBFrame widget directly. The sole purpose of this maual page is to describe the options that can be used to configure the appearance of the TixNoteBook widget.

The name of the NBFrame subwidget inside the TixNoteBook widget is called **nbframe**. It can be accessed using the **subwidget** command of the TixNoteBook widget or the **−options** switch. See below for an example.

EXAMPLE

```
set nb [tixNoteBook .nb −options {
    nbframe.BackPageColor gray60
}]
$nb subwidget nbframe config −font fixed

$nb add page1 −label "Page1"
set page [$nb subwidget page1]
button $page.b1
pack $page.b1

pack $nb −expand yes −fill both
```

KEYWORDS

notebook, widget

tixTList − Create and manipulate Tix Tabular List widgets

SYNOPSIS

tixTList *pathName* ?*options*?

STANDARD OPTIONS

−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−font, font, Font
−foreground or −fg, foreground, Foreground
−height
−highlightColor −highlightThickness
−relief, relief, Relief
−selectBackground
−selectForeground
−width
−xScrollCommand
−yScrollCommand

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browsecmd***
Database Class: ***BrowseCmd***

Specifies a TCL command to be executed when the user browses through the entries in the TList widget.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the TCL command to be executed when the user invokes a list entry in the TList widget. Normally the user invokes a list entry by double−clicking it or pressing the Return key.

Command−Line Name: ***−foreground***
Database Name: ***foreground***
Database Class: ***Foreground***

Specifies the default foreground color for the list entries.

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***
Specifies the desired height for the window in number of characters.

Command−Line Name: ***−itemtype***
Database Name: ***itemType***
Database Class: ***ItemType***
Specifies the default type of display item for this TList widget. When you call the **insert** widget commands, display items of this type will be created if the **−itemtype** option is not specified .

Command−Line Name: ***−orient***
Database Name: ***orient***
Database Class: ***Orient***
Specifies the order of tabularizing the list entries. When set to "**vertical**", the entries are arranged in a column, from top to bottom. If the entries cannot be contained in one column, the remaining entries will go to the next column, and so on. When set to "**horizontal**", the entries are arranged in a row, from left to right. If the entries cannot be contained in one row, the remaining entries will go to the next row, and so on.

Command−Line Name: ***−padx***
Database Name: ***padX***
Database Class: ***Pad***
The default horizontal padding for list entries.

Command−Line Name: ***−padx***
Database Name: ***padY***
Database Class: ***Pad***
The default vertical padding for list entries.

Command−Line Name: ***−selectbackground***
Database Name: ***selectBackground***
Database Class: ***SelectBackground***
Specifies the background color for the selected list entries.

Command−Line Name: ***−selectborderwidth***
Database Name: ***selectBorderWidth***
Database Class: ***BorderWidth***
Specifies a non−negative value indicating the width of the 3−D border to draw around selected items. The value may have any of the forms acceptable to **Tk_GetPixels**.

Command−Line Name: ***−selectforeground***
Database Name: ***selectForeground***
Database Class: ***SelectForeground***
Specifies the foreground color for the selected list entries.

Command−Line Name: ***−selectmode***
Database Name: ***selectMode***
Database Class: ***SelectMode***

Specifies one of several styles for manipulating the selection. The value of the option may be arbitrary, but the default bindings expect it to be either **single**, **browse**, **multiple**, or **extended**; the default value is **single**.

Command−Line Name: ***−sizecmd***
Database Name: ***sizeCmd***
Database Class: ***SizeCmd***
Specifies a TCL script to be called whenever the TList widget changes its size. This command can be useful to implement "user scroll bars when needed" features.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***
Specifies whether the TList command should react to user actions. When set to "**normal**", the TList reacts to user actions in the normal way. When set to "**disabled**", the TList can only be scrolled, but its entries cannot be selected or activated.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window in characters.

DESCRIPTION

The **tixTList** command creates a new window (given by the *pathName* argument) and makes it into a TList widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the TList widget such as its cursor and relief.

The TList widget can be used to display data in a tabular format. The list entries of a TList widget are similar to the entries in the Tk listbox widget. The main differences are (1) the TList widget can display the list entries in a two dimensional format and (2) you can use graphical images as well as multiple colors and fonts for the list entries.

Each list entry is identified by an **index**, which can be in the following forms:

number
An integer that indicates the position of the entry in the list. 0 means the first position, 1 means the second position, and so on.

end
Indicates the end of the listbox. For some commands this means just after the last entry; for other commands it means the last entry.

@*x,y*
Indicates the element that covers the point in the listbox window specified by x and y (in pixel coordinates). If no element covers that point, then the closest element to that point is used.

DISPLAY ITEMS

Each list entry in an TList widget is associated with a **display item**. The display item determines what visual information should be displayed for this list entry. Please see the **tixDisplayStyle** manual page for a list of all display items.

When a list entry is created by the **insert** command, the type of its display item is determined by the **−itemtype** option passed to these commands. If the **−itemtype** is omitted, then by default the type specified by this TList widget's **−itemtype** option is used.

WIDGET COMMAND

The **tixTList** command creates a new Tcl command whose name is the same as the path name of the TList widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the TList widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for TList widgets:

pathName ***anchor set*** *index*
: Sets the anchor to the list entry identified by *index*. The anchor is the end of the selection that is fixed while dragging out a selection with the mouse.

pathName ***anchor clear***
: Removes the anchor, if any, from this TList widget. This only removes the surrounding highlights of the anchor entry and does not affect its selection status.

pathName ***cget*** *option*
: Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixTList** command.

pathName ***configure*** *?option? ?value option value ...?*
: Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixTList** command.

pathName ***delete*** *from ?to?*
: Deletes one or more list entries between the two entries specified by the indices *from* and *to*. If *to* is not specified, deletes the single entry specified by *from*.

pathName ***dragsite set*** *index*
: Sets the dragsite to the list entry identified by *index*. The dragsite is used to indicate the source of a drag−and−drop action. Currently drag−and−drop functionality has not been implemented in Tix yet.

pathName ***dragsite clear***

Remove the dragsite, if any, from the this TList widget. This only removes the surrounding highlights of the dragsite entry and does not affect its selection status.

pathName ***dropsite set*** *index*

Sets the dropsite to the list entry identified by *index*. The dropsite is used to indicate the target of a grag−and−drop action. Currently drag−and−drop functionality has not been implemented in Tix yet.

pathName ***dropsite clear***

Remove the dropsite, if any, from the this TList widget. This only removes the surrounding highlights of the dropsite entry and does not affect its selection status.

pathName ***entrycget*** *index option*

Returns the current value of the configuration option given by *option* for the entry indentfied by *index*. *Option* may have any of the values accepted by the **insert** widget command.

pathName ***entryconfigure*** *index ?option? ?value option value ...?*

Query or modify the configuration options of the list entry indentfied by *index*. If no *option* is specified, returns a list describing all of the available options for *index* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **insert** widget command. The exact set of options depends on the value of the **−itemtype** option passed to the the **insert** widget command when this list entry is created.

pathName ***insert*** *index ?option value ...?*

Creates a new list entry at the position indicated by *index*. The following configuration options can be given to configure the list entry:

−itemtype *type*

Specifies the type of display item to be display for the new list entry. *type* must be a valid display item type. Currently the available display item types are **image**, **imagetext**, **text**, and **window**. If this option is not specified, then by default the type specified by this TList widget's **−itemtype** option is used.

−state

Specifies whether this entry can be selected or invoked by the user. Must be either **normal** or **disabled**.

The **insert** widget command accepts additional configuration options to configure the display item associated with this list entry. The set of additional configuration options depends on the type of the display item given by the **−itemtype** option. Please see the **tixDisplayStyle** manual page for a list of the configuration options for each of the display item types.

pathName ***info*** *option arg ...*

Query information about the TList widget. *option* can be one of the following:

pathName ***info anchor*** *index*
; Returns the index of the current anchor, if any, of the TList widget. If the anchor is not set, returns the empty string.

pathName ***info dragsite*** *index*
Returns the index of the current dragsite, if any, of the TList widget. If the dragsite is not set, returns the empty string.

pathName ***info dropsite*** *index*
Returns the index of the current dropsite, if any, of the TList widget. If the dropsite is not set, returns the empty string.

pathName ***info selection***
Returns a list of selected elements in the TList widget. If no entries are selectd, returns an empty string.

pathName ***nearest*** *x y*
Given an (x,y) coordinate within the TList window, this command returns the index of the TList element nearest to that coordinate.

pathName ***see*** *index*
Adjust the view in the TList so that the entry given by *index* is visible. If the entry is already visible then the command has no effect; if the entry is near one edge of the window then the TList scrolls to bring the element into view at the edge; otherwise the TList widget scrolls to center the entry.

pathName ***selection*** *option arg ...*
This command is used to adjust the selection within a TList widget. It has several forms, depending on *option*:

pathName ***selection clear*** *?from? ?to?*
When no extra arguments are given, deselects all of the list entrie(s) in this TList widget. When only *from* is given, only the list entry identified by *from* is deselected. When both *from* and *to* are given, deselects all of the list entrie(s) between between *from* and *to*, inclusive, without affecting the selection state of entries outside that range.

pathName ***selection includes*** *index*
Returns 1 if the list entry indicated by *index* is currently selected; returns 0 otherwise.

pathName ***selection set*** *from ?to?*
Selects all of the list entrie(s) between between *from* and *to*, inclusive, without affecting the selection state of entries outside that range. When only *from* is given, only the list entry identified by *from* is selected.

pathName ***xview*** *args*
This command is used to query and change the horizontal position of the information in the widget's window. It can take any of the following forms:

pathName ***xview***

Returns a list containing two elements. Each element is a real fraction between 0 and 1; together they describe the horizontal span that is visible in the window. For example, if the first element is .2 and the second element is .6, 20% of the TList entry is off−screen to the left, the middle 40% is visible in the window, and 40% of the entry is off−screen to the right. These are the same values passed to scrollbars via the **−xscrollcommand** option.

pathName ***xview*** *index*

Adjusts the view in the window so that the list entry identified by *index* is aligned to the left edge of the window.

pathName ***xview moveto*** *fraction*

Adjusts the view in the window so that *fraction* of the total width of the TList is off−screen to the left. *fraction* must be a fraction between 0 and 1.

pathName ***xview scroll*** *number what*

This command shifts the view in the window left or right according to *number* and *what*. *Number* must be an integer. *What* must be either **units** or **pages** or an abbreviation of one of these. If *what* is **units**, the view adjusts left or right by *number* character units (the width of the **0** character) on the display; if it is **pages** then the view adjusts by *number* screenfuls. If *number* is negative then characters farther to the left become visible; if it is positive then characters farther to the right become visible.

pathName ***yview*** *?args?*

This command is used to query and change the vertical position of the entries in the widget's window. It can take any of the following forms:

pathName ***yview***

Returns a list containing two elements, both of which are real fractions between 0 and 1. The first element gives the position of the list element at the top of the window, relative to the TList as a whole (0.5 means it is halfway through the TList, for example). The second element gives the position of the list entry just after the last one in the window, relative to the TList as a whole. These are the same values passed to scrollbars via the **−yscrollcommand** option.

pathName ***yview*** *index*

Adjusts the view in the window so that the list entry given by *index* is displayed at the top of the window.

pathName ***yview moveto*** *fraction*

Adjusts the view in the window so that the list entry given by *fraction* appears at the top of the window. *Fraction* is a fraction between 0 and 1; 0 indicates the first entry in the TList, 0.33 indicates the entry one−third the way through the TList, and so on.

pathName ***yview scroll*** *number what*

This command adjust the view in the window up or down according to *number* and *what*. *Number* must be an integer. *What* must be either **units** or **pages**. If *what* is **units**, the view adjusts up or down by *number* lines; if it is **pages** then the view adjusts by *number* screenfuls. If *number* is negative then earlier entries become visible; if it is positive then later entries become

visible.

BINDINGS

[1]

If the **−selectmode** is "browse", when the user drags the mouse pointer over the list entries, the entry under the pointer will be highlighted and the **−browsecmd** procedure will be called with one parameter, the index of the highlighted entry. Only one entry can be highlighted at a time. The **−command** procedure will be called when the user double−clicks on a list entry.

[2]

If the **−selectmode** is "single", the entries will only be highlighted by mouse <ButtonRelease−1> events. When a new list entry is highlighted, the **−browsecmd** procedure will be called with one parameter indicating the highlighted list entry. The **−command** procedure will be called when the user double−clicks on a list entry.

[3]

If the **−selectmode** is "multiple", when the user drags the mouse pointer over the list entries, all the entries under the pointer will be highlighted. However, only a contiguous region of list entries can be selected. When the highlighted area is changed, the **−browsecmd** procedure will be called with an undefined parameter. It is the responsibility of the **−browsecmd** procedure to find out the exact highlighted selection in the TList. The **−command** procedure will be called when the user double−clicks on a list entry.

[4]

If the **−selectmode** is "extended", when the user drags the mouse pointer over the list entries, all the entries under the pointer will be highlighted. The user can also make disjointed selections using <Control−ButtonPress−1>. When the highlighted area is changed, the **−browsecmd** procedure will be called with an undefined parameter. It is the responsibility of the **−browsecmd** procedure to find out the exact highlighted selection in the TList. The **−command** procedure will be called when the user double−clicks on a list entry.

EXAMPLE

This example demonstrates how to use an TList to store a list of numbers:

```
set image [tix getimage folder]
set t [tixTList .t −orient vertical]
$t insert end −itemtype imagetext −image $image −text one
$t insert end −itemtype imagetext −image $image −text two
$t insert end −itemtype imagetext −image $image −text three
$t insert end −itemtype imagetext −image $image −text four
$t insert end −itemtype imagetext −image $image −text five
$t insert end −itemtype imagetext −image $image −text six
pack $t −expand yes −fill both
```

KEYWORDS

tabular listbox, widget

Chapter 3: Mega Widgets

tixBalloon − Create and manipulate tixBalloon widgets

SYNOPSIS

tixBalloon *pathName* ?*options*?

SUPER−CLASS

The **tixBalloon** class is derived from the **TixShell** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The Balloon widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−initwait***
Database Name: ***initWait***
Database Class: ***InitWait***

In milliseconds. Specifies how long the balloon should wait before popping up in a widget.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***

Specifies the which help message to display when the mouse pointer enters a widget associated with this balloon. Valid options are **both**: display both the balloon message and the status bar message, **balloon**: display only the balloon message, **status**: display only the status bar message and **none**: display no messages.

Command−Line Name: ***−statusbar***
Database Name: ***statusBar***
Database Class: ***statusBar***

Specifies the widget to use as the status bar of this balloon. This widget must have a "−text" option. Usually a label widget is used.

SUBWIDGETS

Name: **label**
Class: **Label**

The label widget that shows the little arrow bitmap in the pop−up balloon window.

Name: **message**
Class: **Label**

The message widget that shows the descriptive message in the the pop−up balloon window.

DESCRIPTION

The **tixBalloon** command creates a new window (given by the *pathName* argument) and makes it into a Balloon widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the Balloon widget such as its cursor and relief. The Balloon widget can be used to show popped−up messages that describe the functions of the widgets in an application. A Balloon widget can be bound to a number of widgets. When the user moves the cursor inside a widget to which a Balloon widget has been bound, a small pop−up window with a descriptive message will be shown on the screen.

WIDGET COMMANDS

The **tixBalloon** command creates a new Tcl command whose name is the same as the path name of the Balloon widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the Balloon widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for Balloon widgets:

pathName ***bind*** *widget ?option value ... ?*

Binds the Balloon widget to the *widget*. The messages to be shown can be passed as extra arguments to this command in *option value* pairs. Possible options: **−balloonmsg** specifies the string to show on the pop−up balloon window; **−statusmsg** specifies the string to show on the status bar; **−msg** specifies a string to show on both the balloon window and the stats bar window. When used together, the **−msg** option has a lower precedence than the **−balloonmsg** and **−statusmsg** options. The **bind** command can also be used to change the messages after the initial bindings were set. Example:

```
button .b
set bal [tixBalloon .bal]

# Add balloon binding
$bal bind .b −msg "This is a button"

# ...

# Change the balloon binding
$bal bind .b −msg "This is a useful button"
```

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixBalloon** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixBalloon** command.

pathName ***unbind*** *widget*
Cancels the Balloon widget's binding with *widget*.

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

After a widget has be bound to a Balloon widget, when the user moves the cursor into this widget, the Balloon widget is activated: if the **−balloonmsg** option of this widget is set, the balloon window pops up; if the **−statusmsg** option of this widget is set, the message will be displayed in the status bar widget.

When the user moves the cursor out of the widget, the Balloon widget is de−activated: the balloon window is withdrawn and the status−bar message removed.

KEYWORDS

Tix, context−sensitive help, balloon, widget

tixButtonBox − Create and manipulate Tix ButtonBox widgets

SYNOPSIS

tixButtonBox *pathName* ?*options*?

STANDARD OPTIONS

−anchor, anchor, Anchor
−background or −bg, background, Background
−borderWidth

−cursor, cursor, Cursor
−relief, relief, Relief

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−orientation***
Database Name: ***orientation***
Database Class: ***Orientation***
Static Option. Specifies the orientation of the button subwidgets. Only the values "horizontal" and "vertical" are recognized.

Command−Line Name: ***−padx***
Database Name: ***padx***
Database Class: ***Pad***
Specifies the horizontal padding between two neighboring button subwidgets in the ButtonBox widget.

Command−Line Name: ***−pady***
Database Name: ***pady***
Database Class: ***Pad***
Specifies the vertical padding between two neighboring button subwidgets in the ButtonBox widget.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***
Specifies the state of all the buttons inside the ButtonBox widget. *Note*: Setting this option using the *config* widget command will enable or disable all the buttons subwidgets. Original states of the individual buttons are *not* saved. Only the values "normal" and "disabled" are recognized.

SUBWIDGETS

All the button subwidgets created as a result of the **add** command can be accessed by the **subwidget** command. They are identified by the **buttonName** parameter to the **add** command. Here is an example:

```
set bbox [tixButtonBox .bbox]
pack $bbox
$bbox add eat   −text Eat
$bbox add sleep −text Sleep
$bbox subwidget eat   config −fg green
$bbox subwidget sleep config −fg red
```

DESCRIPTION

The **tixButtonBox** command creates a new window (given by the *pathName* argument) and makes it into a ButtonBox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ButtonBox such as its cursor and relief.

The ButtonBox widget can be used as a container widget to hold the ``action'' buttons in a dialog box.

WIDGET COMMAND

The **tixButtonBox** command creates a new Tcl command whose name is the same as the path name of the ButtonBox's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ButtonBox widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ButtonBox widgets:

pathName ***add*** *buttonName ?option value ...?*

Add a new button subwidget with the name *buttonName* into the ButtonBox widget. Additional configuration options can be given to configure the new button subwidget.

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixButtonBox** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixButtonBox** command.

pathName ***invoke*** *buttonName*

Invoke the button subwidget with the name *buttonName*.

pathName ***subwidget*** *name ?args?*

When no additional arguments are given, returns the pathname of the subwidget of the specified name. When no additional arguments are given, the widget command of the specified subwidget will be called with these parameters.

BINDINGS

TixButtonBox widgets have no default bindings. The button subwidgets retain their default Tk bindings.

KEYWORDS

container widget, widget

tixCheckList − Create and manipulate tixCheckList widgets

SYNOPSIS

tixCheckList *pathName* ?*options*?

SUPER−CLASS

The **TixCheckList** class is derived from the **TixTree** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

TixCheckList supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browseCmd***
Database Class: ***BrowseCmd***

Specifies a command to call whenever the user browses on an entry (usually by single−clicking on the entry). The command is called with one argument, the pathname of the entry.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies a command to call whenever the user activates an entry (usually by double−clicking on the entry). The command is called with one argument, the pathname of the entry.

Command−Line Name: ***−radio***
Database Name: ***radio***
Database Class: ***Radio***

A Boolean value. If set to true, the user can select at most one item at a time; if set to false, the user can select as many items as possible.

SUBWIDGETS

Name: **hlist**
Class: **TixHList**

The hierarchical listbox that displays the CheckList.

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixCheckList** command creates a new window (given by the *pathName* argument) and makes it into a CheckList widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the CheckList widget such as its cursor and relief.

The CheckList widget displays a list of items to be selected by the user. CheckList acts similarly to the Tk checkbutton or radiobutton widgets, except it is capable of handling many more items than checkbuttons or radiobuttons.

The items are contained in the **hlist** subwidget. Each item may be in one of the following status: **on** (indicated by a check bitmap), **off** (indicated by a cross bitmap) **default** (indicated by a gray box bitmap) or **none**, in which case the item will not be accompanied by a bitmap. The items whose status is **on**, **off** or **default** are called the *selectable* items and can be checked or crossed by the user. All selectable entries must be of the type **imagetext**.

The items whose status is **none** cannot be checked or crossed by the user; usually they are included in the **hlist** subwidget only for explanation purposes or as separators.

Initially, all the items have a *none* status. To make an item selectable, you can call the **setstatus** command to change its status (see below).

Notice that CheckList is a subclass of the TixTree widget and thus is is capable of displaying a hierachy of selectable entries. When necessary, you can call the **setmode** method (see **TixTree**) to define the hierachical structure of the selectable entries.

WIDGET COMMANDS

The **tixCheckList** command creates a new Tcl command whose name is the same as the path name of the CheckList's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the CheckList widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for CheckList widgets:

pathName ***getselection*** *?status?*
Returns a list of items whose status matches *status*. If *status* is not specified, the list of items in the "**on**" status will be returned.

pathName ***getstatus*** *entryPath*
Returns the current status of *entryPath*.

pathName ***setstatus*** *entryPath status*

Sets the status of *entryPath* to be *status*. A bitmap will be displayed next to the entry its status is **on**, **off** or **default**.

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

EXAMPLE

This example creates several choices for the user to select.

```
set c [tixCheckList .c]
$c subwidget hlist add choice1 −itemtype imagetext −text Choice1
$c subwidget hlist add choice2 −itemtype imagetext −text Choice2
$c subwidget hlist add choice3 −itemtype imagetext −text Choice3
$c setstatus choice1 on
$c setstatus choice2 off
$c setstatus choice3 off
pack $c
```

BINDINGS

The basic mouse and keyboard bindings of the CheckList widget are the same as the bindings of the TixTree widget. In addition, the status of the entries in the CheckList are toggled under the following conditions:

[1]
When the user press the mouse button over an entry.

[2]
When the user press the <space> key over an entry.

[3]
When the user press the <Return> key over an entry.

KEYWORDS

hierarchical listbox, tree, widget

tixComboBox − Create and manipulate tixComboBox widgets

SYNOPSIS

tixComboBox *pathName* ?*options*?

SUPER−CLASS

The **TixComboBox** class is derived from the **TixLabelWidget** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

TixComboBox supports all the standard options of a frame widget. See the options(n) manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−anchor***
Database Name: ***anchor***
Database Class: ***Anchor***

Specifies how the string inside the entry subwidget should be aligned. Only the values "w" or "e" are allowed. When set the "w", the entry is aligned to its beginning. When set to "e", it is aligned to its end.

Command−Line Name: ***−arrowbitmap***
Database Name: ***arrowBitmap***
Database Class: ***ArrowBitmap***

Specifies the bitmap to be used in the arrow button beside the entry widget. The default is an downward arrow bitmap in the file $tix_library/bitmaps/cbxarrow

Command−Line Name: ***−browsecmd***
Database Name: ***browseCmd***
Database Class: ***BrowseCmd***

Specifies the command to be called when the user browses through the listbox. This command can be used to provide instant feedback when the user examines items in the listbox before committing a choice.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the ComboBox is invoked or when the **−value** of the ComboBox is changed.

Command−Line Name: ***−crossbitmap***
Database Name: ***crossBitmap***
Database Class: ***CrossBitmap***

Specifies the bitmap to be used in the "cross" button to the left of the entry widget. The default is a bitmap in the file $tix_library/bitmaps/cross

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***
Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the ComboBox. changes.

Command−Line Name: ***−disabledforeground***
Database Name: ***disabledforeground***
Database Class: ***DisabledForeground***
Specifies the foreground color to be used when the ComboBox is disabled.

Command−Line Name: ***−dropdown***
Database Name: ***dropdown***
Database Class: ***Dropdown***
A Boolean value specifying the style of the ComboBox. When set to "true", the listbox is only displayed temporarily when the arrow button is pressed. When set to "false", the listbox is always displayed.

Command−Line Name: ***−editable***
Database Name: ***editable***
Database Class: ***Editable***
Specifies whether the user is allowed to type into the entry subwidget of the ComboBox.

Command−Line Name: ***−fancy***
Database Name: ***fancy***
Database Class: ***Fancy***
A Boolean value specifying whether the cross and tick button subwidgets should be shown.

Command−Line Name: ***−grab***
Database Name: ***grab***
Database Class: ***Grab***
Specifies the pointer grabbing policy when the listbox is popped up. Only values "global", "local" or "none" are allowed. By default global grab is used. However, when you are developing your application, you may want to use only local grabbing so that in the event of errors, your X display won't be locked up.

Command−Line Name: ***−historylimit or −histlimit***
Database Name: ***historyLimit***
Database Class: ***HistoryLimit***
Specifies how many previous user inputs can be stored in the history list.

Command−Line Name: ***−history***
Database Name: ***history***
Database Class: ***History***
A Boolean value specifying whether previous user inputs should be stored in the history list.

Command−Line Name: ***−label***
Database Name: ***label***
Database Class: ***Label***
Specifies the string to display as the label of this ComboBox widget.

Command−Line Name: ***−labelside***

Database Name: ***labelSide***
Database Class: ***LabelSide***

Specifies where the label should be displayed relative to the entry subwidget. Valid options are: **top**, **left**, **right**, **bottom**, **none** or **acrosstop**.

Command−Line Name: ***−listcmd***
Database Name: ***listCmd***
Database Class: ***listCmd***

Specifies a TCL command to be called every time when the listbox pops up. This option allows you to fill up the listbox on−demand. This option is ignored when the listbox is not in the **dropdown** style.

Command−Line Name: ***−listwidth***
Database Name: ***listWidth***
Database Class: ***listWidth***

If set, this option controls the width of the listbox subwidget when it is popped up. The option is ignored when the listbox is not in the **dropdown** style.

Command−Line Name: ***−prunehistory***
Database Name: ***prunehistory***
Database Class: ***PruneHistory***

Specifies whether duplicated previous user inputs should be pruned from the the history list. Only Boolean values are allowed.

Command−Line Name: ***−selection***
Database Name: ***selection***
Database Class: ***Selection***

Contains the selection in the ComboBox (the string displayed in the entry subwidget). Depending on the **−selectmode**, the selection of a ComboBox may be different than its **−value**.

Command−Line Name: ***−selection***
Database Name: ***selection***
Database Class: ***Selection***

This option stores the temporary selection. When the user types in a text string inside the entry widget, that string is considered as a temporary input and is stored inside the **−selection** option. The **−value** option is updated only when the user presses the return key.

Command−Line Name: ***−selectmode***
Database Name: ***selectMode***
Database Class: ***SelectMode***

Specifies the how the combobox responds to the mouse button events in the listbox subwidget; can eithet be **"browse"** or **"immediate"**. The default **−selectmode** is "browse". See the **BINDINGS** section below.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***

Specifies the whether the ComboBox is normal or disabled. Only the values "normal" and "disabled" are recognized.

Command−Line Name: ***−tickbitmap***

Database Name: ***tickBitmap***
Database Class: ***tickBitmap***
Specifies the bitmap to be used in the "tick" button to the left of the entry widget. The default is a bitmap in the file $tix_library/bitmaps/tick

Command−Line Name: ***−validatecmd***
Database Name: ***validateCmd***
Database Class: ***ValidateCmd***
Specifies a TCL command to be called when the **−value** of the ComboBox is about to change. This command is called with one parameter −− the new **−value** entered by the user. This command is to validate this new value by returning a value it deems valid.

Command−Line Name: ***−value***
Database Name: ***value***
Database Class: ***Value***
Specifies the string to be displayed in the entry subwidget of the ComboBox. When queried, the returned value is the last value selected by the user. When the **−value** option is changed as a result of the **config −value** widget command, the TCL command specified by the **−command** option is called.

Command−Line Name: ***−variable***
Database Name: ***variable***
Database Class: ***Variable***
Specifies the global variable in which the value of the ComboBox should be stored. The value of the ComboBox will be automatically updated when this variable is changed.

SUBWIDGETS

Name: **arrow**
Class: **Button**

The down arrow button.

Name: **cross**
Class: **Button**

The cross button. Available only when **−fancy** is set.

Name: **entry**
Class: **Entry**

The entry that shows the value of this **tixControl**.

Name: **label**
Class: **Label**

The label subwidget.

Name: **listbox**
Class: **Listbox**

The listbox that holds all the list entries.

Name: **slistbox**
Class: **TixScrolledListBox**

The scrolled−listbox that provides the scrollbars.

Name: **tick**
Class: **Button**

The tick button. Available only when **−fancy** is set.

DESCRIPTION

The **tixComboBox** command creates a new window (given by the *pathName* argument) and makes it into a **tixComboBox** widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ComboBox such as its cursor and relief. The Tix ComboBox widget is similar to the combo box control in MS Windows. The user can select a choice by either typing in the entry subwdget or selecting from the listbox subwidget.

WIDGET COMMANDS

The **tixComboBox** command creates a new Tcl command whose name is the same as the path name of the ComboBox's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ComboBox widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ComboBox widgets:

pathName ***addhistory*** *string*
Add the string to the beinning of the listbox.

pathName ***appendhistory*** *string*
Append the string to the end of the listbox.

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixComboBox** command.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixComboBox** command.

pathName ***flash*** *index string*

Flashes the ComboBox. **flash** is usually called by a *−command* procedure to acknowledge to the user that he has selected a value for the ComboBox.

pathName ***insert*** *index string*

Insert the *string* into the listbox at the specified index. *index* must be a valid listbox index.

pathName ***pick*** *index*

Set the (*index*)th item in the listbox to be the current value of the ComboBox. As a result, the *value* of the ComboBox is changed and the TCL command sepcified by the *−command* option will be called.

pathName ***subwidget*** *name ?args?*

When no options are given, returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

[1]

If the **−selectmode** is "immediate", when the user enters a keystroke, clicks on an item or drags the mouse pointer in the listbox, the **−value** of the ComboBox will be immediately set to this item and the **−command** procedure will be called.

[2]

If the **−selectmode** is "browse", when the user enters a keystroke, clicks on an item or drags the mouse pointer in the listbox, the **−selection** of the ComboBox will be immediately set to the new content of the entry subwidget; also the **−browsecmd** procedure will be called. The **−value** option will be changed only when the user invokes the ComboBox (see [3] below). If the user presses the <Escape> key at any time, any new **−selection** will be ignored and the text inside the entry subwidget will be restored to the current **−value** of the ComboBox.

[3]

If the **−dropdown** option is true, the user can invoke the ComboBox by releasing the left mouse button over the desired item in the listbox. If the **−dropdown** option is false, the user can invoke the ComboBox by double−clicking over the desired item in the listbox. In both cases, the user can also invoke the listbox by pressing the <Return> or <Tab> key inside the entry subwidget, or switching the input focus to another widget inside the same toplevel widget

BUGS

Starting from Tix vetsion 4.0, the default **−value** of the ComboBox is the empty string. If you want the ComboBox to show a string by default, you must configure its **−value** option explicitly.

KEYWORDS

ComboBox, listbox, widget

tixControl − Create and manipulate tixControl widgets

SYNOPSIS

tixControl *pathName* ?*options*?

SUPER−CLASS

The **TixControl** class is derived from the **TixLabelWidget** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The Control widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−allowempty***
Database Name: ***allowEmpty***
Database Class: ***AllowEmpty***
Specifies whether the Control widget should allow the empty string as a valid input.

Command−Line Name: ***−autorepeat***
Database Name: ***autorepeat***
Database Class: ***AutoRepeat***
Specifies whether the Control widget should have autorepeat behavior. If set to be "true", the value of the Control widget will be automatically incremented or decremented when the user holds down the mouse button over the arrow buttons. Only values "true" and "false" will be recognized.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***
Specifies the command to be called when the **−value** option of the Control widget is changed. The command will be called with one arguments −− the new value of the Control widget.

Command−Line Name: ***−decrcmd***
Database Name: ***decrCmd***
Database Class: ***DecrCmd***
Specifies a TCL command to be called when the the user presses the down−arrow button subwidget. This command is called with one parameter −− the current **−value** of this Control widget. This command is to decrement this value by one step, according to its own definition of "decrement", and return the decremented value,

which will be stored in the **−value** of this Control widget.

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***
Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the Control widget changes.

Command−Line Name: ***−disableforeground***
Database Name: ***disableForeground***
Database Class: ***DisableForeground***

The foreground color to use for of the entry subwidget when the Control widget is disabled.

Command−Line Name: ***−incrcmd***
Database Name: ***incrCmd***
Database Class: ***IncrCmd***

Specifies a TCL command to be called when the the user presses the up−arrow button subwidget. This command is called with one parameter −− the current **−value** of this Control widget. This command is to increment this value by one step, according to its own definition of "increment", and return the incremented value, which will be stored in the **−value** of this Control widget.

Command−Line Name: ***−initwait***
Database Name: ***initwait***
Database Class: ***Initwait***

Specifies how long the Control widget should wait initially before it starts to automatically increment or decrement its value in the autorepeat mode. In milliseconds.

Command−Line Name: ***−integer***
Database Name: ***integer***
Database Class: ***Integer***

A Boolean value specifying whether only integer numbers are accepted.

Command−Line Name: ***−label***
Database Name: ***label***
Database Class: ***Label***

Specifies the string to display as the label of this Control widget.

Command−Line Name: ***−labelside***
Database Name: ***labelSide***
Database Class: ***LabelSide***

Specifies where the label should be displayed relative to the entry subwidget. Valid options are: **top**, **left**, **right**, **bottom**, **none** or **acrosstop**.

Command−Line Name: ***−max or −ulimit***
Database Name: ***max***
Database Class: ***Max***

Specifies the upper limit of the value of the Control widget. When set to empty string, the Control widget has no upper limit.

Command−Line Name: ***−min or −llimit***
Database Name: ***min***
Database Class: ***Min***

Specifies the lower limit of the value of the Control widget.When set to empty string, the Control widget has no lower limit.

Command−Line Name: ***−repeatrate***
Database Name: ***repeatRate***
Database Class: ***RepeatRate***

Specifies how often the value of the Control widget should be incremented or decremented when it is in the autorepeat mode. In milliseconds.

Command−Line Name: ***−selectmode***
Database Name: ***selectMode***
Database Class: ***SelectMode***

Specifies how the Control widget should react to `<KeyPress>` events. When set to "immediate", any user keyboard inputs will immediately change the **−value** option. When set to "normal", the user keyboard inputs will be copied to the **−value** option only if the `<Return>` key is pressed or the keyboard focus is changed. The use of the immediate mode is discouraged. For effective use of the Control widget, one should use the normal mode together with the **update** widget command (see below).

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***

Specifies the whether the Control widget is normal or disabled. Only the values "normal" and "disabled" are recognized.

Command−Line Name: ***−step***
Database Name: ***step***
Database Class: ***Step***

Specifies by how much the value of the Control widget should be incremented or decrmented when the user press the arrow buttons.

Command−Line Name: ***−validatecmd***
Database Name: ***validateCmd***
Database Class: ***ValidateCmd***

Specifies a TCL command to be called when the −value of the Control widget is about to change. This command is called with one parameter −− the new **−value** entered by the user. This command is to validate this new value by returning a value it deems valid.

Command−Line Name: ***−value***
Database Name: ***value***
Database Class: ***Value***

Specifies the value of the Control widget.

Command−Line Name: ***−variable***
Database Name: ***variable***
Database Class: ***Variable***

Specifies the global variable in which the value of the Control widget should be stored. The value of the Control widget will be automatically updated when this variable is changed.

SUBWIDGETS

Name: **decr**
Class: **Button**

The down arrow button.

Name: **entry**
Class: **Entry**

The entry that shows the value of this Control widget.

Name: **incr**
Class: **Button**

The up arrow button.

Name: **label**
Class: **Label**

The label subwidget.

DESCRIPTION

The **tixControl** command creates a new window (given by the *pathName* argument) and makes it into a Control widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the Control widget such as its cursor and relief.

The tixControl widget is sometimes referred to a "spinbox" widget. It is generally used to control a value. The user can adjust the value by pressing the two arrow buttons or by entering the value directly into the entry. The new value will be checked against the user−defined upper and lower limits.

WIDGET COMMANDS

The **tixControl** command creates a new Tcl command whose name is the same as the path name of the Control widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the Control widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for Control widgets:

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixControl** command.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see

Tk_ConfigureInfo for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option*−*value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixControl** command.

pathName ***decr***
Decrements the value of the Control widget by the step specified by the −*step* option.

pathName ***incr***
Increments the value of the Control widget by the step specified by the −*step* option.

pathName ***invoke***
Causes the command specified by the −*command* option to be invoked.

pathName ***update***
If the user has modified the entry using keyboard inputs, the update command will **update** the **−value** of this Control widget. When the Control widget's **−selectmode** option is set to "normal", one should call the **update** command on this widget before examining its **−value** option. This command has no effect in if the **−selectmode** option is set to "immediate".

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

When the user presses the up/down arrow buttons (or press the <Up> and <Down> arrow keys on the keyboard), the value of the tixControl widget is adjusted according to the **−validatecmd**, **−incrcmd**, **−decrcmd**, **−step**, **−max** and **−min** options.

KEYWORDS

spinbox, widget

tixDirList − Create and manipulate tixDirList widgets

SYNOPSIS

tixDirList *pathName* ?*options*?

SUPER−CLASS

The **TixDirList** class is derived from the **TixScrolledHList** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

TixDirList supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browseCmd***
Database Class: ***BrowseCmd***

Specifies a command to call whenever the user browses on a directory (usually by single−clicking on the name of the directory). The command is called with one argument, the complete pathname of the directory.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the user activates on a directory (usually by double−clicking on the name of the directory). The command is called with one argument, the complete pathname of the directory.

Command−Line Name: ***−dircmd***
Database Name: ***dircmd***
Database Class: ***DirCmd***

Specifies the TCL command to be called when a directory listing is needed for a particular directory. If this option is not specified, by default the DirList widget will attempt to read the directory as a Unix directory. On special occasions, the application programmer may want to supply a special method for reading directories: for example, when he needs to list remote directories. In this case, the **−dircmd** option can be used. The specified command accepts two arguments: the first is the name of the directory to be listed; the second is a Boolean value indicating whether hidden sub−directories should be listed. This command returns a list of names of the sub−directories of this directory. For example:

```
proc read_dir {dir show_hidden} {
    if {$dir == "C:\"} {
        return {DOS NORTON WINDOWS}
    } else {
        return {}
    }
}
```

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***

Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the DirList widget changes.

Command−Line Name: ***−showhidden***
Database Name: ***showHidden***
Database Class: ***ShowHidden***

Specifies whether hidden directories should be shown. By default, a directory name starting with a period "." is considered as a hidden directory. This rule can be overridden by supplying an alternative **−dircmd** option.

Command−Line Name: ***−root***
Database Name: ***root***
Database Class: ***Root***

Specifies the name of the root directory. Usually this is "/" under Unix machines, but can be changed to "**C:**" in DOS environments.

Command−Line Name: ***−rootname***
Database Name: ***rootName***
Database Class: ***RootNAme***

Specifies a text string to display at the root directory. If unspecified, the text string will be the same as the string specified by **−root**.

Command−Line Name: ***−value or −directory***
Database Name: ***value***
Database Class: ***Value***

Specifies the name of the current directory to be displayed in the DirList widget.

SUBWIDGETS

Name: **hlist**
Class: **TixHList**

The hierarchical listbox that displays the directory listing.

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixDirList** command creates a new window (given by the *pathName* argument) and makes it into a DirList widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the DirList such as its cursor and relief. The DirList widget displays a list view of a directory, its previous directories and

its sub−directories. The user can choose one of the directories displayed in the list or change to another directory.

WIDGET COMMANDS

The **tixDirList** command creates a new Tcl command whose name is the same as the path name of the DirList's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the DirList widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for DirList widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixDirList** command.

pathName ***chdir*** *dir*

Change the current directory to *dir*.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixDirList** command.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

The mouse and keyboard bindings of the DirList widget are the same as the bindings of the HList widget.

KEYWORDS

directory list, widget

tixDirSelectDialog − Create and manipulate directory selection dialogs.

SYNOPSIS

tixDirSelectDialog *pathName* ?*options*?

STANDARD OPTIONS

TixDirSelectDialog supports all the standard options of a toplevel widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the user selects a directory in the dialog box. The command is called with one extra argument, the complete pathname of the directory. If the user cancels the selection, this command is not called.

SUBWIDGETS

Name: **dirbox**
Class: **TixDirSelectBox**

The DirSelectBox widget that consists of the main part of the dialog.

Name: **cancel**
Class: **Button**

The "Cancel" button.

Name: **ok**
Class: **Buton**

The "OK" button.

DESCRIPTION

The **tixDirSelectDialog** command creates a new window (given by the *pathName* argument) and makes it into a DirSelectDialog widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the DirSelectDialog such as its cursor and relief. The DirSelectDialog widget presents the directories in the file system in a dialog window. The user can use this dialog window to navigate through the file system to select the desired directory.

WIDGET COMMANDS

The **tixDirSelectDialog** command creates a new Tcl command whose name is the same as the path name of the DirSelectDialog's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the DirSelectDialog widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for DirSelectDialog widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixDirSelectDialog** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixDirSelectDialog** command.

pathName ***popup***

Pops up the DirSelectDialog widget on the screen.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

EXAMPLE

```
set dlg [tixDirSelectDialog .dlg −command SelectDir]
$dlg popup

proc SelectDir {dir} {
    puts "You have selected \"$dir\""
}
```

KEYWORDS

directory selector, widget, dialog

tixDirTree − Create and manipulate tixDirTree widgets

SYNOPSIS

tixDirTree *pathName* ?*options*?

SUPER−CLASS

The **TixDirTree** class is derived from the **TixScrolledHList** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

TixDirTree supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browseCmd***
Database Class: ***BrowseCmd***

Specifies a command to call whenever the user browses on a directory (usually by single−clicking on the name of the directory). The command is called with one argument, the complete pathname of the directory.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the user activates on a directory (usually by double−clicking on the name of the directory). The command is called with one argument, the complete pathname of the directory.

Command−Line Name: ***−dircmd***
Database Name: ***dircmd***
Database Class: ***DirCmd***

Specifies the TCL command to be called when a directory listing is needed for a particular directory. If this option is not specified, by default the DirTree widget will attempt to read the directory as a Unix directory. On special occasions, the application programmer may want to supply a special method for reading directories: for example, when he needs to list remote directories. In this case, the **−dircmd** option can be used. The specified command accepts two arguments: the first is the name of the directory to be listed; the second is a Boolean value indicating whether hidden sub−directories should be listed. This command returns a list of names of the sub−directories of this directory. For example:

```
proc read_dir {dir show_hidden} {
    if {$dir == "C:\"} {
        return {DOS NORTON WINDOWS}
    } else {
        return {}
    }
}
```

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***
Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the DirTree widget changes.

Command−Line Name: ***−showhidden***
Database Name: ***showHidden***
Database Class: ***ShowHidden***

Specifies whether hidden directories should be shown. By default, a directory name starting with a period "." is considered as a hidden directory. This rule can be overridden by supplying an alternative **−dircmd** option.

Command−Line Name: ***−value or −directory***
Database Name: ***value***
Database Class: ***Value***

Specifies the name of the current directory to be displayed in the DirTree widget.

SUBWIDGETS

Name: **hlist**
Class: **TixHList**

The hierarchical listbox that displays the directory listing.

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixDirTree** command creates a new window (given by the *pathName* argument) and makes it into a DirTree widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the DirTree such as its cursor and relief. The DirTree widget displays a list view of a directory, its previous directories and its sub−directories. The user can choose one of the directories displayed in the list or change to another directory.

WIDGET COMMANDS

The **tixDirTree** command creates a new Tcl command whose name is the same as the path name of the DirTree's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the DirTree widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for DirTree widgets:

pathName ***cget*** *option*
: Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixDirTree** command.

pathName ***chdir*** *dir*
: Change the current directory to *dir*.

pathName ***configure*** *?option? ?value option value ...?*
: Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixDirTree** command.

pathName ***subwidget*** *name ?args?*
: When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

The mouse and keyboard bindings of the DirTree widget are the same as the bindings of the HList widget.

KEYWORDS

directory tree, widget

tixExFileSelectBox − Create and manipulate tixExFileSelectBox widgets

SYNOPSIS

tixExFileSelectBox *pathName* ?*options*?

SUPER−CLASS

The **TixExFileSelectBox** class does not have a super−class.

STANDARD OPTIONS

TixExFileSelectBox supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browseCmd***
Database Class: ***BrowseCmd***

Specifies a command to call whenever the user browses on a filename in the file listbox (usually by single−clicking on the filename). The command is called with one argument, the complete pathname of the file.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the user chooses on a filename the file listbox (usually by double−clicking on the filename). The command is called with one argument, the complete pathname of the file.

Command−Line Name: ***−dialog***
Database Name: ***dialog***
Database Class: ***Dialog***

Specifies a dialog box which contains this ExFileSelectBox widget. The dialog box must be a widget of the class TixShell or its descendant classes. *This is an internal option and should not be used by application programmers.*

Command−Line Name: ***−dircmd***
Database Name: ***dircmd***
Database Class: ***DirCmd***

Specifies the TCL command to be called when a file listing is needed for a particular directory. If this option is not specified, by default the ExFileSelectBox widget will attempt to read the directory as a Unix directory. On special occasions, the application programmer may want to supply a special method for reading directories: for example, when he needs to list remote files. In this case, the **−dircmd** option can be used. The specified command accepts three arguments: the first is the name of the directory to be listed; the second is a list of file patterns, the third is a Boolean value indicating whether hidden files should be listed. This command returns a list of names of the files of this directory which match with the file patterns.

Command−Line Name: ***−directory or −dir***
Database Name: ***directory***
Database Class: ***Directory***

Specifies the current directory whose files and sub−directories are displayed in the ExFileSelectBox.

Command−Line Name: ***−disablecallback***

Database Name: ***disableCallback***
Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the ExFileSelectBox widget changes.

Command−Line Name: ***−filetypes***
Database Name: ***fileTypes***
Database Class: ***FileTypes***

Specifies the file types that can be selected from the "List Files of Type:" ComboBox subwidget. The value of this option must be a TCL list; each item of this list must in turn be a list of two elements. The first element is a list of file patterns. The second element is a string that describe these file patterns. For example:

```
tixExFileSelectBox .box −filetypes {
{{*} {All files}}
{{*.txt} {Text files}}
{{*.c *.h} {C source files}}
}
```

Command−Line Name: ***−showhidden***
Database Name: ***showHidden***
Database Class: ***ShowHidden***

Specifies whether hidden directories should be shown. By default, a directory name starting with a period "." is considered as a hidden directory.

Command−Line Name: ***−pattern***
Database Name: ***pattern***
Database Class: ***Pattern***

Specifies whether the file pattern(s) to match with the files in the current directory. One or more file patterns can be given at the same time. For example, {*.c *.h} will match all files that have either the ".h" or ".c" extensions.

Command−Line Name: ***−value or −selection***
Database Name: ***value***
Database Class: ***Value***

Specifies the name of the filename currently selected by the user.

SUBWIDGETS

Name: **cancel**
Class: **Button**

The button widget with the "Cancel" label.

Name: **dir**
Class: **TixComboBox**

The ComboBox subwidget under the "Directories" heading.

Name: **dirlist**
Class: **TixDirList**

The DirList subwidget that shows the hierarchical list of directories.

Name: **file**
Class: **TixComboBox**

The ComboBox subwidget under the "Files" heading.

Name: **filelist**
Class: **TixScrolledListBox**

The ScrolledListBox subwidget that shows the list of filenames.

Name: **hidden**
Class: **Checkbutton**

The checkbutton widget with the "Show Hidden Files" label.

Name: **ok**
Class: **Button**

The button widget with the "OK" label.

Name: **types**
Class: **TixComboBox**

The ComboBox subwidget under the "List Files of Type" heading.

DESCRIPTION

The **tixExFileSelectBox** command creates a new window (given by the *pathName* argument) and makes it into a ExFileSelectBox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ExFileSelectBox such as its cursor and relief. The ExFileSelectBox widget is usually embedded in a tixExFileSelectDialog widget. It provides an convenient method for the user to select files. The style of the ExFileSelectBox widget is very similar to the standard file dialog in MS Windows 3.1.

WIDGET COMMANDS

The **tixExFileSelectBox** command creates a new Tcl command whose name is the same as the path name of the ExFileSelectBox's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ExFileSelectBox widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ExFileSelectBox widgets:

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixExFileSelectBox** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixExFileSelectBox** command.

pathName ***filter***
Forces the ExFileSelectBox widget to re−filter all the filenames according to the **−pattern** option.

pathName ***invoke***
Forces the ExFileSelectBox widget to perform actions as if the user has pressed the "OK" button.

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

KEYWORDS

file selector, widget

tixExFileSelectDialog − Create and manipulate tixExFileSelectDialog widgets

SYNOPSIS

tixExFileSelectDialog *pathName* ?*options*?

SUPER−CLASS

The **TixExFileSelectDialog** class does not have a super−class.

STANDARD OPTIONS

TixExFileSelectDialog supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the user chooses on a filename (usually by selecting the filename and clicking on the "OK" button"). The command is called with one argument, the complete pathname of the file.

SUBWIDGETS

Name: **fsbox**
Class: **TixExFileSelectBox**

The ExFileSelectBox subwidget embedded inside the ExFileSelectDialog.

DESCRIPTION

The **tixExFileSelectDialog** command creates a new window (given by the *pathName* argument) and makes it into a ExFileSelectDialog widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ExFileSelectDialog such as its cursor and relief. The ExFileSelectDialog widget provides an convenient method for the user to select files. The style of the ExFileSelectDialog widget is very similar to the standard file dialog in MS Windows 3.1.

WIDGET COMMANDS

The **tixExFileSelectDialog** command creates a new Tcl command whose name is the same as the path name of the ExFileSelectDialog's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ExFileSelectDialog widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ExFileSelectDialog widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixExFileSelectDialog** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixExFileSelectDialog** command.

pathName ***popdown***
Withdraws the ExFileSelectDialog from the screen.

pathName ***popup***
Pops up the ExFileSelectDialog on the screen.

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

KEYWORDS

file selector, widget, dialog

tixFileEntry − Create and manipulate tixFileEntry widgets

SYNOPSIS

tixFileEntry *pathName* ?*options*?

SUPER−CLASS

The **TixFileEntry** class is derived from the **TixLabelWidget** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The FileEntry widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−activatecmd***
Database Name: ***activateCmd***
Database Class: ***ActivateCmd***
Specifies the command to be called when the user activates the **button** subwidget. This command is called before the file dialog is popped up and can be used to customize the file dialog (which may be shared by several FileEnt widget).

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***
Specifies the command to be called when the **−value** option of the FileEntry is

changed. This usually happens when the user inputs a filename into the entry subwidget and hits the <Return> key. The command will be called with one arguments −− the new value of the FileEntry widget.

Command−Line Name: ***−dialogtype***
Database Name: ***dialogType***
Database Class: ***DialogType***
Specifies which type of file selection dialog should be popped up when the user invokes the **button** subwidget. Current only two values are valid: **tixFileSelectDialog** or **tixExFileSelectDialog**.

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***
Database Class: ***DisableCallback***
A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the FileEntry widget changes.

Command−Line Name: ***−disableforeground***
Database Name: ***disableForeground***
Database Class: ***DisableForeground***
The foreground color to use for of the entry subwidget when the FileEntry widget is disabled.

Command−Line Name: ***−filebitmap***
Database Name: ***fileBitmap***
Database Class: ***FileBitmap***
Specifies the bitmap to display in side the **button** subwidget.

Command−Line Name: ***−label***
Database Name: ***label***
Database Class: ***Label***
Specifies the string to display as the label of this FileEntry widget.

Command−Line Name: ***−labelside***
Database Name: ***labelSide***
Database Class: ***LabelSide***
Specifies where the label should be displayed relative to the entry subwidget. Valid options are: **top**, **left**, **right**, **bottom**, **none** or **acrosstop**.

Command−Line Name: ***−selectmode***
Database Name: ***selectMode***
Database Class: ***SelectMode***
Specifies how the FileEntry widget should react to `<KeyPress>` events. When set to "immediate", any user keyboard inputs will immediately change the **−value** option. When set to "normal", the user keyboard inputs will be copied to the **−value** option only if the `<Return>` key is pressed or the keyboard focus is changed. The use of the immediate mode is discouraged. For effective use of the FileEntry widget, one should use the normal mode together with the **update** widget command (see below).

Command−Line Name: ***−state***
Database Name: ***state***

Database Class: ***State***
Specifies the whether the FileEntry widget is normal or disabled. Only the values "normal" and "disabled" are recognized.

Command−Line Name: ***−validatecmd***
Database Name: ***validateCmd***
Database Class: ***ValidateCmd***
Specifies a TCL command to be called when the −value of the FileEntry widget is about to change. This command is called with one parameter −− the new **−value** entered by the user. This command is to validate this new value by returning a value it deems valid.

Command−Line Name: ***−value***
Database Name: ***value***
Database Class: ***Value***
Specifies the value of the FileEntry.

Command−Line Name: ***−variable***
Database Name: ***variable***
Database Class: ***Variable***
Specifies the global variable in which the value of the FileEntry should be stored. The value of the FileEntry will be automatically updated when this variable is changed.

SUBWIDGETS

Name: **button**
Class: **Button**

The button subwidget next to the entry subwidget.

Name: **entry**
Class: **Entry**

The entry subwidget in which the user can type in a filename.

DESCRIPTION

The **tixFileEntry** command creates a new window (given by the *pathName* argument) and makes it into a FileEntry widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the FileEntry such as its cursor and relief.

The FileEntry widget can be used to input a filename. The user can type in the filename manually. Alternatively, the user can press the button widget that sits next to the entry, which will bring up a file selection dialog of the type specified by the **−dialogtype** option.

WIDGET COMMANDS

The **tixFileEntry** command creates a new Tcl command whose name is the same as the path name of the FileEntry's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the FileEntry widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for FileEntry widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixFileEntry** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixFileEntry** command.

pathName ***invoke***

Forces the FileEntry widget to act as if the user has pressed the <return> key inside the entry subwidget.

pathName ***filedialog*** *?args?*

When no additional arguments are given, this command returns the pathname of the file dialog box associated with this FileEnt widget. When additional arguments are given, the widget command of the file dialog will be called with these arguments.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

pathName ***update***

If the user has modified the entry using keyboard inputs, the update command will **update** the **−value** of this FileEntry widget. When the FileEntry widget's **−selectmode** option is set to "normal", one should call the **update** command on this widget before examining its **−value** option. This command has no effect in if the **−selectmode** option is set to "immediate".

KEYWORDS

file entry, widget

tixFileSelectBox − Create and manipulate Tix FileSelectBox widgets

SYNOPSIS

tixFileSelectBox *pathName* ?*options*?

STANDARD OPTIONS

The FileSelectBox widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browsecmd***
Database Class: ***browseCmd***

Specifies the command to execute when the user browses through the files. By default, if the **−browsecmd** is specified, the browse command will be executed when the user clicks on a filename in the *Files* listbox.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to execute when the FileSelectBox is invoked. This command is executed with one parameter : the filename selected by the user.

Command−Line Name: ***−directory or −dir***
Database Name: ***directory***
Database Class: ***Directory***

Specifies the directory to look for files. By default this will be the current working directory of the program and will be changed as the user browses through the directories.

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***
Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the ExFileSelectBox widget changes.

Command−Line Name: ***−pattern***
Database Name: ***pattern***
Database Class: ***Pattern***

Specifies the matching pattern of the file names that should be listed in the *Files* listbox. For example "*.c" matches all the filenames that end with ".c". If this option is set to the empty string, the default pattern "*" will be used.

Command−Line Name: ***−value or −selection***

Database Name: ***value***
Database Class: ***Value***
Specifies the name of the filename currently selected by the user.

SUBWIDGETS

Name: **dirlist**
Class: **TixScrolledListBox**

The scrolled listbox that shows the directories.

Name: **filelist**
Class: **TixScrolledListBox**

The scrolled listbox that shows the files.

Name: **filter**
Class: **TixComboBox**

The ComboBox listbox that shows the filter string.

Name: **selection**
Class: **TixComboBox**

The ComboBox listbox that shows the file selection.

DESCRIPTION

The **tixFileSelectBox** command creates a new window (given by the *pathName* argument) and makes it into a FileSelectBox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the FileSelectBox such as its cursor and relief.

The FileSelectBox is similar to the standard Motif(TM) file−selection box. It is generally used for the user to choose a file. FileSelectBox stores the files mostly recently selected into a ComboBox widget so that they can be quickly selected again. The **tixFileSelectDialog** widget is a combination of the FileSelectBox widget and a dialog widget.

WIDGET COMMAND

The **tixFileSelectBox** command creates a new Tcl command whose name is the same as the path name of the FileSelectBox's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the FileSelectBox widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for FileSelectBox widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixFileSelectBox** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixFileSelectBox** command.

pathName ***filter***

Updates the files listed in the FileSelectBox according to the filtering pattern sepcified in the **filter** subwidget.

pathName ***invoke***

Execute the command specified by the **−command** option with the filename stored in the **selection** subwidget.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

DEFAULT BINDINGS

TIX automatically creates class bindings for FileSelectBoxes that give them the following default behavior:

[1]

Mouse button 1 in the *Directory* listbox will change the filter string to the selected directory.

[2]

Mouse button 1 in the *Files* listbox will change the filename that appears in the *Selection* entry. It will also trigger the **−browsecmd** if the option has been specified.

[3]

The current directory will be changed by (1) double clicking the *Directory* listbox or (2) invoking the *Filter* ComboBox. Please refer to the man page of **tixComboBox** for the default bindings of the ComboBoxes and how they can be invoked.

[4]

The command specified by the option −command will be invoked by (1) double clicking the *Files* listbox or (2) invoking *Selection* ComboBox.

KEYWORDS

file selector, widget

tixFileSelectDialog − Create and manipulate tixFileSelectDialog widgets

SYNOPSIS

tixFileSelectDialog *pathName* ?*options*?

SUPER−CLASS

The **TixFileSelectDialog** class does not have a super−class.

STANDARD OPTIONS

TixFileSelectDialog supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the user chooses on a filename (usually by selecting the filename and clicking on the "OK" button"). The command is called with one argument, the complete pathname of the file.

SUBWIDGETS

Name: **btns**
Class: **TixStdButtonBox**

The StdButtonBox subwidget at the bottom of FileSelectDialog. It contains the "OK", "Filter, "Cancel" and "Help" buttons.

Name: **fsbox**
Class: **TixFileSelectBox**

The FileSelectBox subwidget at the top of the FileSelectDialog.

DESCRIPTION

The **tixFileSelectDialog** command creates a new window (given by the *pathName* argument) and makes it into a FileSelectDialog widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the FileSelectDialog such as its cursor and relief.

The FileSelectDialog widget provides an convenient method for the user to select files. The FileSelectBox is similar to the standard Motif(TM) file−selection box.

WIDGET COMMANDS

The **tixFileSelectDialog** command creates a new Tcl command whose name is the same as the path name of the FileSelectDialog's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the FileSelectDialog widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for FileSelectDialog widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixFileSelectDialog** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixFileSelectDialog** command.

pathName ***popdown***

Withdraws the FileSelectDialog from the screen.

pathName ***popup***

Pops up the FileSelectDialog on the screen.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

KEYWORDS

file selector, widget, dialog

tixLabelEntry − Create and manipulate tixLabelEntry widgets

SYNOPSIS

tixLabelEntry *pathName* ?*options*?

SUPER−CLASS

The **TixLabelEntry** class is derived from the **TixLabelWidget** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The LabelEntry widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−disableforeground***
Database Name: ***disableForeground***
Database Class: ***DisableForeground***

The foreground color to use for of the entry subwidget when the LabelEntry widget is disabled.

Command−Line Name: ***−label***
Database Name: ***label***
Database Class: ***Label***

Specifies the string to display as the label of this LabelEntry widget.

Command−Line Name: ***−labelside***
Database Name: ***labelSide***
Database Class: ***LabelSide***

Specifies where the label should be displayed relative to the entry subwidget. Valid options are: **top**, **left**, **right**, **bottom**, **none** or **acrosstop**.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***

Specifies the whether the LabelEntry widget is normal or disabled. Only the values "normal" and "disabled" are recognized.

SUBWIDGETS

Name: **label**
Class: **Label**

The label subwidget.

Name: **entry**
Class: **Entry**

The entry subwidget.

DESCRIPTION

The **tixLabelEntry** command creates a new window (given by the *pathName* argument) and makes it into a LabelEntry widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the LabelEntry such as its cursor and relief.

The LabelEntry widget packages an entry widget and a label into one mega widget. It can be used be used to simplify the creation of "entry−form" type of interface. In this kind of interface, one must create many entry widgets with label widgets next to them and describe the use of each of the entry widgets.

WIDGET COMMANDS

The **tixLabelEntry** command creates a new Tcl command whose name is the same as the path name of the LabelEntry's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the LabelEntry widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for LabelEntry widgets:

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixLabelEntry** command.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixLabelEntry** command.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

KEYWORDS

label entry, widget

tixLabelFrame − Create and manipulate tixLabelFrame widgets

SYNOPSIS

tixLabelFrame *pathName* ?*options*?

SUPER−CLASS

The **TixLabelFrame** class is derived from the **TixLabelWidget** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The LabelFrame widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−label***
Database Name: ***label***
Database Class: ***Label***
Specifies the string to display as the label of this LabelFrame widget.

Command−Line Name: ***−labelside***
Database Name: ***labelSide***
Database Class: ***LabelSide***
Specifies where the label should be displayed relative to the entry subwidget. Valid options are: **top**, **left**, **right**, **bottom**, **none** or **acrosstop**.

Command−Line Name: ***−padx***
Database Name: ***padX***
Database Class: ***Pad***
Specifies the amount of the horizontal padding around the **frame** subwidget. Must be a valid non−negative integer number.

Command−Line Name: ***−pady***
Database Name: ***padY***
Database Class: ***Pad***
Specifies the amount of the vertical padding around the **frame** subwidget.

SUBWIDGETS

Name: **frame**
Class: **Frame**

The frame subwidget.

Name: **label**
Class: **Label**

The label subwidget.

DESCRIPTION

The **tixLabelFrame** command creates a new window (given by the *pathName* argument) and makes it into a LabelFrame widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the LabelFrame such as its cursor and relief.

CREATING WIDGETS INSIDE A LABELFRAME

The LabelFrame widget packages a frame widget and a label into one mega widget. To create widgets inside a LabelFrame widget, one must create the new widgets relative to the **frame** subwidget and manage them inside the **frame** subwidget. An error will be generated if one tries to create widgets as immediate children of the LabelFrame. For example: the following is correct code, which creates new widgets inside the frame subwidget:

```
tixLabelFrame .f
set f [.f subwidget frame]
button $f.b −text hi
pack $f.b
```

The following example code is *incorrect* because it tries to create immediate children of the LabelFrame **.f**:

```
tixLabelFrame .f
# wrong!
button .f.b −text hi
pack .f.b
```

WIDGET COMMANDS

The **tixLabelFrame** command creates a new Tcl command whose name is the same as the path name of the LabelFrame's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the LabelFrame widget's path

name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for LabelFrame widgets:

pathName ***cget*** *option*
: Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixLabelFrame** command.

pathName ***configure*** *?option? ?value option value ...?*
: Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixLabelFrame** command.

pathName ***subwidget*** *name ?args?*
: When no options are given, this command returns the pathname of the subwidget of the specified name.

When options are given, the widget command of the specified subwidget will be called with these options.

KEYWORDS

label frame, widget

tixListNoteBook − Create and manipulate tixListNoteBook widgets

SYNOPSIS

tixListNoteBook *pathName* ?*options*?

STANDARD OPTIONS

The ListNoteBook widget supports all the standard options of a frame widget. See the options(n) manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−dynamicgeometry***
Database Name: ***dynamicGeometry***
Database Class: ***DynamicGeometry***

If set to false, the size of the ListNotebook will match the size of the largest page. If set to true, the size of the ListNotebook will match the size of the current page (therefore, the size may change when the user selects different pages). The default value is false. A setting of true is discouraged.

Command−Line Name: ***−ipadx***
Database Name: ***ipadX***
Database Class: ***Pad***

The amount of internal horizontal paddings around the sides of the page subwidgets.

Command−Line Name: ***−ipady***
Database Name: ***ipadY***
Database Class: ***Pad***

The amount of internal vertical paddings around the sides of the page subwidgets.

SUBWIDGETS

Name: **hlist**
Class: **TixHList**

The HList widget that displays the names of the pages.

In addition, all the page subwidgets created as a result of the **add** command can be accessed by the **subwidget** command. They are identified by the **pageName** parameter to the **add** command.

DESCRIPTION

The **tixListNoteBook** command creates a new window (given by the *pathName* argument) and makes it into a ListNoteBook widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ListNoteBook widget such as its cursor and relief. The ListNoteBook widget is very similar to the TixNoteBook widget: it can be used to display many windows in a limited space using a "notebook" metaphore. The notebook is divided into a stack of pages (windows). At one time only one of these pages can be shown. The user can navigate through these pages by choosing the name of the desired page in the **hlist** subwidget.

WIDGET COMMANDS

The **tixListNoteBook** command creates a new Tcl command whose name is the same as the path name of the ListNoteBook widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ListNoteBook widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ListNoteBook widgets:

pathName ***add*** *pageName ?option value ...?*

Adds a new ListNotebook page subwidget into the ListNoteBook widget. *pageName* must be the name of an existing entry of the **hlist** subwidget. You must create the entry before calling the **add** command. Please refer to the **tixHList** manual entry for adding entries in an HList widget. Additional parameters may be supplied to configure this page subwidget. Possible options are:

−createcmd

Specifies a TCL command to be called the first time a page is shown on the screen. This option can be used to delay the creation of the contents of a page until necessary. Therefore, it can be used to speed up interface creation process especially when there are a large number of pages in a ListNoteBook widget.

−raisecmd

Specifies a TCL command to be called whenever this page is raised by the user.

When successful, this command returns the pathname of the newly created page.

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*.*Option* may have any of the values accepted by the **tixListNoteBook** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixListNoteBook** command.

pathName ***delete*** *pageName?*

Deletes the page identified by *pageName*.

pathName ***pagecget*** *pageName option*

Returns the current value of the configuration option given by *option* in the page given by *pageName*. *Option* may have any of the values accepted by the **add** widget command.

pathName ***pageconfigure*** *pageName ?option? ?value ...?*

When no option is given, prints out the values of all options of this page. If *option* is specified with no *value*, then the command returns the current value of that option. If one or more *option−value* pairs are specified, then the command modifies the given page's option(s) to have the given value(s); in this case the command returns an empty string. *Option* may be any of options accepted by the **add** widget command.

pathName ***pages***

Returns a list of the names of all the pages.

pathName ***raise*** *pageName*
Raise the page identified by *pageName*.

pathName ***raised***
Returns the name of the currently raised page.

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

EXAMPLE

```
set n [tixListNoteBook .n]; pack $n
$n subwidget hlist add page1 −text "Page 1"
$n subwidget hlist add page2 −text "Page 2"

set page1 [$n add page1]
set page2 [$n add page2]

button $page1.b −text "On page1"
button $page2.b −text "On page2"

pack $page1.b
pack $page2.b

$n raise page2
```

BINDINGS

When the user activates an entry in the **hlist** subwidget, the page associated with that entry will be raised to the front. This can be done by using the mouse or keyboard. The *hlist* subwidget operates with its **−selectmode** option set to single. See the event bindings of the HList widget for more details.

KEYWORDS

notebook, widget

tixMeter − Create and manipulate Tix Meter widgets

SYNOPSIS

tixMeter *pathName* ?*options*?

SUPER−CLASS

None.

STANDARD OPTIONS

The Meter widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−fillcolor***
Database Name: ***fillColor***
Database Class: ***FillColor***
The color of the progress bar.

Command−Line Name: ***−text***
Database Name: ***text***
Database Class: ***Text***
The text string to place inside the progress bar. If not specified, then the text string will be the percentage value specified by the **−value** option.

Command−Line Name: ***−value***
Database Name: ***value***
Database Class: ***Value***
A real value that specifies the progress. Must be between 0.0 to 1.0.

DESCRIPTION

The **tixMeter** command creates a new window (given by the *pathName* argument) and makes it into a Meter widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the Meter widget such as its cursor and relief. The Meter widget can be used to show the pregress of a background job which may take a long time to execute.

WIDGET COMMANDS

The **tixMeter** command creates a new Tcl command whose name is the same as the path name of the Meter widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the Meter widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for Meter widgets:

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixMeter** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixMeter** command.

BINDINGS

There is no bindings for the Meter widget.

KEYWORDS

meter, progress, widget

tixNoteBook − Create and manipulate tixNoteBook widgets

SYNOPSIS

tixNoteBook *pathName* ?*options*?

STANDARD OPTIONS

The NoteBook widget supports all the standard options of a frame widget. See the options(n) manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−dynamicgeometry***
Database Name: ***dynamicGeometry***
Database Class: ***DynamicGeometry***

If set to false, the size of the Notebook will match the size of the largest page. If set to true, the size of the Notebook will match the size of the current page (therefore, the size may change when the user selects different pages). The default value is false. A setting of true is discouraged.

Command−Line Name: ***−ipadx***
Database Name: ***ipadX***
Database Class: ***Pad***

The amount of internal horizontal paddings around the sides of the page subwidgets.

Command−Line Name: ***−ipady***
Database Name: ***ipadY***
Database Class: ***Pad***
The amount of internal vertical paddings around the sides of the page subwidgets.

SUBWIDGETS

Name: **nbframe**
Class: **tixNoteBookFrame**

The "note book frame" widget that displays ths tabs of the notebook. Most of the display options of the page tabs are controlled by this subwidget. For example, if you need to choose a different font to display the tab names of the pages, the color of the inactive tabs or the color behind the tabs, you can configure the options of the **nbframe** subwidget. See the manual page of **tixNoteBookFrame** for more details.

In addition, all the page subwidgets created as a result of the **add** command can be accessed by the **subwidget** command. They are identified by the **pageName** parameter to the **add** command.

DESCRIPTION

The **tixNoteBook** command creates a new window (given by the *pathName* argument) and makes it into a NoteBook widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the NoteBook widget such as its cursor and relief. The NoteBook widget can be used to display many windows in a limited space using a "notebook" metaphore. The notebook is divided into a stack of pages (windows). At one time only one of these pages can be shown. The user can navigate through these pages by choosing the visual "tabs" at the top of the NoteBook widget.

WIDGET COMMANDS

The **tixNoteBook** command creates a new Tcl command whose name is the same as the path name of the NoteBook widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the NoteBook widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for NoteBook widgets:

pathName ***add*** *pageName ?option value ...?*
Adds a new notebook page subwidget into the NoteBook widget. Additional parameters may be supplied to configure this page subwidget. Possible options are:

−anchor
Specifies how the information in a tab (e.g. text or a bitmap) is to be displayed in the widget. Must be one of the values **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, or **center**. For example, nw means display the information such that its top−left corner is at the top−left corner of the widget.

−bitmap

Specifies a bitmap to display on the tab of this page. The bitmap is displayed only if none of the **−label** or **−image** options are specified.

−createcmd
Specifies a TCL command to be called the first time a page is shown on the screen. This option can be used to delay the creation of the contents of a page until necessary. Therefore, it can be used to speed up interface creation process especially when there are a large number of pages in a NoteBook widget.

−image
Specifies an image to display on the tab of this page. The image is displayed only if the **−label** options is not specified.

−justify
When there are multiple lines of text displayed in a tab, this option determines how the lines line up with each other. Must be one of left, center, or right. **Left** means that the lines' left edges all line up, **center** means that the lines' centers are aligned, and **right** means that the lines' right edges line up.

−label
Specifies a text label string to display on the tab of this page subwidget.

−raisecmd
Specifies a TCL command to be called whenever this page is raised by the user.

−state
Specifies whether this page can be raised by the user. Must be either **normal** or **disabled**.

−underline
Specifies the integer index of a character to underline in the tab. This option is used by the default bindings to implement keyboard traversal for menu buttons and menu entries. 0 corresponds to the first character of the text displayed in the widget, 1 to the next character, and so on.

−wraplength
This option specifies the maximum line length of the label string on this tab. If the line length of the label string exceeds this length, it is wrapped onto the next line, so that no line is longer than the specified length. The value may be specified in any of the standard forms for screen distances. If this value is less than or equal to 0 then no wrapping is done: lines will break only at newline characters in the text.

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*.*Option* may have any of the values accepted by the **tixNoteBook** command.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see

Tk_ConfigureInfo for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixNoteBook** command.

pathName ***delete*** *pageName?*
Deletes the page identified by *pageName*.

pathName ***pagecget*** *pageName option*
Returns the current value of the configuration option given by *option* in the page given by *pageName*. *Option* may have any of the values accepted by the **add** widget command.

pathName ***pageconfigure*** *pageName ?option? ?value ...?*
When no option is given, prints out the values of all options of this page. If *option* is specified with no *value*, then the command returns the current value of that option. If one or more *option−value* pairs are specified, then the command modifies the given page's option(s) to have the given value(s); in this case the command returns an empty string. *Option* may be any of options accepted by the **add** widget command.

pathName ***pages***
Returns a list of the names of all the pages.

pathName ***raise*** *pageName*
Raise the page identified by *pageName*.

pathName ***raised***
Returns the name of the currently raised page.

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

[1]
When the user pressed the left mouse button over a notebook tab, the notebook page associated with that tab will be raised to the top of the stack of pages.

[2]
The pages can also be selected using the keyboard. The user can type the **<Tab>** key to cycle among the set of pages. When the focus appears on the desired page, the user can type **<Return>** or **<space>** to select that page. Or, if the user wants to cancel the selection, he/she can type the **<Escape>** key.

KEYWORDS

notebook, widget

tixOptionMenu − Create and manipulate tixOptionMenu widgets

SYNOPSIS

tixOptionMenu *pathName* ?*options*?

SUPER−CLASS

The **TixOptionMenu** class is derived from the **TixLabelWidget** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The OptionMenu widget supports all the standard Tix widget options. See the **Tix−Options** manual entry for details on the standard Tix widget options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the command to be called when the **−value** option of the OptionMenu is changed. The command will be called with one arguments −− the new value of the OptionMenu widget.

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***
Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the OptionMenu widget changes.

Command−Line Name: ***−dynamicgeometry***
Database Name: ***dynamicGeometry***
Database Class: ***DynamicGeometry***

A boolean value indicating whether the size of the **menubutton** subwidget should change dynamically to match the width of the currently selected menu entry. If set to false (the default), the the size of the menubutton subwidget will be wide enough to display every menu entry fully and does not change when the user selects different

entries.

Command−Line Name: ***−label***
Database Name: ***label***
Database Class: ***Label***
Specifies the string to display as the label of this OptionMenu widget.

Command−Line Name: ***−labelside***
Database Name: ***labelSide***
Database Class: ***LabelSide***
Specifies where the label should be displayed relative to the entry subwidget. Valid options are: **top**, **left**, **right**, **bottom**, **none** or **acrosstop**.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***
Specifies the whether the OptionMenu widget is normal or disabled. Only the values "normal" and "disabled" are recognized.

Command−Line Name: ***−value***
Database Name: ***value***
Database Class: ***Value***
Specifies the value of the OptionMenu. The value of the OptionMenu widget is the name of the item currently displayed by its **menubutton** subwidget.

Command−Line Name: ***−variable***
Database Name: ***variable***
Database Class: ***Variable***
Specifies the global variable in which the value of the OptionMenu should be stored. The value of the OptionMenu will be automatically updated when this variable is changed.

SUBWIDGETS

Name: **menu**
Class: **Menu**

The menu subwidget, which is popped up when the user press the **menubutton** subwidget.

Name: **menubutton**
Class: **Menubutton**

The menubutton subwidget.

DESCRIPTION

The **tixOptionMenu** command creates a new window (given by the *pathName* argument) and makes it into a OptionMenu widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the OptionMenu such as its cursor and relief.

WIDGET COMMANDS

The **tixOptionMenu** command creates a new Tcl command whose name is the same as the path name of the OptionMenu's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the OptionMenu widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for OptionMenu widgets:

pathName ***add*** *type name ?option value ...?*

Adds a new item into the OptionMenu widget. *type* must be either **command** or **separator**. The *options* may be any of the valid options for the **command** or **separator** menu entry types for the TK **menu** widget class, except **−command**.

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixOptionMenu** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixOptionMenu** command.

pathName ***delete*** *name*

Deletes the menu entry identified by *name*.

pathName ***disable*** *name*

Disables the menu entry identified by *name*.

pathName ***enable*** *name*

Enables the menu entry identified by *name*.

pathName ***entrycget*** *name option*

Returns the current value of the configuration option given by *option* in the menu entry identified by *name*. *Option* may have any of the values accepted by the **add** widget command.

pathName ***entryconfigure*** *name ?option? ?value option value ...?*

Query or modify the configuration options of the menu entry identified by *name*. If no *option* is specified, returns a list describing all of the available options for the menu entry (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this

case the command returns an empty string. *Option* may have any of the values accepted by the **add** widget command.

pathName ***entries***
Returns the names of all the entries currently in the OptionMenu widget.

pathName ***subwidget*** *name ?args?*
When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

KEYWORDS

option menu, widget

tixPanedWindow − Create and manipulate tixPanedWindow widgets

SYNOPSIS

tixPanedWindow *pathName* ?*options*?

STANDARD OPTIONS

The PanedWindow widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Name: **command**
Class: **Command**
Switch: **−command**

Specifies the command to invoke when the panes change their sizes. This command is called with a list of integers that record the new sizes of the panes. The sizes of the panes are listed in the order of the panes' creation.

Command−Line Name: ***−dynamicgeometry***
Database Name: ***dynamicGeometry***
Database Class: ***DynamicGeometry***
If set to true, the size of the PanedWindow will dynamically change if the size of any of its panes changes. Otherwise, the size of the PanedWindow will only increase when size of any of its panes changes and will not decrease. The default value is true.

Command−Line Name: ***−handleactivebg***
Database Name: ***handleActiveBg***
Database Class: ***HandleActiveBg***
Specifies the active background color of the resize handles. When the mouse cursor enters a resize handle, the resize handle will adopt the active background color.

Command−Line Name: ***−handlebg***
Database Name: ***handleBg***
Database Class: ***Background***
Specifies the normal background color of the resize handles.

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***
Specifies the desired height for the window.

Command−Line Name: ***−orientation***
Database Name: ***orientation***
Database Class: ***Orientation***
Specifies the orientation of the panes. Must be either **vertical** or **horizontal**.

Command−Line Name: ***−paneborderwidth or −panebd***
Database Name: ***paneBorderWidth***
Database Class: ***PaneBorderWidth***
Specifies the border width of the panes.

Command−Line Name: ***−panerelief***
Database Name: ***paneRelief***
Database Class: ***PaneRelief***
Specifies the border relief of the panes.

Command−Line Name: ***−separatoractivebg***
Database Name: ***separatorActiveBg***
Database Class: ***SeparatorActiveBg***
Specifies the active background color of the separators. When the user grabs a resize handle, the separators will adopt the active background color.

Command−Line Name: ***−separatorbg***
Database Name: ***separatorBg***
Database Class: ***Background***
Specifies the normal background color of the separators.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window.

SUBWIDGETS

All the pane subwidgets created as a result of the **add** command can be accessed by the **subwidget** command. They are identified by the **paneName** parameter to the **add** command.

DESCRIPTION

The **tixPanedWindow** command creates a new window (given by the *pathName* argument) and makes it into a PanedWindow widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the PanedWindow widget such as its cursor and relief.

The PanedWindow widget allows the user to interactively manipulate the sizes of several panes. The panes can be arranged either vertically or horizontally. Each individual pane may have upper and lower limits of its size. The user changes the sizes of the panes by dragging the resize handle between two panes.

WIDGET COMMAND

The **tixPanedWindow** command creates a new Tcl command whose name is the same as the path name of the PanedWindow widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the frame widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for PanedWindow widgets:

pathName ***add*** *paneName ?option value ...?*

Adds a new pane subwidget with the name *paneName* into the PanedWindow widget. Additional configuration options can be given to configure the new button subwidget. Three configuration options are supported:

−after *pane*

Specifies that the new pane should be placed after *pane* in the list of panes in this PanedWindow widget.

−at *integer*

Specifies the position of the new pane in the list of panes in this PanedWindow widget. **0** means the first position, **1** means the second, and so on. In addition, **end** means the end of the list.

−before *pane*

Specifies that the new pane should be placed before *pane* in the list of panes in this PanedWindow widget.

−expand *factor*

Specifies the **expand/shrink factor** of this pane. *Factor* must be a non−negative floating point number. The default value is 0.0. The expand/shrink factor is used to calculate how much each pane should grow or shrink when the size of the PanedWindow main window is changed. When the main window expands/shrinks by *n* pixels, then pane *i* will grow/shrink by about *n * factor(i) / summation(factors)*, where *factor(i)* is the expand/shrink factor of pane i and *summation(factors)* is the summation of the expand/shrink factors of all the panes. If *summation(factors)* is 0.0, however, only the last visible pane will be grown or shrunk.

−min *integer*
Specifies the minimum size, in pixels, of the new pane; the default is 0.

−max *integer*
Specifies the maximum size, in pixels, of the new pane; the default is 10000.

−size *integer*
Specifies the size, in pixels, of the new pane; if the **−size** option is not given, or set to the empty string, the PanedWindow widget will use the natural size of the pane subwidget.

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may be **−min**, **−max** and/or **−size**, or any option accepted by the Tk frame widget.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may be any of the non−static options of the PanedWindow widget.

pathName ***delete*** *paneName*
Removes the pane given by *paneName* and deletes its contents.

pathName ***forget*** *paneName*
Removes the pane given by *paneName* but does not delete its contents. This pane can be later added back to the PanedWindow widget by the **manage** method.

pathName ***manage*** *paneName ?option value ...?*
Adds the pane given by *paneName* back to the PanedWindow widget. *PaneName* must be already forgotten by the **forget** method. Additional *option−value* pairs, same as those accepted by the **add** method, can be given to control the appearance and position of the pane.

pathName ***panecget*** *paneName option*
Returns the current value of the configuration option given by *option* in the pane given by *paneName*. *Option* may have any of the values accepted by the **add** widget command.

pathName ***paneconfigure*** *paneName ?option? ?value ...?*
When no option is given, prints out the values of all options of this pane. If *option* is specified with no *value*, then the command returns the current value of that option. If one or more *option−value* pairs are specified, then the command modifies the given pane's option(s) to have the given value(s); in this case the command returns an empty string. *Option* may be **−min**, **−max** and/or **−size**, or any option accepted by the Tk frame widget. The sizes of the panes may be changed as a result of calling the **paneconfigure** command.

pathName ***panes***
Returns a list of the names of all panes.

pathName ***setsize*** *paneName newSize ?direction?*
Sets the size of the pane specified by *paneName* to *newSize*. The *direction* parameter specifies in which direction the pane should grow/shrink. Possible values are **next**: the pane will grow or shrink by moving the boundary between itself and the pane to its right or bottom; **prev**: the pane will grow or shrink by moving the boundary between itself and the pane to its left or top.

pathName ***subwidget*** *name ?args?*
When no options are given, returns the pathname of the subwidget of the specified name.

When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

The panes' sizes will be changed when the user drags the handles. The change in the panes' sizes may be subjected to the **−min**, **−max** and **−size** options of the panes.

KEYWORDS

TIX, Container Widget

tixPopupMenu − Create and manipulate tixPopupMenu widgets

SYNOPSIS

tixPopupMenu *pathName* ?*options*?

SUPER−CLASS

The **tixPopupMenu** class is derived from the **TixShell** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The PopupMenu widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−buttons***
Database Name: ***buttons***
Database Class: ***Buttons***

A Tcl list that specifies the mouse button(s) and key modifier(s) that bring up the popup menu. Each element of this list is in turn a list that contains two elements: the first element is an integer that indicates the mouse button that brings up the popup menu; the second element specifies the key modifiers that should be used in conjunction with the mouse button. For example, the value **{{1 {Control Meta}} {3 {Any}}}** specifies that the popup menu can be popped up by (a) pressing mouse button 1 with either the Control or the Meta key or (b) pressing mouse button 3 with any key modifier. The default value is **{{3 {Any}}}**: only mouse button 3 brings up the popup menu.

Command−Line Name: ***−postcmd***
Database Name: ***postCmd***
Database Class: ***PostCmd***

Specifies a command to be evaluated just before the menu is about to pop−up. This command is called with two default arguments: the root x−y coordinates where the user has pressed the mouse button. This command must return a boolean value: a false indicates that the menu shouldn't be popped up at this point; a true indicates that the menu should be popped up. This option can be used to find out where the user has pressed the mouse−button and optionally disable the popup menu over certain screen areas.

Command−Line Name: ***−spring***
Database Name: ***spring***
Database Class: ***Spring***

When set to **true**, the menu will be automatically popped down if the user releases the mouse button outside of the menu and no menu commands will be invoked. This makes it easy for the user to cancel the popup menu without pressing the Escape key. The default value is **true**.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***

Must be either **disabled** or **normal**. The PopupMenu widget will not pop up unless its **−state** is set to **normal**.

Command−Line Name: ***−title***
Database Name: ***title***
Database Class: ***Title***

Specifies a text string to display inside the **menubutton** subwidget, as the title of this PopupMenu.

SUBWIDGETS

Name: **menu**
Class: **Menu**

The menu subwidget.

Name: **menubutton**
Class: **Menubutton**

The menubutton subwidget.

DESCRIPTION

The **tixPopupMenu** command creates a new window (given by the *pathName* argument) and makes it into a PopupMenu widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the PopupMenu widget such as its cursor and relief. The Tix PopupMenu widget can be used as a replacement of the **tk_popup** command. The advantage of the Tix PopupMenu widget is it requires less application code to manipulate. Also, it provides a title for the popup menu, which is not available from **tk_popup**.

WIDGET COMMANDS

The **tixPopupMenu** command creates a new Tcl command whose name is the same as the path name of the PopupMenu widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the PopupMenu widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for PopupMenu widgets:

pathName ***bind*** *widget ?widget ...?*
Binds this PopupMenu to one or more *widgets*. The PopupMenu will be activated when the user presses the right mouse button over these *widgets*.

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixPopupMenu** command.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixPopupMenu** command.

pathName ***post*** *widget x y*
Posts the PopupMenu inside the *widget* at the coordinate *x,y*.

pathName ***unbind*** *widget ?widget ...?*
Cancels the PopupMenu's binding with the *widget(s)*.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

KEYWORDS

popup menu, widget

tixScrolledHList − Create and manipulate Tix ScrolledHList widgets

SYNOPSIS

tixScrolledHList *pathName* ?*options*?

STANDARD OPTIONS

−anchor, anchor, Anchor
−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−relief, relief, Relief

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***
Specifies the desired height for the window, in pixels.

Command−Line Name: ***−scrollbar***
Database Name: ***scrollbar***
Database Class: ***Scrollbar***
Specifies the display policy of the scrollbars. The following values are recognized:

auto *?+x? ?−x? ?+y? ?−y?*
When **−scrollbar** is set to "**auto**", the scrollbars are shown only when needed. Additional modifiers can be used to force a scrollbar to be shown or hidden. For example, "**auto −y**" means the horizontal scrollbar should be shown when needed but the vertical scrollbar should always be hidden; "**auto +x**" means the vertical scrollbar should be shown when needed but the horizontal scrollbar should always be shown, and so on.

both

Both scrollbars are shown

none
The scrollbars are never shown.

x
Only the horizontal scrollbar is shown;

y
Only the vertical scrollbar is shown.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window, in pixels.

SUBWIDGETS

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **hlist**
Class: **Hlist**

The tixHList subwidget inside the ScrolledHList widget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixScrolledHList** command creates a new window (given by the *pathName* argument) and makes it into a ScrolledHList widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ScrolledHList widget such as its cursor and relief.

WIDGET COMMANDS

The **tixScrolledHList** command creates a new Tcl command whose name is the same as the path name of the ScrolledHList widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ScrolledHList widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ScrolledHList widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixScrolledHList** command.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixScrolledHList** command.

pathName ***subwidget*** *name ?args?*
When no additional arguments are given, returns the pathname of the subwidget of the specified name. When no additional arguments are given, the widget command of the specified subwidget will be called with these parameters.

KEYWORDS

scroll, hierarchical listbox, widget

tixScrolledListBox − Create and manipulate Tix ScrolledListBox widgets

SYNOPSIS

tixScrolledListBox *pathName* ?*options*?

STANDARD OPTIONS

−anchor, anchor, Anchor
−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−relief, relief, Relief

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−anchor***
Database Name: ***anchor***
Database Class: ***Anchor***

Specifies the alignment of the items inside the listbox subwidget. Only the values **w** and **e** are allowed. When set to **w**, the listbox is automatically aligned to the beginning of the items. When set to **e**, the listbox is automatically aligned to the end of the items. Automatically alignment only happens when the ScrolledListBox widget changes its size.

Command−Line Name: ***−browsecmd***
Database Name: ***browsecmd***
Database Class: ***BrowseCmd***
Specifies the command to be called when the user browses the elements inside the **listbox** subwidget (see the BINDINGS section below).

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***
Specifies the command to be called when the user invokes the **listbox** subwidget (see the BINDINGS section below).

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***
Specifies the desired height for the window, in pixels.

Command−Line Name: ***−scrollbar***
Database Name: ***scrollbar***
Database Class: ***Scrollbar***
Specifies the display policy of the scrollbars. The following values are recognized:

auto *?+x? ?−x? ?+y? ?−y?*
When **−scrollbar** is set to "**auto**", the scrollbars are shown only when needed. Additional modifiers can be used to force a scrollbar to be shown or hidden. For example, "**auto −y**" means the horizontal scrollbar should be shown when needed but the vertical scrollbar should always be hidden; "**auto +x**" means the vertical scrollbar should be shown when needed but the horizontal scrollbar should always be shown, and so on.

both
Both scrollbars are shown

none
The scrollbars are never shown.

x
Only the horizontal scrollbar is shown;

y
Only the vertical scrollbar is shown.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window, in pixels.

SUBWIDGETS

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **listbox**
Class: **Listbox**

The listbox subwidget inside the ScrolledListBox widget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixScrolledListBox** command creates a new window (given by the *pathName* argument) and makes it into a ScrolledListBox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ScrolledListBox widget such as its cursor and relief.

WIDGET COMMANDS

The **tixScrolledListBox** command creates a new Tcl command whose name is the same as the path name of the ScrolledListBox widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ScrolledListBox widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ScrolledListBox widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixScrolledListBox** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixScrolledListBox** command.

pathName ***subwidget*** *name ?args?*

When no additional arguments are given, returns the pathname of the subwidget of the specified name. When no additional arguments are given, the widget command of the specified subwidget will be called with these parameters.

BINDINGS

[1]

If the **−browsecmd** option is set, the command which it referes to is called whenever a <ButtonPress−1> or a <Motion−1> event occurrs inside the **listbox** subwidget.

[2]

The command specified by the **−command** option is invoked when a <Double−1> event occurrs inside the **listbox** subwidget.

BUGS

The capitalization of some of the commands names in Tix 3.x has been changed in Tix 4.0. All commands that ended with **box** have been changed to a capitalized **Box**. Hence, the command **tixScrolledListbox** in Tix 3.x has been changed to **tixScrolledListBox** in Tix 4.0

KEYWORDS

scroll, listbox, widget

tixScrolledText − Create and manipulate Tix ScrolledText widgets

SYNOPSIS

tixScrolledText *pathName* ?*options*?

STANDARD OPTIONS

−anchor, anchor, Anchor
−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−relief, relief, Relief

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***

Specifies the desired height for the window, in pixels.

Command−Line Name: ***−scrollbar***
Database Name: ***scrollbar***
Database Class: ***Scrollbar***
Specifies the display policy of the scrollbars. The following values are recognized:

auto *?+x? ?−x? ?+y? ?−y?*
When **−scrollbar** is set to "**auto**", the scrollbars are shown only when needed. Additional modifiers can be used to force a scrollbar to be shown or hidden. For example, "**auto −y**" means the horizontal scrollbar should be shown when needed but the vertical scrollbar should always be hidden; "**auto +x**" means the vertical scrollbar should be shown when needed but the horizontal scrollbar should always be shown, and so on.

both
Both scrollbars are shown

none
The scrollbars are never shown.

x
Only the horizontal scrollbar is shown;

y
Only the vertical scrollbar is shown.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window, in pixels.

SUBWIDGETS

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **text**
Class: **Text**

The Text subwidget inside the ScrolledText widget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixScrolledText** command creates a new window (given by the *pathName* argument) and makes it into a ScrolledText widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ScrolledText widget such as its cursor and relief.

WIDGET COMMANDS

The **tixScrolledText** command creates a new Tcl command whose name is the same as the path name of the ScrolledText widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ScrolledText widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ScrolledText widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixScrolledText** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixScrolledText** command.

pathName ***subwidget*** *name ?args?*

When no additional arguments are given, returns the pathname of the subwidget of the specified name. When no additional arguments are given, the widget command of the specified subwidget will be called with these parameters.

KEYWORDS

scroll, text, widget

tixScrolledWindow − Create and manipulate Tix ScrolledWindow widgets

SYNOPSIS

tixScrolledWindow *pathName* ?*options*?

STANDARD OPTIONS

−anchor, anchor, Anchor
−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−relief, relief, Relief

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***
Specifies the desired height for the window, in pixels.

Command−Line Name: ***−scrollbar***
Database Name: ***scrollbar***
Database Class: ***Scrollbar***
Specifies the display policy of the scrollbars. The following values are recognized:

auto *?+x? ?−x? ?+y? ?−y?*
When **−scrollbar** is set to "**auto**", the scrollbars are shown only when needed. Additional modifiers can be used to force a scrollbar to be shown or hidden. For example, "**auto −y**" means the horizontal scrollbar should be shown when needed but the vertical scrollbar should always be hidden; "**auto +x**" means the vertical scrollbar should be shown when needed but the horizontal scrollbar should always be shown, and so on.

both
Both scrollbars are shown

none
The scrollbars are never shown.

x
Only the horizontal scrollbar is shown;

y
Only the vertical scrollbar is shown.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window, in pixels.

Command−Line Name: ***−xscrollincrement***
Database Name: ***xScrollIncrement***
Database Class: ***ScrollIncrement***

Specifies by how much the window should be scrolled in the horizontal direction when the user presses the arrows in the horizontal scrollbar. In Pixels.

Command−Line Name: ***−yscrollincrement***
Database Name: ***yScrollIncrement***
Database Class: ***ScrollIncrement***

Specifies by how much the window should be scrolled in the vertical direction when the user presses the arrows in the horizontal scrollbar. In pixels.

SUBWIDGETS

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **window**
Class: **Frame**

The frame subwidget which is scrolled by the ScrolledWindow widget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixScrolledWindow** command creates a new window (given by the *pathName* argument) and makes it into a ScrolledWindow widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the ScrolledWindow widget such as its cursor and relief.

CREATING WIDGETS INSIDE A SCROLLEDWINDOW WIDGET

To create widgets inside a ScrolledWindow widget, one must create the new widgets relative to the **window** subwidget and manage them inside the **window** subwidget. An error will be generated if one tries to create widgets as immediate children of the ScrolledWindow. For example: the following is correct code, which creates new widgets inside the window subwidget:

```
tixScrolledWindow .w; pack .w
set f [.w subwidget window]
button $f.b −text hi −width 40 −height 40
pack $f.b
```

The following example code is *incorrect* because it tries to create immediate children of the ScrolledWindow **.w**:

```
tixScrolledWindow .w; pack .w
button .w.b −text hi −width 40 −height 40
pack .w.b
```

WIDGET COMMANDS

The **tixScrolledWindow** command creates a new Tcl command whose name is the same as the path name of the ScrolledWindow widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the ScrolledWindow widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for ScrolledWindow widgets:

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixScrolledWindow** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixScrolledWindow** command.

pathName ***subwidget*** *name ?args?*

When no additional arguments are given, returns the pathname of the subwidget of the specified name. When no additional arguments are given, the widget command of the specified subwidget will be called with these parameters.

KEYWORDS

scroll, window, frame, container, widget

tixSelect − Create and manipulate tixSelect widgets

SYNOPSIS

tixSelect *pathName* ?*options*?

SUPER−CLASS

The **TixSelect** class is derived from the **TixLabelWidget** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

The Select widget supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−allowzero***
Database Name: ***allowZero***
Database Class: ***AllowZero***

A boolean value that specifies whether the selection can be empty. When set to false, at least one button subwidget must be selected at any time. **Note**: When the Select widget is first constructed, the default selection is always empty, even if **−allowzero** is set to **false**.

Command−Line Name: ***−buttontype***
Database Name: ***buttonType***
Database Class: ***ButtonType***

The type of buttons to be used as subwidgets inside the Select widget. By default, the standard Tk **button** widget class is used.

Command−Line Name: ***−command***
Database Name: ***command***
Database Class: ***Command***

Specifies the TCL command to be executed when the **−value** of the Select widget is changed. This command will be invoked with two arguments. The first is the name of the button subwidget that has toggled. The second is a boolean value indicating whether the button subwidget is selected. This command is executed only when the **−disableCallback** option is set to false.

Command−Line Name: ***−disablecallback***
Database Name: ***disableCallback***
Database Class: ***DisableCallback***

A boolean value indicating whether callbacks should be disabled. When set to true, the TCL command specified by the **−command** option is not executed when the **−value** of the Select widget changes.

Command−Line Name: ***−orientation or −orient***
Database Name: ***orientation***
Database Class: ***Orientation***

Specifies the orientation of the button subwidgets. Only the values **horizontal** and **vertical** are recognized. This is a *static option* and it can only be assigned during the creation of the widget.

Command−Line Name: ***−label***
Database Name: ***label***
Database Class: ***Label***

Specifies the string to display as the label of this Select widget.

Command−Line Name: ***−labelside***
Database Name: ***labelSide***
Database Class: ***LabelSide***

Specifies where the label should be displayed relative to the Select widget. Valid options are: **top**, **left**, **right**, **bottom**, **none** or **acrosstop**.

Command−Line Name: ***−padx***
Database Name: ***padX***
Database Class: ***Pad***

Specifies the horizontal padding between two neighboring button subwidgets. This is a *static option* and it can only be assigned during the creation of the widget.

Command−Line Name: ***−padx***
Database Name: ***padY***
Database Class: ***Pad***

Specifies the vertical padding between two neighboring button subwidgets. This is a *static option* and it can only be assigned during the creation of the widget.

Command−Line Name: ***−radio***
Database Name: ***radio***
Database Class: ***Radio***

A boolean value that specifies whether the Select widget should act as a radio−box. When set to true, at most one button subwidget can be selected at any time. This is a *static option* and it can only be assigned during the creation of the widget.

Command−Line Name: ***−selectedbg***
Database Name: ***selectedBg***
Database Class: ***SelectedBg***

Specifies the background color of all the selected button subwidgets.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***

Specifies the state of all the buttons inside the Select widget. Only the values **normal** and **disabled** are recognized. When the state is set to **disabled**, all user actions on this Select widget are ignore.

Command−Line Name: ***−validatecmd***
Database Name: ***validateCmd***
Database Class: ***ValidateCmd***

Specifies a TCL command to be called when the −value of the Select widget is about to change. This command is called with one parameter −− the new **−value** entered by the user. This command is to validate this new value by returning a value it deems valid.

Command−Line Name: ***−value***
Database Name: ***value***
Database Class: ***Value***

The value of a Select widget is a list of the names of the button subwidgets that have been selected by the user. When you assign the value of a Select widget using the "config −value" widget command, the TCL command specified by the **−command** option will be invoked if some button subwidgets are toggled.

Command−Line Name: ***−variable***
Database Name: ***variable***
Database Class: ***Variable***

Specifies the global variable in which the value of the Select widget should be stored. The value of a Select widget is stored as a list of the names of the button subwidgets that have been selected by the user. The value of the Select widget will be automatically updated when this variable is changed.

SUBWIDGETS

Name: **label**
Class: **Label**

The label subwidget.

In addition, all the button subwidgets created as a result of the **add** widget command can be accessed by the **subwidget** command. They are identified by the *buttonName* parameter to the **add** widget command. Here is an example:

```
set s [tixSelect .s]
pack $s
$s add eat   -text Eat
$s add sleep -text Sleep
$s subwidget eat   config -fg green
$s subwidget sleep config -fg red
```

DESCRIPTION

The **tixSelect** command creates a new window (given by the *pathName* argument) and makes it into a Select widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the Select widget such as its cursor and relief. The Select widget is a container of button subwidgets. It can be used to provide radio−box or check−box style of selection options for the user.

WIDGET COMMANDS

The **tixSelect** command creates a new Tcl command whose name is the same as the path name of the Select widget's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the Select widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for Select widgets:

pathName ***add*** *buttonName ?option value ... ?*

Adds a new button subwidget with the name *buttonName* into the Select widget. Additional configuration options can be given to configure the new button subwidget.

pathName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixSelect** command.

pathName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixSelect** command.

pathName ***invoke*** *buttonName*
Invokes the button subwidget with the name *buttonName*.

pathName ***subwidget*** *name ?args?*
When no options are given, returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

When the user presses the left mouse button over the a button subwidget, it will be toggled and the **−value** option of the tixSelect widget will be changed.

EXAMPLE

The following example creates a radio−box style iconbar for the user to choose one value among **eat**, **work** or **sleep**.

```
set s [tixSelect .s −radio true −allowzero false]
$s add eat   −bitmap [tix getbitmap eat]
$s add work  −bitmap [tix getbitmap work]
$s add sleep −bitmap [tix getbitmap sleep]
```

KEYWORDS

choice, container, widget

tixStdButtonBox − Create and manipulate Tix StdButtonBox widgets

SYNOPSIS

tixStdButtonBox *pathName* ?*options*?

STANDARD OPTIONS

−anchor, anchor, Anchor
−background or −bg, background, Background
−borderWidth
−cursor, cursor, Cursor
−relief, relief, Relief

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−orientation***
Database Name: ***orientation***
Database Class: ***Orientation***

Static Option. Specifies the orientation of the button subwidgets. Only the values "horizontal" and "vertical" are recognized.

Command−Line Name: ***−padx***
Database Name: ***padx***
Database Class: ***Pad***

Specifies the horizontal padding between two neighboring button subwidgets in the StdButtonBox widget.

Command−Line Name: ***−padx***
Database Name: ***padx***
Database Class: ***Pad***

Specifies the vertical padding between two neighboring button subwidgets in the StdButtonBox widget.

Command−Line Name: ***−state***
Database Name: ***state***
Database Class: ***State***

Specifies the state of all the buttons inside the StdButtonBox widget. *Note*: Setting this option using the *config* widget command will enable or disable all the buttons subwidgets. Original states of the individual buttons are *not* saved.

SUBWIDGETS

Name: **ok**
Class: **Button**

The first button subwidget. By default it displays the text string "Ok"

Name: **apply**
Class: **Button**

The second button subwidget. By default it displays the text string "Apply"

Name: **cancel**
Class: **Button**

The third button subwidget. By default it displays the text string "Cancel"

Name: **help**
Class: **Button**

The fourth button subwidget. By default it displays the text string "Help"

DESCRIPTION

The **tixStdButtonBox** command creates a new window (given by the *pathName* argument) and makes it into a StdButtonBox widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the StdButtonBox such as its cursor and relief.

The StdButtonBox widget is a group of Standard buttons for Motif−like dialog boxes.

WIDGET COMMAND

The **tixStdButtonBox** command creates a new Tcl command whose name is the same as the path name of the StdButtonBox's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the StdButtonBox widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for StdButtonBox widgets:

pathName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixStdButtonBox** command.

pathName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixStdButtonBox** command.

pathName ***invoke*** *buttonName*
Invoke the button subwidget with the name *buttonName*.

pathName ***subwidget*** *name ?args?*
When no additional arguments are given, returns the pathname of the subwidget of the specified name.

When no additional arguments are given, the widget command of the specified subwidget will be called with these parameters.

BINDINGS

TixStdButtonBox widgets have no default bindings. The button subwidgets retain their default Tk bindings.

KEYWORDS

container, button box, widget

tixTree − Create and manipulate tixTree widgets

SYNOPSIS

tixTree *pathName* ?*options*?

SUPER−CLASS

The **TixTree** class is derived from the **TixScrolledHList** class and inherits all the commands, options and subwidgets of its super−class.

STANDARD OPTIONS

TixTree supports all the standard options of a frame widget. See the **options** manual entry for details on the standard options.

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−browsecmd***
Database Name: ***browseCmd***
Database Class: ***BrowseCmd***

Specifies a command to call whenever the user browses on an entry (usually by single−clicking on the entry). The command is called with one argument, the pathname of the entry.

Command−Line Name: ***−closecmd***
Database Name: ***closeCmd***
Database Class: ***CloseCmd***

Specifies a command to call whenever an entry needs to be closed (See the BINDINGS section below). This command is called with one argument, the pathname of the entry. This command should perform appropriate actions to close the specified entry. If the **−closecmd** option is not specified, the default closing action is to hide all child entries of the specified entry.

Command−Line Name: ***−command***

Database Name: ***command***
Database Class: ***Command***

Specifies a command to call whenever the user activates an entry (usually by double−clicking on the entry). The command is called with one argument, the pathname of the entry.

Command−Line Name: ***−ignoreinvoke***
Database Name: ***ignoreInvoke***
Database Class: ***IgnoreInvoke***

A Boolean value that specifies when a branch should be opened or closed. A branch will always be opened or closed when the user presses the (+) and (−) indicators. However, when the user invokes a branch (by doublc−clicking or pressing <Return>), the branch will be opened or closed only if **−ignoreinvoke** is set to false (the default setting).

Command−Line Name: ***−opencmd***
Database Name: ***openCmd***
Database Class: ***OpenCmd***

Specifies a command to call whenever an entry needs to be opened (See the BINDINGS section below). This command is called with one argument, the pathname of the entry. This command should perform appropriate actions to open the specified entry. If the **−opencmd** option is not specified, the default opening action is to show all the child entries of the specified entry.

SUBWIDGETS

Name: **hlist**
Class: **TixHList**

The hierarchical listbox that displays the tree.

Name: **hsb**
Class: **Scrollbar**

The horizontal scrollbar subwidget.

Name: **vsb**
Class: **Scrollbar**

The vertical scrollbar subwidget.

DESCRIPTION

The **tixTree** command creates a new window (given by the *pathName* argument) and makes it into a Tree widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the Tree widget such as its cursor and relief.

The Tree widget can be used to display hierachical data in a tree form. The user can adjust the view of the tree by opening or closing parts of the tree.

To display a static tree structure, you can add the entries into the **hlist** subwidget and hide any entries as desired. Then you can call the **autosetmode** method. This will set up the Tree widget so that it handles all the *open* and *close* events automatically.

The above method is not applicable if you want to maintain a dynamic tree structure, i.e, you do not know all the entries in the tree and you need to add or delete entries subsequently. To do this, you should first create the entries in the **hlist** subwidget. Then, use the setmode method to indicate the entries that can be opened or closed, and use the **−opencmd** and **−closecmd** options to handle the opening and closing events.

WIDGET COMMANDS

The **tixTree** command creates a new Tcl command whose name is the same as the path name of the Tree's window. This command may be used to invoke various operations on the widget. It has the following general form:

```
pathName option ?arg arg ...?
```

PathName is the name of the command, which is the same as the Tree widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for Tree widgets:

pathName ***autosetmode***
: This command calls the **setmode** method for all the entries in this Tree widget: if an entry has no child entries, its mode is set to **none**. Otherwise, if the entry has any hidden child entries, its mode is set to **open**; otherwise its mode is set to **close**.

pathName ***cget*** *option*
: Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **tixTree** command.

pathName ***close*** *entryPath*
: Close the entry given by *entryPath* if its *mode* is **close**.

pathName ***configure*** *?option? ?value option value ...?*
: Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **tixTree** command.

pathName ***getmode*** *entryPath*
: Returns the current *mode* of the entry given by *entryPath*.

pathName ***open*** *entryPath*
: Open the entry givaen by *entryPath* if its *mode* is **open**.

pathName ***setmode*** *entryPath mode*
: This command is used to indicate whether the entry given by *entryPath* has children entries and whether the children are visible. *mode* must be one of **open**, **close** or

none. If *mode* is set to **open**, a (+) indicator is drawn next the the entry. If *mode* is set to **close**, a (−) indicator is drawn next the the entry. If *mode* is set to **none**, no indicators will be drawn for this entry. The default *mode* is none. The **open** mode indicates the entry has hidden children and this entry can be opened by the user. The **close** mode indicates that all the children of the entry are now visible and the entry can be closed by the user.

pathName ***subwidget*** *name ?args?*

When no options are given, this command returns the pathname of the subwidget of the specified name. When options are given, the widget command of the specified subwidget will be called with these options.

BINDINGS

The basic mouse and keyboard bindings of the Tree widget are the same as the bindings of the HList widget. In addition, the entries can be opened or closed under the following conditions:

[1]

If the *mode* of the entry is **open**, it can be opened by clicking on its (+) indicator or double−clicking on the entry.

[2]

If the *mode* of the entry is **close**, it can be closed by clicking on its (−) indicator or double−clicking on the entry.

KEYWORDS

tree, hierarchical listbox, widget

Chapter 4: Display Items

tixDisplayStyle − Create style object for Tix display items.

SYNOPSIS

tixDisplayStyle *itemType* ?−*stylename name*? ?−*refwindow pathName*? ?*options value ...*?

DESCRIPTION

The Tix **Display Items** mechanism is devised to solve a general problem: many Tix widgets (both existing and planned ones) display many items of many types simutaneously.

For example, a hierarchical listbox widget (HList) can display items of images, plain text and subwindows in the form of a hierarchy. Another widget, the tabular listbox, (TList, currently planned and will be released in Tix 4.1) also display items of the same types, although it arranges the items in a tabular form. Yet another widget, the spreadsheet widget, also displays similar types items, but in yet another format.

In these examples, the display items in different widgets are only different in how they are arranged by the **host widget**. In Tix, display items are clearly separated from the host widgets. The advantage is two−fold: first, the creation and configuration of display items become uniform across different host widgets. Second, new display item types can be added without the need to modify the existing host widgets.

In a way, Tix display items are similar to the items inside Tk the canvas widget. However, unlike the Tix display items, the canvas items are not independent of the canvas widget; this makes it impossible to use the canvas items inside other types of TK widgets.

The appearance of a display item is controlled by a set of *attributes*. It is observed that each the attributes usually fall into one of two categroies: "*individual*" or "*collective*". For example, the text items inside a HList widget may all display a different text string; however, in most cases, the text items share the same color, font and spacing. Instead of keeping a duplicated version of the same attributes inside each display item, it will be advantageous to put the collective attributes in a special object called a **display style**. First, there is the space concern: a host widget may have many thousands of items; keeping dupilcated attributes will be very wasteful. Second, when it becomes necessary to change a collective attribute, such as changing all the text items' foreground color to red, it will be more efficient to change only the display style object than to modify all the text items one by one.

The attributes of the a display item are thus stored in two places: it has a set of **item options** to store its individual attributes. Each display item is also associated with a *display style*, which specifies the collective attributes of all items associated with itself.

The division between the individual and collective attributes are fixed and cannot be changed. Thus, when it becomes necessary for some items to differ in their collective attributes, two or more **display styles** can be used. For example, suppose you want to display two columns of text items inside an HList widget, one column in red and the other in blue. You can create a TextStyle object called "red", which defines a red foreground, and another called "blue", which defines a blue foreground. You can then associate all text items of the first column to "red" and the second column to "blue".

DISPLAY ITEM TYPES AND OPTIONS

Currently there are four types of display items: **text**, **image**, **imagetext** and **window**. (TODO: need to document the "image" item)

IMAGETEXT ITEMS

Display items of the type **imagetext** are used to display an image together with a text string. Imagetext items support the following options:

ITEM OPTIONS

Command−Line Name: ***−bitmap***
Database Name: ***bitmap***
Database Class: ***Bitmap***
Specifies the bitmap to display in the item.

Command−Line Name: ***−image***
Database Name: ***image***
Database Class: ***Image***
Specifies the image to display in the item. When both the **−bitmap** and **−image** options are specified, only the image will be displayed.

Command−Line Name: ***−style***
Database Name: ***imageTextStyle***
Database Class: ***ImageTextStyle***
Specifies the display style to use for this item. Must be the name of a **imagetext** display style that has already be created by the **tixDisplayStyle** command.

Command−Line Name: ***−showimage***
Database Name: ***showImage***
Database Class: ***ShowImage***
A Boolean value that specifies whether the image/bitmap should be displayed.

Command−Line Name: ***−showtext***
Database Name: ***showText***
Database Class: ***ShowText***
A Boolean value that specifies whether the text string should be displayed.

Command−Line Name: ***−text***
Database Name: ***text***
Database Class: ***Text***

Specifies the text string to display in the item.

Command−Line Name: ***−underline***
Database Name: ***underline***
Database Class: ***Underline***
Specifies the integer index of a character to underline in the text string in the item. 0 corresponds to the first character of the text displayed in the widget, 1 to the next character, and so on.

STYLE OPTIONS

The style information of **imagetext** items are stored in the **imagetext** display style. The following options are supported:

STANDARD OPTIONS

activeBackground activeForeground
anchor background
disabledBackground disabledForeground
foreground font
justify padX
padY selectBackground
selectForeground wrapLength

See the **options** manual entry for details on the standard options.

STYLE−SPECIFIC OPTIONS

Name: **gap**
Class: **Gap**
Switch: **−gap**

Specifies the distance between the bitmap/image and the text string, in number of pixels.

TEXT ITEMS

Display items of the type **text** are used to display a text string in a widget. Text items support the following options:

ITEM OPTIONS

Command−Line Name: ***−style***
Database Name: ***textStyle***
Database Class: ***TextStyle***
Specifies the display style to use for this text item. Must be the name of a **text** display style that has already be created by the **tixDisplayStyle** command.

Command−Line Name: ***−text***
Database Name: ***text***
Database Class: ***Text***

Specifies the text string to display in the item.

Command−Line Name: ***−underline***
Database Name: ***underline***
Database Class: ***Underline***
Specifies the integer index of a character to underline in the item. 0 corresponds to the first character of the text displayed in the widget, 1 to the next character, and so on.

STYLE OPTIONS

STANDARD OPTIONS

activeBackground activeForeground
anchor background
disabledBackground disabledForeground
foreground font
justify padX
padY selectBackground
selectForeground wrapLength

See the **options** manual entry for details on the standard options.

WINDOW ITEMS

Display items of the type **window** are used to display a sub−window in a widget. **Window** items support the following options:

ITEM OPTIONS

Command−Line Name: ***−style***
Database Name: ***windowStyle***
Database Class: ***WindowStyle***
Specifies the display style to use for this window item. Must be the name of a **window** display style that has already be created by the **tixDisplayStyle** command.

Command−Line Name: ***−window or −widget***
Database Name: ***window***
Database Class: ***Window***
Specifies the sub−window to display in the item.

STYLE OPTIONS

STANDARD OPTIONS

anchor padX padY

See the **options** manual entry for details on the standard options.

CREATING DISPLAY ITEMS

Display items do not exist on their and thus they cannot be created independently of the widgets they reside in. As a rule, display items are created by special widget commands of

their "host" widgets. For example, the HList widgets has a command **item** which can be used to create new display items. The following code creates a new imagetext item at the third column of the entry foo inside an HList widget:

```
tixHList .h −columns 3
.h add foo
.h item create foo 2 −itemtype imagetext −text Hello −image image1
```

The **item create** command of the HList widget accepts a variable number of arguments. The special argument **−itemtype** specifies which type of display item to create. Options that are valid for this type of display items can then be specified by one or more *option−value* pairs.

After the display item is created, they can then be configured or destroyed using the commands provided by the host widget. For example, the HList widget has the command **item configure**, **item cget** and **item delete** for accessing the display items.

CREATING AND MANIPULATING DISPLAY STYLES

Display styles are created by the command **tixDisplayStyle**:

itemType must be one of the existing display items types such as **text**, **imagetext**, **window** or any new types added by the user. Additional arguments can be given in one or more *option−value* pairs. *option* can be any of the valid option for this display style or any of the following:

−stylename *name*
: Specifies a name for this style. If unspecified, then a default name will be chosen for this style.

−refwindow *pathName*
: Specifies a window to use for determine the default values of the display type. If unspecified, the main window will be used. Default values for the display types can be set via the options database. The following example sets the **−disablebackground** and **−disabledforeground** options of a **text** display style via the option database:

```
option add *table.list*disabledForeground blue
option add *table.list*disabledBackground darkgray
tixDisplayStyle text −refwindow .table.list −fg red
```

By using the option database to set the options of the display styles, we can advoid hard−coding the option values and give the user more flexibility in customization. See option(n) for a detailed description of the option database.

STYLE COMMAND

The **tixDisplayStyle** command creates a new Tcl command whose name is the same as the name of the newly created display style. This command may be used to invoke various operations on the display style. It has the following general form:

```
styleName option ?arg arg ...?
```

styleName is the name of the command. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible:

styleName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the valid options of this display style.

styleName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options of the display style. If no *option* is specified, returns a list describing all of the available options for *styleName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the valid options of this display style.

styleName ***delete***

Destroy this display style object.

EXAMPLE

The following example creates two columns of data in a HList widget. The first column is in red and the second column in blue. The colors of the columns are controlled by two different **text** styles. Also, the anchor and font of the second column is chosen so that the income data is aligned properly.

```
set courier {courier 14}
set h [tixHList .h −columns 2]; pack $h
set red  [tixDisplayStyle text −fg #800000]
set blue [tixDisplayStyle text −fg #000080 \
          −anchor e −font $courier]

foreach n {{Joe $10,000} {Peter $20,000} {Raj $90,000}} {
    set entry [$h addchild {}]
    $h item create $entry 0 −itemtype text \
        −text [lindex $n 0] −style $red
    $h item create $entry 1 −itemtype text \
        −text [lindex $n 1] −style $blue
}
```

KEYWORDS

display item, display style, imagetext

Chapter 5: Image Types

compound − multi−line compound image type.

SYNOPSIS

image create compound ?*name*? ?*options*?

DESCRIPTION

Compound image types can be used to create images that consists of multiple horizontal lines; each line is composed of a series of items (texts, bitmaps, images or spaces) arranged from left to right. Compound images are mainly used to embed complex drawings into widgets that support the **−image** option. As shown in the EXAMPLE section below, a compound image can be used to display a bitmap and a text string simutaneously in a TK **button** widget.

CREATING COMPOUND IMAGES

Like all images, compound images are created using the **image create** command. Compound images support the following *options*:

−background *color*
Specifies the background color of the compound image. This color is also used as the default background color for the bitmap items in the compound image.

−borderwidth *pixels*
Specifies a non−negative value indicating the width of the 3−D border drawn around the compound image.

−font *font*
Specifies the default font for the text items in the compound image.

−foreground *color*
Specifies the default foreground color for the bitmap and text items in the compound image.

−padx *value*
Specifies a non−negative value indicating how much extra space to request for the compound image in the X−direction. The *value* may have any of the forms acceptable to **Tk_GetPixels**.

−pady *value*

Specifies a non−negative value indicating how much extra space to request for the compound image in the Y−direction.

−relief *value*
Specifies the 3−D effect desired for the background of the compound image. Acceptable values are **raised**, **sunken**, **flat**, **ridge**, and **groove**.

−showbackground *value*
Specifies whether the background and the 3D borders should be drawn. Must be a valid boolean value. By default the background is not drawn and the compound image appears to have a transparent background.

−window *pathName*
Specifies the window in which the compound image is displayed. One compound image can be displayed in only one window. When that window is destroyed, the compound image is automatically destroyed as well. This option must be specified when calling the **image create compound** command and cannot be changed by the **configure** image command.

IMAGE COMMAND

When a compound image is created, Tk also creates a new command whose name is the same as the image. This command may be used to invoke various operations on the image. It has the following general form:

```
imageName option ?arg arg ...?
```

Option and the *arg*s determine the exact behavior of the command. The following commands are possible for compound images:

imageName ***add line*** *?option value ...?*
Creates a new line at the bottom of the compound image. Lines support the following *options*:

−anchor *value*
Specifies how the line should be aligned along the horizontal axis. When the values are **w**, **sw** or **nw**, the line is aligned to the left. When the values are **c**, **s** or **n**, the line is aligned to the middle. When the values are **e**, **se** or **ne**, the line is aligned to the right.

−padx *value*
Specifies a non−negative value indicating how much extra space to request for this line in the X−direction.

imageName ***add*** *item−type ?option value ...?*
Creates a new item of the type *item−type* at the end of the last line of the compound image. All types of items support these following common *options*:

−anchor *value*
Specifies how the item should be aligned along the vertical axis. When the values are **n**, **nw** or **ne**, the item is aligned to the top of the line. When the values are **c**, **w** or **e**, the item is aligned to the middle of the line. When the values are **s**, **se** or **sw**, the item is aligned to the bottom of the line.

−padx *value*
Specifies a non−negative value indicating how much extra space to request for this item in the X−direction.

−pady *value*
Specifies a non−negative value indicating how much extra space to request for this item in the Y−direction.

item−type can be any of the following:

imageName ***add bitmap*** *?option value ...?*
Creates a new bitmap item of at the end of the last line of the compound image. Additional *options* accepted by the bitmap type are:

−background *color*
Specifies the background color of the bitmap item.

−bitmap *name*
Specifies a bitmap to display in this item, in any of the forms acceptable to **Tk_GetBitmap**.

−foreground *color*
Specifies the foreground color of the bitmap item.

imageName ***add image*** *?option value ...?*
Creates a new image item of at the end of the last line of the compound image. Additional *options* accepted by the image type are:

−image *name*
Specifies an image to display in this item. *name* must have been created with the **image create** command.

imageName ***add space*** *?option value ...?*
Creates a new space item of at the end of the last line of the compound image. Space items do not display anything. They just acts as space holders that add additional spaces between items inside a compound image. Additional *options* accepted by the image type are:

−width *value*
Specifies the width of this space. The *value* may have any of the forms acceptable to **Tk_GetPixels**.

−height *value*
Specifies the height of this space. The *value* may have any of the forms acceptable to **Tk_GetPixels**.

imageName ***add text*** *?option value ...?*
Creates a new text item of at the end of the last line of the compound image. Additional *options* accepted by the text type are:

−background *color*
Specifies the background color of the text item.

−font *name*
Specifies the font to be used for this text item.

−foreground *color*
Specifies the foreground color of the text item.

−justify *value*
When there are multiple lines of text displayed in a text item, this option determines how the lines line up with each other. *value* must be one of **left**, **center**, or **right**. **Left** means that the lines' left edges all line up, **center** means that the lines' centers are aligned, and **right** means that the lines' right edges line up.

−text *string*
Specifies a text string to display in this text item.

−underline *value*
Specifies the integer index of a character to underline in the text item. 0 corresponds to the first character of the text displayed in the text item, 1 to the next character, and so on.

−wraplength *value*
This option specifies the maximum line length of the label string on this text item. If the line length of the label string exceeds this length, it is wrapped onto the next line, so that no line is longer than the specified length. The value may be specified in any of the standard forms for screen distances. If this value is less than or equal to 0 then no wrapping is done: lines will break only at newline characters in the text.

imageName ***cget*** *option*
Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **image create compound** command.

imageName ***configure*** *?option? ?value option value ...?*
Query or modify the configuration options for the image. If no *option* is specified, returns a list describing all of the available options for *imageName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **image create compound** command, except the **−window** option

EXAMPLE

The following example creates a compound image with a bitmap and a text string and places this image into a Tk **button** widget. Notice that the image must be created after the creation of the window that it resides in.

```
set btn [button .b]
set img [image create compound −window $btn]
$img add line
```

```
$img add bitmap -bitmap warning
$img add space -width 8
$img add text -text "Warning" -underline 0
$btn config -image $img
pack $btn
```

KEYWORDS

image, compound

pixmap − image type for the XPM file format.

SYNOPSIS

image create pixmap ?*name*? ?*options*?

DESCRIPTION

XPM is a popular X Window image file format for storing color icons. The **pixmap** image type defined by the **Tix** library can be used to create color images using XPM files.

CREATING PIXMAPS

Like all images, pixmaps are created using the **image create** command. Pixmaps support the following *options*:

−data *string*
: Specifies the contents of the source pixmap as a string. The string must adhere to the XPM file format (e.g., as generated by the **pixmap** program). If both the **−data** and **−file** options are specified, the **−data** option takes precedence. Please note that the XPM file parsing code in the xpm library is extremely fragile. The first line of the string must be "**/* XPM */**" or otherwise a segmatation fault will be caused.

−file *name*
: *name* gives the name of a file whose contents define the source pixmap. The file must adhere to the XPM file format (e.g., as generated by the **pixmap** program).

IMAGE COMMAND

When a pixmap image is created, Tk also creates a new command whose name is the same as the image. This command may be used to invoke various operations on the image. It has the following general form:

```
imageName option ?arg arg ...?
```

Option and the *arg*s determine the exact behavior of the command. The following commands are possible for pixmap images:

imageName ***cget*** *option*

Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **image create pixmap** command.

imageName ***configure*** *?option? ?value option value ...?*

Query or modify the configuration options for the image. If no *option* is specified, returns a list describing all of the available options for *imageName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **image create pixmap** command.

KEYWORDS

pixmap, image, XPM

Chapter 6: Other Commands

tix − Manipulate internal states of the Tix library

SYNOPSIS

tix *option* ?*arg arg* ...?

CONFIGURATION OPTIONS

This manual page descripts the **tix** command, which manipulates the internal states of the Tix library. If you're looking for a general introduction to the Tix library, please refer to the **TixIntro** manual page.

The Tix application context supports the following configuration options. Usually, these options are set using the X resource database, i.e., in the user's **.Xdefault** file. For example, to choose a different color scheme for the Tix widgets, these two lines can be added to the user's **.Xdefault** file:

```
*TixScheme:  Gray
*TixFontSet: 14Point
```

Command−Line Name: ***−binding***
Database Name: ***binding***
Database Class: ***Binding***
This is an obsolete option.

Command−Line Name: ***−debug***
Database Name: ***debug***
Database Class: ***Debug***
Specifies whether the Tix widgets should run in debug mode.

Command−Line Name: ***−fontset***
Database Name: ***tixFontSet***
Database Class: ***TixFontSet***
Specifies the fontset to use for the Tix widgets. Valid options are **TK**, **TkWin**, **12Point** and **14Point**. **TK** specifies that the standard TK fonts should be used. The default value is **14Point**.

Command−Line Name: ***−scheme***
Database Name: ***tixScheme***
Database Class: ***TixScheme***
Specifies the color scheme to use for the Tix widgets. Valid options are **TK**, **TkWin**, **Gray**, **Blue**, **Bisque**, **SGIGray** and **TixGray**. The default value is **TixGray**. If you want the standard TK color scheme, you can use the value **TK**. If you want to use the TK 3.6 bisque color scheme, you can use the value **Bisque**.

Command−Line Name: ***−schemepriority***
Database Name: ***tixSchemePriority***
Database Class: ***TixSchemePriority***

Specifies the priority level of the TK options set by th Tix schemes. Please refer to the TK **option** manual page for a discussion of the priority level of Tix options. The default value is 79, which makes the Tix schemes at a higher priority than the settings in the .Xdefaults file. If you want to allow the Tix schemes to be overridden by the settings in the .Xdefaults file, you can set the following line in you .Xdefaults file:

```
*TixSchemePriority: 21
```

DESCRIPTION

The **tix** command provides access to miscellaneous elements of Tix's internal state and the Tix **application context**. Most of the information manipulated by this command pertains to the application as a whole, or to a screen or display, rather than to a particular window. The command can take any of a number of different forms depending on the *option* argument. The legal forms are:

tix addbitmapdir *directory*

Tix maintains a list of directory under which which the **tix getimage** and **tix getbitmap** commands will search for image files. The standard bitmap directory is **$TIX_LIBRARY/bitmaps**. The **addbitmapdir** command adds *directory* into this list. By using this command, the image files of an applications can also be located using the **tix getimage** ot **tix getbitmap** command.

tix cget *option*

Returns the current value of the configuration option given by *option*. *Option* may be any of the options described in the **CONFIGURATION OPTIONS** section.

tix configure *?option? ?value option value ...?*

Query or modify the configuration options of the Tix application context. If no *option* is specified, returns a list describing all of the available options (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given option(s) to have the given value(s); in this case the command returns an empty string. *Option* may be any of the options described in the **CONFIGURATION OPTIONS** section.

tix filedialog *?class?*

Returns the file selection dialog that may be shared among different modules of this application. This command will create a file selection dialog widget when it is called the first time. This dialog will be returned by all subsequent calls to **tix filedialog**. An optional *class* parameter can be passed to specified what type of file selection dialog widget is desired. Possible options are **tixFileSelectDialog** or **tixExFileSelectDialog.**

tix getbitmap *name*

Locates a bitmap file of the name *name*.xpm or *name* in one of the bitmap directories (see the **addbitmapdir** command above). By using **tix getbitmap**, you can advoid hard coding the pathnames of the bitmap files in your application. When successful,

it returns the complete pathname of the bitmap file, prefixed with the character **@**. The returned value can be used to configure the **−bitmap** option of the TK and Tix widgets.

tix getimage *name*

Locates an image file of the name *name*.xpm, *name*.xbm or *name*.ppm in one of the bitmap directories (see the **addbitmapdir** command above). If more than one file with the same name (but different extensions) exist, then the image type is chosen according to the depth of the X display: xbm images are chosen on monochrome displays and color images are chosen on color displays. By using **tix getimage**, you can advoid hard coding the pathnames of the image files in your application. When successful, this command returns the name of the newly created image, which can be used to configure the **−image** option of the TK and Tix widgets.

tix option *?args ...?*

Manipulates the options manitained by the Tix scheme mechanism. Available options are:

```
active_bg   active_fg   bg
bold_font   dark1_bg    dark1_fg
dark2_bg    dark2_fg    disabled_fg
fg          fixed_font  font
inactive_bg inactive_fg input1_bg
input2_bg   italic_font light1_bg
light1_fg   light2_bg   light2_fg
menu_font   output1_bg  output2_bg
select_bg   select_fg   selector
```

The arguments to the **tix option** command can take the following form(s):

tix option get *option*

Returns the current value of *option*.

tix resetoptions *newScheme newFontSet ?newScmPrio?*

Resets the scheme and fontset of the Tix application to *newScheme* and *newFontSet*, respectively. This affects only those widgets created **after** this call. Therefore, it is best to call the **resetoptions** command **before** the creation of any widgets in a Tix application. The optional parameter *newScmPrio* can be given to reset the priority level of the TK options set by the Tix schemes.

BUGS

[1]

In this release of Tix, the following configuration options have been disabled. Assigning values to them will cause no effect:

```
−binding
−debug
−fontset
−scheme
−schemepriority
```

In addition, the following options to the **tix** command have been disabled. Invoking the **tix** command with these options will cause no effect:

```
resetoptions
```

[2]

Because of the way TK handles the X option database, after tixwish has started up, it is not possible to reset the color schemes and font sets using the **tix config** command. Instead, the **tix resetoptions** command must be used.

The tk_setPalette command does not work very well under Tix. To use it, one must follow these steps:

```
tix resetoptions TK TK
tk_setPalette lightblue
```

KEYWORDS

file selection dialog

tixDestroy − Destroy Tix Objects

SYNOPSIS

tixDestroy *objectName*

DESCRIPTION

The **tixDestroy** destroys a Tix object whose class is declared by the **tixClass** keyword. When the object is destroyed, its **Destructor** function is called and the memory allocated for this object is freed.

KEYWORDS

Tix, Object

tixForm − Geometry manager based on attachment rules

SYNOPSIS

tixForm *option arg ?arg ...?*

DESCRIPTION

The **tixForm** command is used to communicate with the **tixForm** Geometry Manager, a geometry manager that arranges the geometry of the children in a parent window according to attachment rules. The **tixForm** geometry manager is very flexible and powerful; it can be used to emulate all the existing features of the Tk packer and placer geometry managers (see **pack**, **place**). The **tixForm** command can have any of several forms, depending on the *option* argument:

tixForm *slave ?options?*

If the first argument to **tixForm** is a window name (any value starting with ``."), then the command is processed in the same way as **tixForm configure**.

tixForm check *master*

This command checks whether there is circular dependency in the attachments of the master's slaves (see the section **CIRCULAR DEPENDENCY** below). It returns the Boolean value **TRUE** if it discover circular dependency and **FALSE** otherwise.

tixForm configure *slave ?−option value ...?*

Sets or adjusts the attachment values of the slave window according to the *−option value* argument pairs.

−b *attachment*

Abbreviation for the **−bottom** option.

−bottom *attachment*

Specifies an attachment for the bottom edge of the slave window. The attachment must specied according to the section **SPECIFYING ATTACHMENTS** below.

−bottomspring *weight*

Specifies the weight of the spring at the bottom edge of the slave window. See the section **USING SPRINGS** below.

−bp *value*

Abbreviation for the **−padbottom** option.

−bs *weight*

Abbreviation for the **−bottomspring** option.

−fill *master*

Specifies the fillings when springs are used for this widget. The value must be **x**, **y**, **both** or **none**.

−in *master*

Places the slave window into the specified master window. If the slave was originally in another master window, all attachment values with respect to the original master window are discarded. Even if the attachment values are the same as in the original master window, they need to be specified again. The **−in** flag, when needed, must appear as the first flag after the name of the slave. Otherwise an error is generated.

−l *attachment*
Abbreviation for the **−left** option.

−left *attachment*
Specifies an attachment for the left edge of the slave window. The attachment must specied according to the section **SPECIFYING ATTACHMENTS** below.

−leftspring *weight*
Specifies the weight of the spring at the left edge of the slave window. See the section **USING SPRINGS** below.

−lp *value*
Abbreviation for the **−padleft** option.

−ls *weight*
Abbreviation for the **−leftspring** option.

−padbottom *value*
Specifies the amount of external padding to leave on the bottom side of the slave. The *value* may have any of the forms acceptable to **Tk_GetPixels**.

−padleft *value*
Specifies the amount of external padding to leave on the left side of the slave.

−padright *value*
Specifies the amount of external padding to leave on the right side of the slave.

−padtop *value*
Specifies the amount of external padding to leave on the top side of the slave.

−padx *value*
Specifies the amount of external padding to leave on both the left and the right sides of the slave.

−pady *value*
Specifies the amount of external padding to leave on both the top and the bottom sides of the slave.

−r *attachment*
Abbreviation for the **−right** option.

−right *attachment*
Specifies an attachment for the right edge of the slave window. The attachment must specied according to the section **SPECIFYING ATTACHMENTS** below.

−rightspring *weight*
Specifies the weight of the spring at the right edge of the slave window. See the section **USING SPRINGS** below.

−rp *value*
Abbreviation for the **−padright** option.

−rs *weight*
Abbreviation for the **−rightspring** option.

−t *attachment*
Abbreviation for the **−top** option.

−top *attachment*
Specifies an attachment for the top edge of the slave window. The attachment must specied according to the section **SPECIFYING ATTACHMENTS** below.

−topspring *weight*
Specifies the weight of the spring at the top edge of the slave window. See the section **USING SPRINGS** below.

−tp *value*
Abbreviation for the **−padtop** option.

−ts *weight*
Abbreviation for the **−topspring** option.

tixForm forget *slave ?slave ...?*
Removes each of the slaves from its master and unmaps their windows. The slaves will no longer be managed by tixForm. All attachment values with respect to their master windows are discarded. If another slave is attached to this slave, then the attachment of the other slave will be changed to grid attachment based on its geometry.

tixForm grid *master ?x_size y_size?*
When *x_size* and *y_size* are given, this command returns the number of grids of the master window in a pair of integers of the form {*x_size y_size*}. When both *x_size* and *y_size* are given, this command changes the number of horizontal and vertical grids on the master window.

tixForm info *slave ?option?*
Queries the attachment options of a slave window. *option* can be any of the options accepted by the **tixForm configure** command. If *option* is given, only the value of that option is returned. Otherwise, this command returns a list whose elements are the current configuration state of the slave given in the same *option−value* form that might be specified to **tixForm configure**. The first two elements in this list list are "**−in** *master*" where *master* is the slave's master window.

tixForm slaves *master*
Returns a list of all of the slaves for the master window. The order of the slaves in the list is the same as their order in the packing order. If master has no slaves then an empty string is returned.

SPECIFYING ATTACHMENTS

One can specify an attachment for each side of a slave window managed by tixForm. An attachment is specified in the the form "−*side* {*anchor_point offset*}". −*side* can be one of **−top**, **−bottom**, **−left** or **−right**.

Offset is given in screen units (i.e. any of the forms acceptable to **Tk_GetPixels**). A positive offset indicates shifting to a position to the right or bottom of an anchor point. A negative offset indicates shifting to a position to the left or top of an anchor point.

Anchor_point can be given in one of the following forms:

Grid Attachment

The master window is divided into a number of horizontal and vertical grids. By default the master window is divided into 100x100 grids; the number of grids can be adjusted by the **tixForm grid** command. A grid attachment anchor point is given by a **%** sign followed by an **integer** value. For example, **%0** spceifies the first grid line (the top or left edge of the master window). **%100** spceifies the last grid line (the bottom or right edge of the master window).

Opposite Side Attachment

Opposite attachment specifies an anchor point located on the **opposite** side of another slave widget, which must be managed by tixForm in the same master window. An opposite attachment anchor point is given by the name of another widget. For example, "tixForm .b −top {.a 0}" attaches the **top** side of the widget **.b** to the **bottom** of the widget **.a**.

Parallel Side Attachment

Opposite attachment specifies an anchor point located on the **same** side of another slave widget, which must be managed by tixForm in the same master window. An parallel attachment anchor point is given by the sign **&** follwed by the name of another widget. For example, "tixForm .b −top {&.a 0}" attaches the **top** side of the widget **.b** to the **top** of the widget **.a**, making the **top** sides of these two widgets at the same vertical position in their parent window.

No Attachment

Specifies a side of the slave to be attached to nothing, indicated by the keyword **none**. When the **none** anchor point is given, the offser must be zero. When a side of a slave is attached to **{none 0}**, the position of this side is calculated by the position of the other side and the natural size of the slave. For example, if a the **left** side of a widget is attached to **{%0 100}**, its **right** side attached to **{none 0}**, and the natural size of the widget is **50** pixels, the **right** side of the widget will be positioned at pixel **{%0 149}**. When both **−top** and **−bottom** are attached to **none**, then by default **−top** will be attached to **{%0 0}**. When both **−left** and **−right** are attached to none, then by default **−left** will be attached to **{%0 0}**.

Shifting effects can be achieved by specifying a non−zero offset with an anchor point. In the following example, the **top** side of widget **.b** is attached to the **bottom** of **.a**; hence **.b** always appears below **.a**. Also, the left edge of **.b** is attached to the **left** side of **.a** with a 10 pixel offest. Therefore, the **left** edge of **.b** is always shifted 10 pixels to the right of **.a**'s **left** edge:

```
tixForm .b −left {.a 10} −top {.a 0}
```

ABBREVIATIONS: Certain abbreviations can be made on the attachment specifications: First an offset of zero can be omitted. Thus, the following two lines are equivalent:

```
tixForm .b −top {.a 0} −right {%100 0}
tixForm .b −top {.a}   −right {%100}
```

Also, because of the way TCL handles lists, when you omit the offset, you can also leave out the braces. So you can further simplify the above to:

```
tixForm .b −top .a −right %100
```

In the second case, when the anchor point is omitted, the offset must be given. A default anchor point is chosen according to the value of the offset. If the anchor point is **0** or positive, the default anchor point %0 is used; thus, "tixForm .b −top 15" attaches the top edge of **.b** to a position 15 pixels below the top edge of the master window. If the anchor point is "**−0**" or negative, the default anchor point **%100** is used; thus, "tixForm .a −right −2" attaches the right edge of **.a** to a position 2 pixels to the left of the master window's **right** edge. An further example below shows a command with its equivalent abbreviation.

```
tixForm .b −top {%0 10} −bottom {%100 0}
tixForm .b −top 10      −bottom −0
```

USING SPRINGS

To be written.

ALGORITHM OF TIXFORM

TixForm starts with any slave in the list of slaves of the master window. Then it tries to determine the position of each side of the slave.

If the attachment of a side of the slave is grid attachment, the position of the side is readily determined.

If the attachment of this side is **none**, then tixForm tries to determine the position of the opposite side first, and then use the position of the opposite side and the natural size of the slave to determine the position of this side.

If the attachment is opposite or parallel widget attachments, then tixForm tries to determine the positions of the other widget first, and then use the positions of the other widget and the natural size of the slave determine the position of this side. This recursive algorithmis carried on until the positions of all slaves are determined.

CIRCULAR DEPENDENCY

The algorithm of tixForm will fail if a circular dependency exists in the attachments of the slaves. For example:

```
tixForm .c −left .b
tixForm .b −right .c
```

In this example, the position of the left side of **.b** depends on the right side of **.c**, which in turn depends on the left side of **.b**.

When a circular dependency is discovered during the execution of the tixForm algorithm, tixForm will generate a background error and the geometry of the slaves are undefined (and will be arbitrary). Notice that tixForm only executes the algorithm when the specification of the slaves' attachments is complete. Therefore, it allows intermediate states of circular dependency during the specification of the slaves' attachments. Also, unlike the Motif Form manager widget, tixForm defines circular dependency as "*dependency in the same dimension*". Therefore, the following code fragment will does not have circular dependency because the two widgets do not depend on each other in the same dimension (**.b** depends **.c** in the horizontal dimension and **.c** depends on **.b** in the vertical dimension):

```
tixForm .b -left .c
tixForm .c -top .b
```

BUGS

Springs have not been fully implemented yet.

KEYWORDS

form, geometry management

tixGetBoolean − Get the boolean value of a string.

SYNOPSIS

tixGetBoolean *?−nocomplain? string*

DESCRIPTION

The command **tixGetBoolean** returns "0" if the string is a valid TCL string for the boolean value FALSE. It returns "1" if the string is a valid TCL string for the boolean value TRUE.

When the string is not a valid TCL boolean value and the **−nocomplain** option is specified, **tixGetBoolean** will return "0". Otherwise it will generate an error.

tixGetInt − Get the integer value of a string.

SYNOPSIS

tixGetInt *?−nocomplain? ?−trunc? string*

DESCRIPTION

The command **tixGetInt** converts any number into an integer number. By default, it will round the number to the nearest integer. When the **−trunc** option is specified, the number is truncated instead of rounded.

When the string is not a valid TCL numerical value and the **−nocomplain** option is specified, **tixGetInt** will return "0". Otherwise it will generate an error.

tixMwm − Communicate with the Motif(tm) window manager.

SYNOPSIS

tixMwm *option pathName* ?*args*?

COMMAND OPTIONS

tixMwm decoration *pathName ?option? ?value ...?*

When no options are given, this command returns the values of all the decorations options for the toplevel window with the *pathName*. When only one option is given without specifying the value, the current value of that option is returned. When more than one "option value" pairs are passed to this command, the specified values will be assigned to the corresponding options. As a result, the appearance of the Motif decorations around the toplevel window will be changed. Possible options are: **−border**, **−menu**, **−maximize**, **−minimize**, **−resizeh** and **−title**. The value must be a Boolean value. The values returned by this command are undefined when the window is not managed by mwm.

tixMwm ismwmrunning *pathName*

This returns true if mwm is running on the screen where the specified window is located, false otherwise.

tixMwm protocol *pathName*

When no additional options are given, this command returns all protocols associated with this toplevel window.

tixMwm protocol *pathName* ***activate*** *protocol_name*

Activate the mwm protocol message in mwm's menu.

tixMwm protocol *pathName* ***add*** *protocol_name menu_message*

Add a new mwm protocol message for this toplevel window. The message is identified by the string name specified in *protocol_name*. A menu item will be added into mwm's menu as specified by *menu_message*. Once a new mwm protocol message is added to a toplevel, it can be catched by the TK **wm protocol** command. Here is an example:

```
tixMwm protocol . add MY_PRINT_HELLO \
    {"Print Hello"  _H Ctrl<Key>H}
wm protocol . MY_PRINT_HELLO {puts Hello}
```

tixMwm protocol *pathName* ***deactivate*** *protocol_name*

Deactivate the mwm protocol message in mwm's menu.

tixMwm protocol *pathName* ***delete*** *protocol_name*

Delete the mwm protocol message from mwm's menu. Please note that the window manager protocol handler associated with this protocol (by the **wm protocol** command) is not deleted automatically. You have to delete the protocol handle explicitly. E.g.:

```
tixMwm protocol . delete MY_PRINT_HELLO
wm protocol . MY_PRINT_HELLO {}
```

BUGS

On some versions of Mwm, the **−border** will not disappear unless **−resizeh** is turned off. Also, the −title will not disappear unless all of **−title**, **−menu**, **−maximize** and **−minimize** are turned off.

KEYWORDS

Motif window manager, mwm

tixUtils − Utility commands in Tix.

SYNOPSIS

tixDescendants *pathName*
tixDisableAll *pathName*
tixEnableAll *pathName*
tixPushGrab ?*−global*? *window*
tixPopGrab

DESCRIPTION

tixDescendants *pathName*

Returns a list of all the descendant widgets of *pathName* plus *pathName* itself.

tixDisableAll *pathName*
Disables *pathName* and all its descendants.

tixEnableAll *pathName*
Enables *pathName* and all its descendants.

tixPushGrab *?−global? window*
The **tixPushGrab** and **tixPopGrab** commands allows you to perform "cascade−grabbing". **tixPushGrab** calls the **grab** command on *window* and saves *window* on a grabbing stack.

tixPopGrab
tixPopGrab pops the top−most element from the grabbing stack and release its grab. If the grabbing stack is not empty, then **tixPopGrab** will execute grab(n) on the current top−most element in the grabbing stack.

NOTES

Some Tix widgets (for example, tixComboBox and tixPanedWindow) grabs the screen on certain occasions using **tixPushGrab** and **tixPopGrab**. Therefore, if you need to grab the screen when these widgets are present, you should also call **tixPushGrab** and **tixPopGrab** in place of the Tk **grab** and **grab release** commands. Otherwise, the behavior of these widgets may be undefined.

KEYWORDS

grab

Chapter 7: Executable Programs

tixwish − Windowing shell for interpreting Tix commands.

SYNOPSIS

tixwish ?*fileName arg arg ...*?

OPTIONS

−display *display*
Display (and screen) on which to display window.

−geometry *geometry*
Initial geometry to use for window. If this option is specified, its value is stored in the **geometry** global variable of the application's Tcl interpreter.

−name *name*
Use *name* as the title to be displayed in the window, and as the name of the interpreter for **send** commands.

−sync
Execute all X server commands synchronously, so that errors are reported immediately. This will result in much slower execution, but it is useful for debugging.

DESCRIPTION

Note: the use of the **tixwish** program is deprecated. You should use the standard **wish** program from Tk and access Tix via the "package require Tix" command.

Tixwish is a simple program consisting of the Tcl command language, the Tk toolkit, the Tix library, and a main program that reads commands from standard input or from a file. It creates a main window and then processes Tcl commands. If **tixwish** is invoked with no arguments, or with a first argument that starts with ``−", then it reads Tcl commands interactively from standard input. It will continue processing commands until all windows have been deleted or until end−of−file is reached on standard input. If there exists a file **.tixwishrc** in the home directory of the user, **tixwish** evaluates the file as a Tcl script just before reading the first command from standard input.

If **tixwish** is invoked with an initial *fileName* argument, then *fileName* is treated as the name of a script file. **Tixwish** will evaluate the script in *fileName* (which presumably creates a user interface), then it will respond to events until all windows have been deleted. Commands will not be read from standard input. There is no automatic evaluation of **.tixwishrc** in this case,

but the script file can always **source** it if desired.

OPTIONS

Tixwish automatically processes all of the command−line options described in the **OPTIONS** summary above. Any other command−line arguments besides these are passed through to the application using the **argc** and **argv** variables described later.

APPLICATION NAME AND CLASS

The name of the application, which is used for purposes such as **send** commands, is taken from the **−name** option, if it is specified; otherwise it is taken from *fileName*, if it is specified, or from the command name by which **tixwish** was invoked. In the last two cases, if the name contains a ``/'' character, then only the characters after the last slash are used as the application name.

The class of the application, which is used for purposes such as specifying options with a **RESOURCE_MANAGER** property or .Xdefaults file, is the same as its name except that the first letter is capitalized.

VARIABLES

Tixwish sets the following Tcl variables:

argc
Contains a count of the number of *arg* arguments (0 if none), not including the options described above.

argv
Contains a Tcl list whose elements are the *arg* arguments (not including the options described above), in order, or an empty string if there are no *arg* arguments.

argv0
Contains *fileName* if it was specified. Otherwise, contains the name by which **tixwish** was invoked.

geometry
If the **−geometry** option is specified, **tixwish** copies its value into this variable. If the variable still exists after *fileName* has been evaluated, **tixwish** uses the value of the variable in a **wm geometry** command to set the main window's geometry.

tcl_interactive
Contains 1 if **tixwish** is reading commands interactively (**fileName** was not specified and standard input is a terminal−like device), 0 otherwise.

X RESOURCES

Tixwish makes use of several X Resources to determine the **Toolkit Options** for the Tix library. These X resources must be set using **RESOURCE_MANAGER** properties or .Xdefaults files **before tixwish** starts running. These resources must be associated with the main window of the **tixwish** application. These options include:

Name: **tixScheme**
Class: **TixScheme**

Specifies the color scheme to use for the Tix application. Currently only these schemes are supported: Blue, Gray, SGIGray, TixGray, and TK.

Name: **tixFontSet**
Class: **TixFontSet**

Specifies the FontSet to use for the Tix application. A FontSet designates the fonts to use for different types of widgets. Currently only these FontSets are supported: 12Point, 14Point and TK.

For example, you may put these two lines in your .Xdefaults file *tixwish.tixScheme: Gray *tixwish.tixFontSet: 12Point

SCRIPT FILES

If you create a Tcl script in a file whose first line is

```
#!/usr/local/bin/tixwish
```

then you can invoke the script file directly from your shell if you mark it as executable. This assumes that **tixwish** has been installed in the default location in /usr/local/bin; if it's installed somewhere else then you'll have to modify the above line to match. Many UNIX systems do not allow the **#!** line to exceed about 30 characters in length, so be sure that the **tixwish** executable can be accessed with a short file name.

PROMPTS

When **tixwish** is invoked interactively it normally prompts for each command with ``**%** ''. You can change the prompt by setting the variables **tcl_prompt1** and **tcl_prompt2**. If variable **tcl_prompt1** exists then it must consist of a Tcl script to output a prompt; instead of outputting a prompt **tixwish** will evaluate the script in **tcl_prompt1**. The variable **tcl_prompt2** is used in a similar way when a newline is typed but the current command isn't yet complete; if **tcl_prompt2** isn't set then no prompt is output for incomplete commands.

KEYWORDS

shell, wish, Tk, toolkit

Appendix 1: Tk Commands

This chapter includes a subset of the Tk Commands that are frequently mentioned in the Tix documentation. Please consult your Tcl/Tk documentation for other Tcl/Tk references.

frame − Create and manipulate frame widgets

SYNOPSIS

frame *pathName* ?*options*?

STANDARD OPTIONS

−borderwidth or −bd, borderWidth, BorderWidth
−cursor, cursor, Cursor
−highlightbackground, highlightBackground, HighlightBackground
−highlightcolor, highlightColor, HighlightColor
−highlightthickness, highlightThickness, HighlightThickness
−relief, relief, Relief
−takefocus, takeFocus, TakeFocus

WIDGET−SPECIFIC OPTIONS

Command−Line Name: ***−background***
Database Name: ***background***
Database Class: ***Background***

This option is the same as the standard **background** option except that its value may also be specified as an empty string. In this case, the widget will display no background or border, and no colors will be consumed from its colormap for its background and border.

Command−Line Name: ***−class***
Database Name: ***class***
Database Class: ***Class***

Specifies a class for the window. This class will be used when querying the option database for the window's other options, and it will also be used later for other purposes such as bindings. The **class** option may not be changed with the **configure** widget command.

Command−Line Name: ***−colormap***
Database Name: ***colormap***
Database Class: ***Colormap***

Specifies a colormap to use for the window. The value may be either **new**, in which case a new colormap is created for the window and its children, or the name of another window (which must be on the same screen and have the same visual as *pathName*), in which case the new window will use the colormap from the specified

window. If the **colormap** option is not specified, the new window uses the same colormap as its parent. This option may not be changed with the **configure** widget command.

Command−Line Name: ***−container***
Database Name: ***container***
Database Class: ***Container***
The value must be a boolean. If true, it means that this window will be used as a container in which some other application will be embedded (for example, a Tk toplevel can be embedded using the **−use** option). The window will support the appropriate window manager protocols for things like geometry requests. The window should not have any children of its own in this application. This option may not be changed with the **configure** widget command.

Command−Line Name: ***−height***
Database Name: ***height***
Database Class: ***Height***
Specifies the desired height for the window in any of the forms acceptable to **Tk_GetPixels**. If this option is less than or equal to zero then the window will not request any size at all.

Command−Line Name: ***−visual***
Database Name: ***visual***
Database Class: ***Visual***
Specifies visual information for the new window in any of the forms accepted by **Tk_GetVisual**. If this option is not specified, the new window will use the same visual as its parent. The **visual** option may not be modified with the **configure** widget command.

Command−Line Name: ***−width***
Database Name: ***width***
Database Class: ***Width***
Specifies the desired width for the window in any of the forms acceptable to **Tk_GetPixels**. If this option is less than or equal to zero then the window will not request any size at all.

DESCRIPTION

The **frame** command creates a new window (given by the *pathName* argument) and makes it into a frame widget. Additional options, described above, may be specified on the command line or in the option database to configure aspects of the frame such as its background color and relief. The **frame** command returns the path name of the new window.

A frame is a simple widget. Its primary purpose is to act as a spacer or container for complex window layouts. The only features of a frame are its background color and an optional 3−D border to make the frame appear raised or sunken.

WIDGET COMMAND

The **frame** command creates a new Tcl command whose name is the same as the path name of the frame's window. This command may be used to invoke various operations on the widget. It has the following general form:

pathName option ?arg arg ...?

PathName is the name of the command, which is the same as the frame widget's path name. *Option* and the *arg*s determine the exact behavior of the command. The following commands are possible for frame widgets:

pathName ***cget*** *option*
: Returns the current value of the configuration option given by *option*. *Option* may have any of the values accepted by the **frame** command.

pathName ***configure*** *?option? ?value option value ...?*
: Query or modify the configuration options of the widget. If no *option* is specified, returns a list describing all of the available options for *pathName* (see **Tk_ConfigureInfo** for information on the format of this list). If *option* is specified with no *value*, then the command returns a list describing the one named option (this list will be identical to the corresponding sublist of the value returned if no *option* is specified). If one or more *option−value* pairs are specified, then the command modifies the given widget option(s) to have the given value(s); in this case the command returns an empty string. *Option* may have any of the values accepted by the **frame** command.

BINDINGS

When a new frame is created, it has no default event bindings: frames are not intended to be interactive.

KEYWORDS

frame, widget

image − Create and manipulate images

SYNOPSIS

image *option* ?*arg arg ...*?

DESCRIPTION

The **image** command is used to create, delete, and query images. It can take several different forms, depending on the *option* argument. The legal forms are:

image create *type ?name? ?option value ...?*
: Creates a new image and returns its name. *type* specifies the type of the image, which must be one of the types currently defined (e.g., **bitmap**). *name* specifies the name for the image; if it is omitted then Tk picks a name of the form **image***x*, where *x* is an

integer. There may be any number of *option−value* pairs, which provide configuration options for the new image. The legal set of options is defined separately for each image type; see below for details on the options for built−in image types. If an image already exists by the given name then it is replaced with the new image and any instances of that image will redisplay with the new contents.

image delete *?name name ...?*
Deletes each of the named images and returns an empty string. If there are instances of the images displayed in widgets, the images won't actually be deleted until all of the instances are released. However, the association between the instances and the image manager will be dropped. Existing instances will retain their sizes but redisplay as empty areas. If a deleted image is recreated with another call to **image create**, the existing instances will use the new image.

image height *name*
Returns a decimal string giving the height of image *name* in pixels.

image names
Returns a list containing the names of all existing images.

image type *name*
Returns the type of image *name* (the value of the *type* argument to **image create** when the image was created).

image types
Returns a list whose elements are all of the valid image types (i.e., all of the values that may be supplied for the *type* argument to **image create**).

image width *name*
Returns a decimal string giving the width of image *name* in pixels.

BUILT−IN IMAGE TYPES

The following image types are defined by Tk so they will be available in any Tk application. Individual applications or extensions may define additional types.

bitmap
Each pixel in the image displays a foreground color, a background color, or nothing. See the **bitmap** manual entry for more information.

photo
Displays a variety of full−color images, using dithering to approximate colors on displays with limited color capabilities. See the **photo** manual entry for more information.

KEYWORDS

height, image, types of images, width

options − Standard options supported by widgets

DESCRIPTION

This manual entry describes the common configuration options supported by widgets in the Tk toolkit. Every widget does not necessarily support every option (see the manual entries for individual widgets for a list of the standard options supported by that widget), but if a widget does support an option with one of the names listed below, then the option has exactly the effect described below.

In the descriptions below, ``Command−Line Name'' refers to the switch used in class commands and **configure** widget commands to set this value. For example, if an option's command−line switch is **−foreground** and there exists a widget **.a.b.c**, then the command

```
.a.b.c  configure  −foreground black
```

may be used to specify the value **black** for the option in the the widget **.a.b.c**. Command−line switches may be abbreviated, as long as the abbreviation is unambiguous. ``Database Name'' refers to the option's name in the option database (e.g. in .Xdefaults files). ``Database Class'' refers to the option's class value in the option database.

Command−Line Name: ***−activebackground***
Database Name: ***activeBackground***
Database Class: ***Foreground***

Specifies background color to use when drawing active elements. An element (a widget or portion of a widget) is active if the mouse cursor is positioned over the element and pressing a mouse button will cause some action to occur. If strict Motif compliance has been requested by setting the **tk_strictMotif** variable, this option will normally be ignored; the normal background color will be used instead. For some elements on Windows and Macintosh systems, the active color will only be used while mouse button 1 is pressed over the element.

Command−Line Name: ***−activeborderwidth***
Database Name: ***activeBorderWidth***
Database Class: ***BorderWidth***

Specifies a non−negative value indicating the width of the 3−D border drawn around active elements. See above for definition of active elements. The value may have any of the forms acceptable to **Tk_GetPixels**. This option is typically only available in widgets displaying more than one element at a time (e.g. menus but not buttons).

Command−Line Name: ***−activeforeground***
Database Name: ***activeForeground***
Database Class: ***Background***

Specifies foreground color to use when drawing active elements. See above for definition of active elements.

Command−Line Name: ***−anchor***
Database Name: ***anchor***
Database Class: ***Anchor***

Specifies how the information in a widget (e.g. text or a bitmap) is to be displayed in the widget. Must be one of the values **n**, **ne**, **e**, **se**, **s**, **sw**, **w**, **nw**, or **center**. For example, **nw** means display the information such that its top−left corner is at the top−left corner of the widget.

Command−Line Name: ***−background or −bg***
Database Name: ***background***
Database Class: ***Background***

Specifies the normal background color to use when displaying the widget.

Command−Line Name: ***−bitmap***
Database Name: ***bitmap***
Database Class: ***Bitmap***

Specifies a bitmap to display in the widget, in any of the forms acceptable to **Tk_GetBitmap**. The exact way in which the bitmap is displayed may be affected by other options such as **anchor** or **justify**. Typically, if this option is specified then it overrides other options that specify a textual value to display in the widget; the **bitmap** option may be reset to an empty string to re−enable a text display. In widgets that support both **bitmap** and **image** options, **image** will usually override **bitmap**.

Command−Line Name: ***−borderwidth or −bd***
Database Name: ***borderWidth***
Database Class: ***BorderWidth***

Specifies a non−negative value indicating the width of the 3−D border to draw around the outside of the widget (if such a border is being drawn; the **relief** option typically determines this). The value may also be used when drawing 3−D effects in the interior of the widget. The value may have any of the forms acceptable to **Tk_GetPixels**.

Command−Line Name: ***−cursor***
Database Name: ***cursor***
Database Class: ***Cursor***

Specifies the mouse cursor to be used for the widget. The value may have any of the forms acceptable to **Tk_GetCursor**.

Command−Line Name: ***−disabledforeground***
Database Name: ***disabledForeground***
Database Class: ***DisabledForeground***

Specifies foreground color to use when drawing a disabled element. If the option is specified as an empty string (which is typically the case on monochrome displays), disabled elements are drawn with the normal foreground color but they are dimmed by drawing them with a stippled fill pattern.

Command−Line Name: ***−exportselection***
Database Name: ***exportSelection***
Database Class: ***ExportSelection***

Specifies whether or not a selection in the widget should also be the X selection. The value may have any of the forms accepted by **Tcl_GetBoolean**, such as **true**, **false**, **0**, **1**, **yes**, or **no**. If the selection is exported, then selecting in the widget deselects the current X selection, selecting outside the widget deselects any widget selection, and the widget will respond to selection retrieval requests when it has a selection. The default is usually for widgets to export selections.

Command−Line Name: ***−font***
Database Name: ***font***
Database Class: ***Font***
Specifies the font to use when drawing text inside the widget.

Command−Line Name: ***−foreground or −fg***
Database Name: ***foreground***
Database Class: ***Foreground***
Specifies the normal foreground color to use when displaying the widget.

Command−Line Name: ***−highlightbackground***
Database Name: ***highlightBackground***
Database Class: ***HighlightBackground***
Specifies the color to display in the traversal highlight region when the widget does not have the input focus.

Command−Line Name: ***−highlightcolor***
Database Name: ***highlightColor***
Database Class: ***HighlightColor***
Specifies the color to use for the traversal highlight rectangle that is drawn around the widget when it has the input focus.

Command−Line Name: ***−highlightthickness***
Database Name: ***highlightThickness***
Database Class: ***HighlightThickness***
Specifies a non−negative value indicating the width of the highlight rectangle to draw around the outside of the widget when it has the input focus. The value may have any of the forms acceptable to **Tk_GetPixels**. If the value is zero, no focus highlight is drawn around the widget.

Command−Line Name: ***−image***
Database Name: ***image***
Database Class: ***Image***
Specifies an image to display in the widget, which must have been created with the **image create** command. Typically, if the **image** option is specified then it overrides other options that specify a bitmap or textual value to display in the widget; the **image** option may be reset to an empty string to re−enable a bitmap or text display.

Command−Line Name: ***−insertbackground***
Database Name: ***insertBackground***
Database Class: ***Foreground***
Specifies the color to use as background in the area covered by the insertion cursor. This color will normally override either the normal background for the widget (or the selection background if the insertion cursor happens to fall in the selection).

Command−Line Name: ***−insertborderwidth***
Database Name: ***insertBorderWidth***
Database Class: ***BorderWidth***
Specifies a non−negative value indicating the width of the 3−D border to draw around the insertion cursor. The value may have any of the forms acceptable to **Tk_GetPixels**.

Command−Line Name: ***−insertofftime***

Database Name: ***insertOffTime***
Database Class: ***OffTime***
Specifies a non−negative integer value indicating the number of milliseconds the insertion cursor should remain ``off'' in each blink cycle. If this option is zero then the cursor doesn't blink: it is on all the time.

Command−Line Name: ***−insertontime***
Database Name: ***insertOnTime***
Database Class: ***OnTime***
Specifies a non−negative integer value indicating the number of milliseconds the insertion cursor should remain ``on'' in each blink cycle.

Command−Line Name: ***−insertwidth***
Database Name: ***insertWidth***
Database Class: ***InsertWidth***
Specifies a value indicating the total width of the insertion cursor. The value may have any of the forms acceptable to **Tk_GetPixels**. If a border has been specified for the insertion cursor (using the **insertBorderWidth** option), the border will be drawn inside the width specified by the **insertWidth** option.

Command−Line Name: ***−jump***
Database Name: ***jump***
Database Class: ***Jump***
For widgets with a slider that can be dragged to adjust a value, such as scrollbars, this option determines when notifications are made about changes in the value. The option's value must be a boolean of the form accepted by **Tcl_GetBoolean**. If the value is false, updates are made continuously as the slider is dragged. If the value is true, updates are delayed until the mouse button is released to end the drag; at that point a single notification is made (the value ``jumps'' rather than changing smoothly).

Command−Line Name: ***−justify***
Database Name: ***justify***
Database Class: ***Justify***
When there are multiple lines of text displayed in a widget, this option determines how the lines line up with each other. Must be one of **left**, **center**, or **right**. **Left** means that the lines' left edges all line up, **center** means that the lines' centers are aligned, and **right** means that the lines' right edges line up.

Command−Line Name: ***−orient***
Database Name: ***orient***
Database Class: ***Orient***
For widgets that can lay themselves out with either a horizontal or vertical orientation, such as scrollbars, this option specifies which orientation should be used. Must be either **horizontal** or **vertical** or an abbreviation of one of these.

Command−Line Name: ***−padx***
Database Name: ***padX***
Database Class: ***Pad***
Specifies a non−negative value indicating how much extra space to request for the widget in the X−direction. The value may have any of the forms acceptable to **Tk_GetPixels**. When computing how large a window it needs, the widget will add this amount to the width it would normally need (as determined by the width of the

things displayed in the widget); if the geometry manager can satisfy this request, the widget will end up with extra internal space to the left and/or right of what it displays inside. Most widgets only use this option for padding text: if they are displaying a bitmap or image, then they usually ignore padding options.

Command−Line Name: ***−pady***
Database Name: ***padY***
Database Class: ***Pad***

Specifies a non−negative value indicating how much extra space to request for the widget in the Y−direction. The value may have any of the forms acceptable to **Tk_GetPixels**. When computing how large a window it needs, the widget will add this amount to the height it would normally need (as determined by the height of the things displayed in the widget); if the geometry manager can satisfy this request, the widget will end up with extra internal space above and/or below what it displays inside. Most widgets only use this option for padding text: if they are displaying a bitmap or image, then they usually ignore padding options.

Command−Line Name: ***−relief***
Database Name: ***relief***
Database Class: ***Relief***

Specifies the 3−D effect desired for the widget. Acceptable values are **raised**, **sunken**, **flat**, **ridge**, **solid**, and **groove**. The value indicates how the interior of the widget should appear relative to its exterior; for example, **raised** means the interior of the widget should appear to protrude from the screen, relative to the exterior of the widget.

Command−Line Name: ***−repeatdelay***
Database Name: ***repeatDelay***
Database Class: ***RepeatDelay***

Specifies the number of milliseconds a button or key must be held down before it begins to auto−repeat. Used, for example, on the up− and down−arrows in scrollbars.

Command−Line Name: ***−repeatinterval***
Database Name: ***repeatInterval***
Database Class: ***RepeatInterval***

Used in conjunction with **repeatDelay**: once auto−repeat begins, this option determines the number of milliseconds between auto−repeats.

Command−Line Name: ***−selectbackground***
Database Name: ***selectBackground***
Database Class: ***Foreground***

Specifies the background color to use when displaying selected items.

Command−Line Name: ***−selectborderwidth***
Database Name: ***selectBorderWidth***
Database Class: ***BorderWidth***

Specifies a non−negative value indicating the width of the 3−D border to draw around selected items. The value may have any of the forms acceptable to **Tk_GetPixels**.

Command−Line Name: ***−selectforeground***
Database Name: ***selectForeground***
Database Class: ***Background***

Specifies the foreground color to use when displaying selected items.

Command−Line Name: ***−setgrid***
Database Name: ***setGrid***
Database Class: ***SetGrid***

Specifies a boolean value that determines whether this widget controls the resizing grid for its top−level window. This option is typically used in text widgets, where the information in the widget has a natural size (the size of a character) and it makes sense for the window's dimensions to be integral numbers of these units. These natural window sizes form a grid. If the **setGrid** option is set to true then the widget will communicate with the window manager so that when the user interactively resizes the top−level window that contains the widget, the dimensions of the window will be displayed to the user in grid units and the window size will be constrained to integral numbers of grid units. See the section GRIDDED GEOMETRY MANAGEMENT in the **wm** manual entry for more details.

Command−Line Name: ***−takefocus***
Database Name: ***takeFocus***
Database Class: ***TakeFocus***

Determines whether the window accepts the focus during keyboard traversal (e.g., Tab and Shift−Tab). Before setting the focus to a window, the traversal scripts consult the value of the **takeFocus** option. A value of **0** means that the window should be skipped entirely during keyboard traversal. **1** means that the window should receive the input focus as long as it is viewable (it and all of its ancestors are mapped). An empty value for the option means that the traversal scripts make the decision about whether or not to focus on the window: the current algorithm is to skip the window if it is disabled, if it has no key bindings, or if it is not viewable. If the value has any other form, then the traversal scripts take the value, append the name of the window to it (with a separator space), and evaluate the resulting string as a Tcl script. The script must return **0**, **1**, or an empty string: a **0** or **1** value specifies whether the window will receive the input focus, and an empty string results in the default decision described above. Note: this interpretation of the option is defined entirely by the Tcl scripts that implement traversal: the widget implementations ignore the option entirely, so you can change its meaning if you redefine the keyboard traversal scripts.

Command−Line Name: ***−text***
Database Name: ***text***
Database Class: ***Text***

Specifies a string to be displayed inside the widget. The way in which the string is displayed depends on the particular widget and may be determined by other options, such as **anchor** or **justify**.

Command−Line Name: ***−textvariable***
Database Name: ***textVariable***
Database Class: ***Variable***

Specifies the name of a variable. The value of the variable is a text string to be displayed inside the widget; if the variable value changes then the widget will automatically update itself to reflect the new value. The way in which the string is displayed in the widget depends on the particular widget and may be determined by other options, such as **anchor** or **justify**.

Command−Line Name: ***−troughcolor***

Database Name: ***troughColor***
Database Class: ***Background***

Specifies the color to use for the rectangular trough areas in widgets such as scrollbars and scales. This option is ignored for scrollbars on Windows (native widget doesn't recognize this option).

Command−Line Name: ***−underline***
Database Name: ***underline***
Database Class: ***Underline***

Specifies the integer index of a character to underline in the widget. This option is used by the default bindings to implement keyboard traversal for menu buttons and menu entries. 0 corresponds to the first character of the text displayed in the widget, 1 to the next character, and so on.

Command−Line Name: ***−wraplength***
Database Name: ***wrapLength***
Database Class: ***WrapLength***

For widgets that can perform word−wrapping, this option specifies the maximum line length. Lines that would exceed this length are wrapped onto the next line, so that no line is longer than the specified length. The value may be specified in any of the standard forms for screen distances. If this value is less than or equal to 0 then no wrapping is done: lines will break only at newline characters in the text.

Command−Line Name: ***−xscrollcommand***
Database Name: ***xScrollCommand***
Database Class: ***ScrollCommand***

Specifies the prefix for a command used to communicate with horizontal scrollbars. When the view in the widget's window changes (or whenever anything else occurs that could change the display in a scrollbar, such as a change in the total size of the widget's contents), the widget will generate a Tcl command by concatenating the scroll command and two numbers. Each of the numbers is a fraction between 0 and 1, which indicates a position in the document. 0 indicates the beginning of the document, 1 indicates the end, .333 indicates a position one third the way through the document, and so on. The first fraction indicates the first information in the document that is visible in the window, and the second fraction indicates the information just after the last portion that is visible. The command is then passed to the Tcl interpreter for execution. Typically the **xScrollCommand** option consists of the path name of a scrollbar widget followed by ``set'', e.g. ``.x.scrollbar set'': this will cause the scrollbar to be updated whenever the view in the window changes. If this option is not specified, then no command will be executed.

Command−Line Name: ***−yscrollcommand***
Database Name: ***yScrollCommand***
Database Class: ***ScrollCommand***

Specifies the prefix for a command used to communicate with vertical scrollbars. This option is treated in the same way as the **xScrollCommand** option, except that it is used for vertical scrollbars and is provided by widgets that support vertical scrolling. See the description of **xScrollCommand** for details on how this option is used.

KEYWORDS

class, name, standard option, switch

Appendix 2: Tk Library References

This chapter includes a subset of the Tk Library References that are frequently mentioned in the Tix documentation. Please consult your Tcl/Tk documentation for other Tcl/Tk references.

Tk_ConfigureWidget, Tk_Offset, Tk_ConfigureInfo, Tk_ConfigureValue, Tk_FreeOptions − process configuration options for widgets

SYNOPSIS

#include <tk.h>
int
Tk_ConfigureWidget(*interp, tkwin, specs, argc, argv, widgRec, flags***)**
int
Tk_Offset(*type, field***)**
int
Tk_ConfigureInfo(*interp, tkwin, specs, widgRec, argvName, flags***)**
int
Tk_ConfigureValue(*interp, tkwin, specs, widgRec, argvName, flags***)**
Tk_FreeOptions(*specs, widgRec, display, flags***)**

ARGUMENTS

Tcl_Interp ****interp*** *(in)*
Interpreter to use for returning error messages.

Tk_Window ***tkwin*** *(in)*
Window used to represent widget (needed to set up X resources).

Tk_ConfigSpec ****specs*** *(in)*
Pointer to table specifying legal configuration options for this widget.

int ***argc*** *(in)*
Number of arguments in *argv*.

char *****argv*** *(in)*
Command−line options for configuring widget.

char ****widgRec*** *(in/out)*
Points to widget record structure. Fields in this structure get modified by **Tk_ConfigureWidget** to hold configuration information.

int ***flags*** *(in)*
If non−zero, then it specifies an OR−ed combination of flags that control the

processing of configuration information. TK_CONFIG_ARGV_ONLY causes the option database and defaults to be ignored, and flag bits TK_CONFIG_USER_BIT and higher are used to selectively disable entries in *specs*.

type name ***type*** *(in)*
The name of the type of a widget record.

field name ***field*** *(in)*
The name of a field in records of type *type*.

char ****argvName*** *(in)*
The name used on Tcl command lines to refer to a particular option (e.g. when creating a widget or invoking the **configure** widget command). If non−NULL, then information is returned only for this option. If NULL, then information is returned for all available options.

Display ****display*** *(in)*
Display containing widget whose record is being freed; needed in order to free up resources.

DESCRIPTION

Tk_ConfigureWidget is called to configure various aspects of a widget, such as colors, fonts, border width, etc. It is intended as a convenience procedure to reduce the amount of code that must be written in individual widget managers to handle configuration information. It is typically invoked when widgets are created, and again when the **configure** command is invoked for a widget. Although intended primarily for widgets, **Tk_ConfigureWidget** can be used in other situations where *argc−argv* information is to be used to fill in a record structure, such as configuring graphical elements for a canvas widget or entries of a menu.

Tk_ConfigureWidget processes a table specifying the configuration options that are supported (*specs*) and a collection of command−line arguments (*argc* and *argv*) to fill in fields of a record (*widgRec*). It uses the option database and defaults specified in *specs* to fill in fields of *widgRec* that are not specified in *argv*. **Tk_ConfigureWidget** normally returns the value TCL_OK; in this case it does not modify *interp*. If an error occurs then TCL_ERROR is returned and **Tk_ConfigureWidget** will leave an error message in *interp−>result* in the standard Tcl fashion. In the event of an error return, some of the fields of *widgRec* could already have been set, if configuration information for them was successfully processed before the error occurred. The other fields will be set to reasonable initial values so that **Tk_FreeOptions** can be called for cleanup.

The *specs* array specifies the kinds of configuration options expected by the widget. Each of its entries specifies one configuration option and has the following structure:

```
typedef struct {
    int type;
    char *argvName;
    char *dbName;
    char *dbClass;
    char *defValue;
    int offset;
    int specFlags;
    Tk_CustomOption *customPtr;
} Tk_ConfigSpec;
```

The *type* field indicates what type of configuration option this is (e.g. TK_CONFIG_COLOR for a color value, or TK_CONFIG_INT for an integer value). The *type* field indicates how to use the value of the option (more on this below). The *argvName* field is a string such as ``−font" or ``−bg", which is compared with the values in *argv* (if *argvName* is NULL it means this is a grouped entry; see GROUPED ENTRIES below). The *dbName* and *dbClass* fields are used to look up a value for this option in the option database. The *defValue* field specifies a default value for this configuration option if no value is specified in either *argv* or the option database. *Offset* indicates where in *widgRec* to store information about this option, and *specFlags* contains additional information to control the processing of this configuration option (see FLAGS below). The last field, *customPtr*, is only used if *type* is TK_CONFIG_CUSTOM; see CUSTOM OPTION TYPES below.

Tk_ConfigureWidget first processes *argv* to see which (if any) configuration options are specified there. *Argv* must contain an even number of fields; the first of each pair of fields must match the *argvName* of some entry in *specs* (unique abbreviations are acceptable), and the second field of the pair contains the value for that configuration option. If there are entries in *spec* for which there were no matching entries in *argv*, **Tk_ConfigureWidget** uses the *dbName* and *dbClass* fields of the *specs* entry to probe the option database; if a value is found, then it is used as the value for the option. Finally, if no entry is found in the option database, the *defValue* field of the *specs* entry is used as the value for the configuration option. If the *defValue* is NULL, or if the TK_CONFIG_DONT_SET_DEFAULT bit is set in *flags*, then there is no default value and this *specs* entry will be ignored if no value is specified in *argv* or the option database.

Once a string value has been determined for a configuration option, **Tk_ConfigureWidget** translates the string value into a more useful form, such as a color if *type* is TK_CONFIG_COLOR or an integer if *type* is TK_CONFIG_INT. This value is then stored in the record pointed to by *widgRec*. This record is assumed to contain information relevant to the manager of the widget; its exact type is unknown to **Tk_ConfigureWidget**. The *offset* field of each *specs* entry indicates where in *widgRec* to store the information about this configuration option. You should use the **Tk_Offset** macro to generate *offset* values (see below for a description of **Tk_Offset**). The location indicated by *widgRec* and *offset* will be referred to as the ``target" in the descriptions below.

The *type* field of each entry in *specs* determines what to do with the string value of that configuration option. The legal values for *type*, and the corresponding actions, are:

TK_CONFIG_ACTIVE_CURSOR

The value must be an ASCII string identifying a cursor in a form suitable for passing to **Tk_GetCursor**. The value is converted to a **Tk_Cursor** by calling **Tk_GetCursor** and the result is stored in the target. In addition, the resulting cursor is made the active cursor for *tkwin* by calling **XDefineCursor**. If TK_CONFIG_NULL_OK is specified in *specFlags* then the value may be an empty string, in which case the target and *tkwin*'s active cursor will be set to **None**. If the previous value of the target wasn't **None**, then it is freed by passing it to **Tk_FreeCursor**.

TK_CONFIG_ANCHOR

The value must be an ASCII string identifying an anchor point in one of the ways accepted by **Tk_GetAnchor**. The string is converted to a **Tk_Anchor** by calling **Tk_GetAnchor** and the result is stored in the target.

TK_CONFIG_BITMAP

The value must be an ASCII string identifying a bitmap in a form suitable for passing to **Tk_GetBitmap**. The value is converted to a **Pixmap** by calling **Tk_GetBitmap** and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in *specFlags* then the value may be an empty string, in which case the target is set to **None**. If the previous value of the target wasn't **None**, then it is freed by passing it to **Tk_FreeBitmap**.

TK_CONFIG_BOOLEAN

The value must be an ASCII string specifying a boolean value. Any of the values ``true", ``yes", ``on", or ``1", or an abbreviation of one of these values, means true; any of the values ``false", ``no", ``off", or ``0", or an abbreviation of one of these values, means false. The target is expected to be an integer; for true values it will be set to 1 and for false values it will be set to 0.

TK_CONFIG_BORDER

The value must be an ASCII string identifying a border color in a form suitable for passing to **Tk_Get3DBorder**. The value is converted to a (**Tk_3DBorder ***) by calling **Tk_Get3DBorder** and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in *specFlags* then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target wasn't NULL, then it is freed by passing it to **Tk_Free3DBorder**.

TK_CONFIG_CAP_STYLE

The value must be an ASCII string identifying a cap style in one of the ways accepted by **Tk_GetCapStyle**. The string is converted to an integer value corresponding to the cap style by calling **Tk_GetCapStyle** and the result is stored in the target.

TK_CONFIG_COLOR

The value must be an ASCII string identifying a color in a form suitable for passing to **Tk_GetColor**. The value is converted to an (**XColor ***) by calling **Tk_GetColor** and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in *specFlags* then the value may be an empty string, in which case the target will be set to **None**. If the previous value of the target wasn't NULL, then it is freed by passing it to **Tk_FreeColor**.

TK_CONFIG_CURSOR

This option is identical to **TK_CONFIG_ACTIVE_CURSOR** except that the new cursor is not made the active one for *tkwin*.

TK_CONFIG_CUSTOM

This option allows applications to define new option types. The *customPtr* field of the entry points to a structure defining the new option type. See the section CUSTOM OPTION TYPES below for details.

TK_CONFIG_DOUBLE

The value must be an ASCII floating−point number in the format accepted by **strtol**. The string is converted to a **double** value, and the value is stored in the target.

TK_CONFIG_END

Marks the end of the table. The last entry in *specs* must have this type; all of its other fields are ignored and it will never match any arguments.

TK_CONFIG_FONT

The value must be an ASCII string identifying a font in a form suitable for passing to **Tk_GetFont**. The value is converted to an (**XFontStruct ***) by calling **Tk_GetFont** and the result is stored in the target. If TK_CONFIG_NULL_OK is specified in *specFlags* then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target wasn't NULL, then it is freed by passing it to **Tk_FreeFont**.

TK_CONFIG_INT

The value must be an ASCII integer string in the format accepted by **strtol** (e.g. ``0" and ``0x" prefixes may be used to specify octal or hexadecimal numbers, respectively). The string is converted to an integer value and the integer is stored in the target.

TK_CONFIG_JOIN_STYLE

The value must be an ASCII string identifying a join style in one of the ways accepted by **Tk_GetJoinStyle**. The string is converted to an integer value corresponding to the join style by calling **Tk_GetJoinStyle** and the result is stored in the target.

TK_CONFIG_JUSTIFY

The value must be an ASCII string identifying a justification method in one of the ways accepted by **Tk_GetJustify**. The string is converted to a **Tk_Justify** by calling **Tk_GetJustify** and the result is stored in the target.

TK_CONFIG_MM

The value must specify a screen distance in one of the forms acceptable to **Tk_GetScreenMM**. The string is converted to double−precision floating−point distance in millimeters and the value is stored in the target.

TK_CONFIG_PIXELS

The value must specify screen units in one of the forms acceptable to **Tk_GetPixels**. The string is converted to an integer distance in pixels and the value is stored in the target.

TK_CONFIG_RELIEF

The value must be an ASCII string identifying a relief in a form suitable for passing to **Tk_GetRelief**. The value is converted to an integer relief value by calling **Tk_GetRelief** and the result is stored in the target.

TK_CONFIG_STRING

A copy of the value is made by allocating memory space with **malloc** and copying the value into the dynamically−allocated space. A pointer to the new string is stored in the target. If TK_CONFIG_NULL_OK is specified in *specFlags* then the value may be an empty string, in which case the target will be set to NULL. If the previous value of the target wasn't NULL, then it is freed by passing it to **free**.

TK_CONFIG_SYNONYM

This *type* value identifies special entries in *specs* that are synonyms for other entries. If an *argv* value matches the *argvName* of a TK_CONFIG_SYNONYM entry, the entry isn't used directly. Instead, **Tk_ConfigureWidget** searches *specs* for another entry whose *argvName* is the same as the *dbName* field in the TK_CONFIG_SYNONYM entry; this new entry is used just as if its *argvName* had

matched the *argv* value. The synonym mechanism allows multiple *argv* values to be used for a single configuration option, such as ``−background'' and ``−bg''.

TK_CONFIG_UID
The value is translated to a **Tk_Uid** (by passing it to **Tk_GetUid**). The resulting value is stored in the target. If TK_CONFIG_NULL_OK is specified in *specFlags* and the value is an empty string then the target will be set to NULL.

TK_CONFIG_WINDOW
The value must be a window path name. It is translated to a **Tk_Window** token and the token is stored in the target.

GROUPED ENTRIES

In some cases it is useful to generate multiple resources from a single configuration value. For example, a color name might be used both to generate the background color for a widget (using TK_CONFIG_COLOR) and to generate a 3−D border to draw around the widget (using TK_CONFIG_BORDER). In cases like this it is possible to specify that several consecutive entries in *specs* are to be treated as a group. The first entry is used to determine a value (using its *argvName*, *dbName*, *dbClass*, and *defValue* fields). The value will be processed several times (one for each entry in the group), generating multiple different resources and modifying multiple targets within *widgRec*. Each of the entries after the first must have a NULL value in its *argvName* field; this indicates that the entry is to be grouped with the entry that precedes it. Only the *type* and *offset* fields are used from these follow−on entries.

FLAGS

The *flags* argument passed to **Tk_ConfigureWidget** is used in conjunction with the *specFlags* fields in the entries of *specs* to provide additional control over the processing of configuration options. These values are used in three different ways as described below.

First, if the *flags* argument to **Tk_ConfigureWidget** has the TK_CONFIG_ARGV_ONLY bit set (i.e., *flags* | TK_CONFIG_ARGV_ONLY != 0), then the option database and *defValue* fields are not used. In this case, if an entry in *specs* doesn't match a field in *argv* then nothing happens: the corresponding target isn't modified. This feature is useful when the goal is to modify certain configuration options while leaving others in their current state, such as when a **configure** widget command is being processed.

Second, the *specFlags* field of an entry in *specs* may be used to control the processing of that entry. Each *specFlags* field may consists of an OR−ed combination of the following values:

TK_CONFIG_COLOR_ONLY
If this bit is set then the entry will only be considered if the display for *tkwin* has more than one bit plane. If the display is monochromatic then this *specs* entry will be ignored.

TK_CONFIG_MONO_ONLY
If this bit is set then the entry will only be considered if the display for *tkwin* has exactly one bit plane. If the display is not monochromatic then this *specs* entry will be ignored.

TK_CONFIG_NULL_OK
: This bit is only relevant for some types of entries (see the descriptions of the various entry types above). If this bit is set, it indicates that an empty string value for the field is acceptable and if it occurs then the target should be set to NULL or **None**, depending on the type of the target. This flag is typically used to allow a feature to be turned off entirely, e.g. set a cursor value to **None** so that a window simply inherits its parent's cursor. If this bit isn't set then empty strings are processed as strings, which generally results in an error.

TK_CONFIG_DONT_SET_DEFAULT
: If this bit is one, it means that the *defValue* field of the entry should only be used for returning the default value in **Tk_ConfigureInfo**. In calls to **Tk_ConfigureWidget** no default will be supplied for entries with this flag set; it is assumed that the caller has already supplied a default value in the target location. This flag provides a performance optimization where it is expensive to process the default string: the client can compute the default once, save the value, and provide it before calling **Tk_ConfigureWidget**.

TK_CONFIG_OPTION_SPECIFIED
: This bit is set and cleared by **Tk_ConfigureWidget**. Whenever **Tk_ConfigureWidget** returns, this bit will be set in all the entries where a value was specified in *argv*. It will be zero in all other entries. This bit provides a way for clients to determine which values actually changed in a call to **Tk_ConfigureWidget**.

The TK_CONFIG_MONO_ONLY and TK_CONFIG_COLOR_ONLY flags are typically used to specify different default values for monochrome and color displays. This is done by creating two entries in *specs* that are identical except for their *defValue* and *specFlags* fields. One entry should have the value TK_CONFIG_MONO_ONLY in its *specFlags* and the default value for monochrome displays in its *defValue*; the other entry entry should have the value TK_CONFIG_COLOR_ONLY in its *specFlags* and the appropriate *defValue* for color displays.

Third, it is possible to use *flags* and *specFlags* together to selectively disable some entries. This feature is not needed very often. It is useful in cases where several similar kinds of widgets are implemented in one place. It allows a single *specs* table to be created with all the configuration options for all the widget types. When processing a particular widget type, only entries relevant to that type will be used. This effect is achieved by setting the high−order bits (those in positions equal to or greater than TK_CONFIG_USER_BIT) in *specFlags* values or in *flags*. In order for a particular entry in *specs* to be used, its high−order bits must match exactly the high−order bits of the *flags* value passed to **Tk_ConfigureWidget**. If a *specs* table is being used for N different widget types, then N of the high−order bits will be used. Each *specs* entry will have one of more of those bits set in its *specFlags* field to indicate the widget types for which this entry is valid. When calling **Tk_ConfigureWidget**, *flags* will have a single one of these bits set to select the entries for the desired widget type. For a working example of this feature, see the code in tkButton.c.

TK_OFFSET

The **Tk_Offset** macro is provided as a safe way of generating the *offset* values for entries in Tk_ConfigSpec structures. It takes two arguments: the name of a type of record, and the name of a field in that record. It returns the byte offset of the named field in records of the

given type.

TK_CONFIGUREINFO

The **Tk_ConfigureInfo** procedure may be used to obtain information about one or all of the options for a given widget. Given a token for a window (*tkwin*), a table describing the configuration options for a class of widgets (*specs*), a pointer to a widget record containing the current information for a widget (*widgRec*), and a NULL *argvName* argument, **Tk_ConfigureInfo** generates a string describing all of the configuration options for the window. The string is placed in *interp−>result*. Under normal circumstances it returns TCL_OK; if an error occurs then it returns TCL_ERROR and *interp−>result* contains an error message.

If *argvName* is NULL, then the value left in *interp−>result* by **Tk_ConfigureInfo** consists of a list of one or more entries, each of which describes one configuration option (i.e. one entry in *specs*). Each entry in the list will contain either two or five values. If the corresponding entry in *specs* has type TK_CONFIG_SYNONYM, then the list will contain two values: the *argvName* for the entry and the *dbName* (synonym name). Otherwise the list will contain five values: *argvName*, *dbName*, *dbClass*, *defValue*, and current value. The current value is computed from the appropriate field of *widgRec* by calling procedures like **Tk_NameOfColor**.

If the *argvName* argument to **Tk_ConfigureInfo** is non−NULL, then it indicates a single option, and information is returned only for that option. The string placed in *interp−>result* will be a list containing two or five values as described above; this will be identical to the corresponding sublist that would have been returned if *argvName* had been NULL.

The *flags* argument to **Tk_ConfigureInfo** is used to restrict the *specs* entries to consider, just as for **Tk_ConfigureWidget**.

TK_CONFIGUREVALUE

Tk_ConfigureValue takes arguments similar to **Tk_ConfigureInfo**; instead of returning a list of values, it just returns the current value of the option given by *argvName* (*argvName* must not be NULL). The value is returned in *interp−>result* and TCL_OK is normally returned as the procedure's result. If an error occurs in **Tk_ConfigureValue** (e.g., *argvName* is not a valid option name), TCL_ERROR is returned and an error message is left in *interp−>result*. This procedure is typically called to implement **cget** widget commands.

TK_FREEOPTIONS

The **Tk_FreeOptions** procedure may be invoked during widget cleanup to release all of the resources associated with configuration options. It scans through *specs* and for each entry corresponding to a resource that must be explicitly freed (e.g. those with type TK_CONFIG_COLOR), it frees the resource in the widget record. If the field in the widget record doesn't refer to a resource (e.g. it contains a null pointer) then no resource is freed for that entry. After freeing a resource, **Tk_FreeOptions** sets the corresponding field of the widget record to null.

CUSTOM OPTION TYPES

Applications can extend the built−in configuration types with additional configuration types by writing procedures to parse and print options of the a type and creating a structure pointing to those procedures:

```
typedef struct Tk_CustomOption {
        Tk_OptionParseProc *parseProc;
        Tk_OptionPrintProc *printProc;
        ClientData clientData;
} Tk_CustomOption;

typedef int Tk_OptionParseProc(
        ClientData clientData,
        Tcl_Interp *interp,
        Tk_Window tkwin,
      char *value,
      char *widgRec,
    int offset);

typedef char *Tk_OptionPrintProc(
        ClientData clientData,
        Tk_Window tkwin,
      char *widgRec,
    int offset,
        Tcl_FreeProc **freeProcPtr);
```

The Tk_CustomOption structure contains three fields, which are pointers to the two procedures and a *clientData* value to be passed to those procedures when they are invoked. The *clientData* value typically points to a structure containing information that is needed by the procedures when they are parsing and printing options.

The *parseProc* procedure is invoked by **Tk_ConfigureWidget** to parse a string and store the resulting value in the widget record. The *clientData* argument is a copy of the *clientData* field in the Tk_CustomOption structure. The *interp* argument points to a Tcl interpreter used for error reporting. *Tkwin* is a copy of the *tkwin* argument to **Tk_ConfigureWidget**. The *value* argument is a string describing the value for the option; it could have been specified explicitly in the call to **Tk_ConfigureWidget** or it could come from the option database or a default. *Value* will never be a null pointer but it may point to an empty string. *RecordPtr* is the same as the *widgRec* argument to **Tk_ConfigureWidget**; it points to the start of the widget record to modify. The last argument, *offset*, gives the offset in bytes from the start of the widget record to the location where the option value is to be placed. The procedure should translate the string to whatever form is appropriate for the option and store the value in the widget record. It should normally return TCL_OK, but if an error occurs in translating the string to a value then it should return TCL_ERROR and store an error message in *interp−>result*.

The *printProc* procedure is called by **Tk_ConfigureInfo** to produce a string value describing an existing option. Its *clientData*, *tkwin*, *widgRec*, and *offset* arguments all have the same meaning as for Tk_OptionParseProc procedures. The *printProc* procedure should examine the option whose value is stored at *offset* in *widgRec*, produce a string describing that option, and return a pointer to the string. If the string is stored in dynamically−allocated memory, then the procedure must set **freeProcPtr* to the address of a procedure to call to free the string's memory; **Tk_ConfigureInfo** will call this procedure when it is finished with the string. If the result string is stored in static memory then *printProc* need not do anything with the *freeProcPtr* argument.

Once *parseProc* and *printProc* have been defined and a Tk_CustomOption structure has been created for them, options of this new type may be manipulated with Tk_ConfigSpec entries whose *type* fields are TK_CONFIG_CUSTOM and whose *customPtr* fields point to the Tk_CustomOption structure.

EXAMPLES

Although the explanation of **Tk_ConfigureWidget** is fairly complicated, its actual use is pretty straightforward. The easiest way to get started is to copy the code from an existing widget. The library implementation of frames (tkFrame.c) has a simple configuration table, and the library implementation of buttons (tkButton.c) has a much more complex table that uses many of the fancy *specFlags* mechanisms.

KEYWORDS

anchor, bitmap, boolean, border, cap style, color, configuration options, cursor, custom, double, font, integer, join style, justify, millimeters, pixels, relief, synonym, uid

Tk_AllocBitmapFromObj, Tk_GetBitmap, Tk_GetBitmapFromObj, Tk_DefineBitmap, Tk_NameOfBitmap, Tk_SizeOfBitmap, Tk_FreeBitmapFromObj, Tk_FreeBitmap, Tk_GetBitmapFromData − maintain database of single−plane pixmaps

SYNOPSIS

#include <tk.h>
Pixmap
Tk_GetBitmapFromObj(*interp, tkwin, objPtr***)**
Pixmap
Tk_GetBitmap(*interp, tkwin, info***)**
Pixmap
Tk_GetBitmapFromObj(*tkwin, objPtr***)**
int
Tk_DefineBitmap(*interp, name, source, width, height***)**
char *
Tk_NameOfBitmap(*display, bitmap***)**
Tk_SizeOfBitmap(*display, bitmap, widthPtr, heightPtr***)**
Tk_FreeBitmapFromObj(*tkwin, objPtr***)**
Tk_FreeBitmap(*display, bitmap***)**

ARGUMENTS

Tcl_Interp ***interp*** *(in)*
: Interpreter to use for error reporting; if NULL then no error message is left after errors.

Tk_Window ***tkwin*** *(in)*
: Token for window in which the bitmap will be used.

Tcl_Obj ***objPtr*** *(in/out)*
: String value describes desired bitmap; internal rep will be modified to cache pointer to corresponding Pixmap.

CONST char ***info*** *(in)*
: Same as *objPtr* except description of bitmap is passed as a string and resulting Pixmap isn't cached.

CONST char ***name*** *(in)*
: Name for new bitmap to be defined.

char ***source*** *(in)*
: Data for bitmap, in standard bitmap format. Must be stored in static memory whose value will never change.

int ***width*** *(in)*
: Width of bitmap.

int ***height*** *(in)*
: Height of bitmap.

int ***widthPtr*** *(out)*
: Pointer to word to fill in with *bitmap*'s width.

int ***heightPtr*** *(out)*
: Pointer to word to fill in with *bitmap*'s height.

Display ***display*** *(in)*
: Display for which *bitmap* was allocated.

Pixmap ***bitmap*** *(in)*
: Identifier for a bitmap allocated by **Tk_AllocBitmapFromObj** or **Tk_GetBitmap**.

DESCRIPTION

These procedures manage a collection of bitmaps (one−plane pixmaps) being used by an application. The procedures allow bitmaps to be re−used efficiently, thereby avoiding server overhead, and also allow bitmaps to be named with character strings.

Tk_AllocBitmapFromObj returns a Pixmap identifier for a bitmap that matches the description in *objPtr* and is suitable for use in *tkwin*. It re−uses an existing bitmap, if possible, and creates a new one otherwise. *ObjPtr*'s value must have one of the following forms:

@*fileName*

FileName must be the name of a file containing a bitmap description in the standard X11 or X10 format.

name

Name must be the name of a bitmap defined previously with a call to **Tk_DefineBitmap**. The following names are pre−defined by Tk:

error

The international "don't" symbol: a circle with a diagonal line across it.

gray75

75% gray: a checkerboard pattern where three out of four bits are on.

gray50

50% gray: a checkerboard pattern where every other bit is on.

gray25

25% gray: a checkerboard pattern where one out of every four bits is on.

gray12

12.5% gray: a pattern where one−eighth of the bits are on, consisting of every fourth pixel in every other row.

hourglass

An hourglass symbol.

info

A large letter ``i".

questhead

The silhouette of a human head, with a question mark in it.

question

A large question−mark.

warning

A large exclamation point.

In addition, the following pre−defined names are available only on the **Macintosh** platform:

document

A generic document.

stationery

Document stationery.

edition

The *edition* symbol.

application

Generic application icon.

accessory
A desk accessory.

folder
Generic folder icon.

pfolder
A locked folder.

trash
A trash can.

floppy
A floppy disk.

ramdisk
A floppy disk with chip.

cdrom
A cd disk icon.

preferences
A folder with prefs symbol.

querydoc
A database document icon.

stop
A stop sign.

note
A face with ballon words.

caution
A triangle with an exclamation point.

Under normal conditions, **Tk_AllocBitmapFromObj** returns an identifier for the requested bitmap. If an error occurs in creating the bitmap, such as when *objPtr* refers to a non−existent file, then **None** is returned and an error message is left in *interp*'s result if *interp* isn't NULL. **Tk_AllocBitmapFromObj** caches information about the return value in *objPtr*, which speeds up future calls to procedures such as **Tk_AllocBitmapFromObj** and **Tk_GetBitmapFromObj**.

Tk_GetBitmap is identical to **Tk_AllocBitmapFromObj** except that the description of the bitmap is specified with a string instead of an object. This prevents **Tk_GetBitmap** from caching the return value, so **Tk_GetBitmap** is less efficient than **Tk_AllocBitmapFromObj**.

Tk_GetBitmapFromObj returns the token for an existing bitmap, given the window and description used to create the bitmap. **Tk_GetBitmapFromObj** doesn't actually create the bitmap; the bitmap must already have been created with a previous call to **Tk_AllocBitmapFromObj** or **Tk_GetBitmap**. The return value is cached in *objPtr*, which speeds up future calls to **Tk_GetBitmapFromObj** with the same *objPtr* and *tkwin*.

Tk_DefineBitmap associates a name with in−memory bitmap data so that the name can be used in later calls to **Tk_AllocBitmapFromObj** or **Tk_GetBitmap**. The *nameId* argument gives a name for the bitmap; it must not previously have been used in a call to **Tk_DefineBitmap**. The arguments *source*, *width*, and *height* describe the bitmap. **Tk_DefineBitmap** normally returns TCL_OK; if an error occurs (e.g. a bitmap named *nameId* has already been defined) then TCL_ERROR is returned and an error message is left in *interp−>result*. Note: **Tk_DefineBitmap** expects the memory pointed to by *source* to be static: **Tk_DefineBitmap** doesn't make a private copy of this memory, but uses the bytes pointed to by *source* later in calls to **Tk_AllocBitmapFromObj** or **Tk_GetBitmap**.

Typically **Tk_DefineBitmap** is used by **#include**−ing a bitmap file directly into a C program and then referencing the variables defined by the file. For example, suppose there exists a file **stip.bitmap**, which was created by the **bitmap** program and contains a stipple pattern. The following code uses **Tk_DefineBitmap** to define a new bitmap named **foo**:

```
Pixmap bitmap;
#include "stip.bitmap"
Tk_DefineBitmap(interp, "foo", stip_bits,
        stip_width, stip_height);
        ...
bitmap = Tk_GetBitmap(interp, tkwin, "foo");
```

This code causes the bitmap file to be read at compile−time and incorporates the bitmap information into the program's executable image. The same bitmap file could be read at run−time using **Tk_GetBitmap**:

```
Pixmap bitmap;
bitmap = Tk_GetBitmap(interp, tkwin, "@stip.bitmap");
```

The second form is a bit more flexible (the file could be modified after the program has been compiled, or a different string could be provided to read a different file), but it is a little slower and requires the bitmap file to exist separately from the program.

Tk maintains a database of all the bitmaps that are currently in use. Whenever possible, it will return an existing bitmap rather than creating a new one. When a bitmap is no longer used, Tk will release it automatically. This approach can substantially reduce server overhead, so **Tk_AllocBitmapFromObj** and **Tk_GetBitmap** should generally be used in preference to Xlib procedures like **XReadBitmapFile**.

The bitmaps returned by **Tk_AllocBitmapFromObj** and **Tk_GetBitmap** are shared, so callers should never modify them. If a bitmap must be modified dynamically, then it should be created by calling Xlib procedures such as **XReadBitmapFile** or **XCreatePixmap** directly.

The procedure **Tk_NameOfBitmap** is roughly the inverse of **Tk_GetBitmap**. Given an X Pixmap argument, it returns the textual description that was passed to **Tk_GetBitmap** when the bitmap was created. *Bitmap* must have been the return value from a previous call to **Tk_AllocBitmapFromObj** or **Tk_GetBitmap**.

Tk_SizeOfBitmap returns the dimensions of its *bitmap* argument in the words pointed to by the *widthPtr* and *heightPtr* arguments. As with **Tk_NameOfBitmap**, *bitmap* must have been created by **Tk_AllocBitmapFromObj** or **Tk_GetBitmap**.

When a bitmap is no longer needed, **Tk_FreeBitmapFromObj** or **Tk_FreeBitmap** should be called to release it. For **Tk_FreeBitmapFromObj** the bitmap to release is specified with the same information used to create it; for **Tk_FreeBitmap** the bitmap to release is specified with its Pixmap token. There should be exactly one call to **Tk_FreeBitmapFromObj** or **Tk_FreeBitmap** for each call to **Tk_AllocBitmapFromObj** or **Tk_GetBitmap**.

BUGS

In determining whether an existing bitmap can be used to satisfy a new request, **Tk_AllocBitmapFromObj** and **Tk_GetBitmap** consider only the immediate value of the string description. For example, when a file name is passed to **Tk_GetBitmap**, **Tk_GetBitmap** will assume it is safe to re−use an existing bitmap created from the same file name: it will not check to see whether the file itself has changed, or whether the current directory has changed, thereby causing the name to refer to a different file.

KEYWORDS

bitmap, pixmap

Tk_GetPixelsFromObj, Tk_GetPixels, Tk_GetMMFromObj, Tk_GetScreenMM − translate between strings and screen units

SYNOPSIS

#include <tk.h>
int
Tk_GetPixelsFromObj(*interp, tkwin, objPtr, intPtr***)**
int
Tk_GetPixels(*interp, tkwin, string, intPtr***)**
int
Tk_GetMMFromObj(*interp, tkwin, objPtr, doublePtr***)**
int
Tk_GetScreenMM(*interp, tkwin, string, doublePtr***)**

ARGUMENTS

Tcl_Interp ****interp*** *(in)*
: Interpreter to use for error reporting.

Tk_Window ***tkwin*** *(in)*
: Window whose screen geometry determines the conversion between absolute units and pixels.

Tcl_Obj ****objPtr*** *(in/out)*
: String value specifies a distance on the screen; internal rep will be modified to cache converted distance.

char ****string*** *(in)*
: Same as *objPtr* except specification of distance is passed as a string.

int ****intPtr*** *(out)*
: Pointer to location in which to store converted distance in pixels.

double ****doublePtr*** *(out)*
: Pointer to location in which to store converted distance in millimeters.

DESCRIPTION

These procedures take as argument a specification of distance on the screen (*objPtr* or *string*) and compute the corresponding distance either in integer pixels or floating−point millimeters. In either case, *objPtr* or *string* specifies a screen distance as a floating−point number followed by one of the following characters that indicates units:

<none>
: The number specifies a distance in pixels.

c
: The number specifies a distance in centimeters on the screen.

i
: The number specifies a distance in inches on the screen.

m
: The number specifies a distance in millimeters on the screen.

p
: The number specifies a distance in printer's points (1/72 inch) on the screen.

Tk_GetPixelsFromObj converts the value of *objPtr* to the nearest even number of pixels and stores that value at **intPtr*. It returns **TCL_OK** under normal circumstances. If an error occurs (e.g. *objPtr* contains a number followed by a character that isn't one of the ones above) then **TCL_ERROR** is returned and an error message is left in *interp*'s result if *interp* isn't NULL. **Tk_GetPixelsFromObj** caches information about the return value in *objPtr*, which speeds up future calls to **Tk_GetPixelsFromObj** with the same *objPtr*.

Tk_GetPixels is identical to **Tk_GetPixelsFromObj** except that the screen distance is specified with a string instead of an object. This prevents **Tk_GetPixels** from caching the return value, so **Tk_GetAnchor** is less efficient than **Tk_GetPixelsFromObj**.

Tk_GetMMFromObj and **Tk_GetScreenMM** are similar to **Tk_GetPixelsFromObj** and **Tk_GetPixels** (respectively) except that they convert the screen distance to millimeters and store a double−precision floating−point result at **doublePtr*.

KEYWORDS

centimeters, convert, inches, millimeters, pixels, points, screen units

Appendix 3: Keywords

A

anchor
ConfigWidg

B

balloon
tixBalloon
bitmap
ConfigWidg, GetBitmap
boolean
ConfigWidg
border
ConfigWidg
button box
tixStdButtonBox

C

cap style
ConfigWidg
centimeters
GetPixels
choice
tixSelect
class
options
color
ConfigWidg
ComboBox
tixComboBox
compound
compound
configuration options
ConfigWidg
container
tixScrolledWindow, tixSelect, tixStdButtonBox
Container Widget
tixPanedWindow
container widget
tixButtonBox
context−sensitive help

tixBalloon
convert
GetPixels
cursor
ConfigWidg
custom
ConfigWidg

D

dialog
tixDirSelectDialog, tixExFileSelectDialog, tixFileSelectDialog
directory list
tixDirList
directory selector
tixDirSelectDialog
directory tree
tixDirTree
display item
tixDisplayStyle
display style
tixDisplayStyle
double
ConfigWidg

F

file entry
tixFileEntry
file selection dialog
tix
file selector
tixExFileSelectBox, tixExFileSelectDialog, tixFileSelectBox, tixFileSelectDialog
font
ConfigWidg
form
tixForm
frame
frame, tixScrolledWindow

G

geometry management
tixForm
grab
tixUtils
grid
tixGrid

H

height
image
hierarchical listbox
tixCheckList, tixHList, tixScrolledHList, tixTree

I

image
image, compound, pixmap
imagetext
tixDisplayStyle
inches
GetPixels
input only
tixInputOnly
integer
ConfigWidg
invisible
tixInputOnly

J

join style
ConfigWidg
justify
ConfigWidg

L

label entry
tixLabelEntry
label frame
tixLabelFrame
listbox
tixComboBox, tixScrolledListBox

M

mega widgets
TixIntro
meter
tixMeter
millimeters
ConfigWidg, GetPixels
Motif window manager
tixMwm
mwm
tixMwm

N

name
options
notebook
tixListNoteBook, tixNBFrame, tixNoteBook

O

Object
tixDestroy
option menu
tixOptionMenu

P

pixels
ConfigWidg, GetPixels
pixmap
GetBitmap, pixmap
points
GetPixels
popup menu
tixPopupMenu
progress
tixMeter

R

relief
ConfigWidg

S

screen units
GetPixels
scroll
tixScrolledHList, tixScrolledListBox, tixScrolledText, tixScrolledWindow
shell
tixwish
spinbox
tixControl
spread sheet
tixGrid
standard option
options
switch
options
synonym
ConfigWidg

T

table
tixGrid
tabular listbox
tixTList
text
tixScrolledText
Tix
TixIntro, tixBalloon, tixDestroy
TIX
tixPanedWindow
Tk
tixwish
toolkit
tixwish
tree
tixCheckList, tixTree
types of images
image

U

uid
ConfigWidg

W

widget
frame, tixBalloon, tixButtonBox, tixCheckList, tixComboBox, tixControl, tixDirList, tixDirSelectDialog, tixDirTree, tixExFileSelectBox, tixExFileSelectDialog, tixFileEntry, tixFileSelectBox, tixFileSelectDialog, tixHList, tixInputOnly, tixLabelEntry, tixLabelFrame, tixListNoteBook, tixMeter, tixNBFrame, tixNoteBook, tixOptionMenu, tixPopupMenu, tixScrolledHList, tixScrolledListBox, tixScrolledText, tixScrolledWindow, tixSelect, tixStdButtonBox, tixTList, tixTree
width
image
window
tixScrolledWindow
wish
tixwish

X

XPM
pixmap