Table of Contents

Practical Programming in Tcl and Tk, Fourth Edition By Ken Jones, Jeffrey Hobbs, Brent B. Welch ISBN:Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
0-13-038560-3 Publisher: Prentice Hall
Print Publication Date: 6/10/2003 User number: 628024 Copyright 2007, Safari Books Online, LLC.

Chapter 23. Tk Fundamentals

This chapter introduces the basic concepts used in the Tk graphical user interface toolkit. Tk adds about 45 Tcl commands that let you create and manipulate widgets in a graphical user interface. Tk works with the X window system, Windows, and Macintosh. The same script can run unchanged on all of these major platforms.

Tk is a toolkit for programming graphical user interfaces. It was designed for the X window system used on UNIX systems, and it was ported later to the Macintosh and Windows environments. Tk shares many concepts with other windowing toolkits, but you do not need to know much about graphical user interfaces to get started with Tk.

Tk provides a set of Tcl commands that create and manipulate *widgets*. A widget is a window in a graphical user interface that has a particular appearance and behavior. The terms *widget* and *window* are often used interchangeably. Widget types include buttons, scrollbars, menus, and text windows. Tk also has a general-purpose drawing widget called a *canvas* that lets you create lighter-weight items such as lines, boxes, and bitmaps. The canvas is extremely powerful, yet very easy to use. The Tcl commands added by Tk are summarized at the end of this chapter.

Tk widgets are organized in a hierarchy. To an application, the window hierarchy means that there is a primary window, and inside that window there can be a number of children windows. The children windows can contain more windows, and so on. Just as a hierarchical file system has directories (i.e., folders) that are containers for files and directories, a hierarchical window system uses windows as containers for other windows. The hierarchy affects the naming scheme used for Tk widgets as described later, and it is used to help arrange widgets on the screen.

Widgets are under the control of a *geometry manager* that controls their size and location on the screen. Until a geometry manager learns about a widget, it will not be mapped onto the screen and you will not see it. Tk has powerful geometry managers that make it very easy to

Practical Programming in Tcl and Tk, Fourth Edition By Ken Jones, Jeffrey Hobbs, Brent B. Welch ISBN:Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
0-13-038560-3 Publisher: Prentice Hall
Print Publication Date: 6/10/2003

create nice screen layouts. The main trick with any geometry manager is that you use *frame* widgets as containers for other widgets. One or more widgets are created and then arranged in a frame by a geometry manager. By putting frames within frames you can create complex layouts. There are three different geometry managers you can use in Tk: `grid`, `pack`, and `place`, and one widget, the panedwindow, that also acts as a geometry manager. The Tk geometry managers are discussed in detail in Chapters 25, 26, and 27; the panedwindow is discussed in Chapter 28.

A Tk-based application has an *event-driven* control flow, like most window system toolkits. The Tk widgets handle most events automatically, so programming your application remains simple. For specialized behaviors, you use the `bind` command to register a Tcl command that runs when an event occurs. There are lots of events, including mouse motion, keystrokes, window resize, and window destruction. You can also define *virtual events*, like `Cut` and `Paste`, that are caused by different events on different platforms. Bindings are discussed in detail in Chapter 29. Chapter 16 describes I/O events and the Tcl event loop, while Chapter 50 describes C programming and the event loop.

Event bindings are grouped into classes, which are called *bindtags*. The `bindtags` command associates a widget with an ordered set of bindtags. The level of indirection between the event bindings and the widgets creates a flexible and powerful system for managing events. You can create your own bindtags and dynamically change the bindtags for a widget to support mode changes in your application.

A concept related to binding is *focus*. At any given time, one of the widgets has the input focus, and keyboard events are directed to it. There are two general approaches to focusing: give focus to the widget under the mouse, or explicitly set the focus to a particular widget. Tk provides commands to change focus so you can implement either style of focus management. To support modal dialog boxes, you can forcibly *grab* the focus away from other widgets. Chapter 39 describes focus, grabs, and dialogs.

The basic structure of a Tk script begins by creating widgets and arranging them with a geometry manager, and then binding actions to the widgets. After the interpreter processes the commands that initialize the user interface, the event loop is entered and your application begins running.

If you use *wish* interactively, it creates and displays an empty main window and gives you a command-line prompt. With this interface, your keyboard commands are handled by the event loop, so you can build your Tk interface gradually. As we will see, you will be able to change virtually all aspects of your application interactively.

Hello, World! in Tk

Our first Tk script is very simple. It creates a button that prints "Hello, World!" to standard output when you press it. Above the button widget is a title bar that is provided by the window manager, which in this case is *twm* under X windows:

Example 23-1. "Hello, World!" Tk program


```
#!/usr/local/bin/wish
button .hello -text Hello \
   -command {puts stdout "Hello, World!"}
pack .hello -padx 20 -pady 10
```

The first line identifies the interpreter for the script:

```
#!/usr/local/bin/wish
```

This special line is necessary if the script is in a file that will be used like other UNIX command files. Chapter 2 describes how to set up scripts on different platforms.

There are two Tcl commands in the script: one to create the button, and one to make it visible on the display. The `button` command creates an instance of a button:

```
button .hello -text Hello \
    -command {puts stdout "Hello, World!"}
=> .hello
```

The name of the button is `.hello`. The label on the button is `Hello`, and the command associated with the button is:

```
puts stdout "Hello, World!"
```

The `pack` command maps the button onto the screen. Some padding parameters are supplied, so there is space around the button:

Practical Programming in Tcl and Tk, Fourth Edition By Ken Jones, Jeffrey Hobbs, Brent B. Welch ISBN:Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
0-13-038560-3 Publisher: Prentice Hall
Print Publication Date: 6/10/2003 User number: 628024 Copyright 2007, Safari Books Online, LLC.

```
pack .hello -padx 20 -pady 10
```

If you type these two commands into `wish`, you will not see anything happen when the `button` command is given. After the `pack` command, though, you will see the empty main window shrink to be just big enough to contain the button and its padding. The behavior of the packer will be discussed further in Chapters 24 and 25.

Tk uses an object-based system for creating and naming widgets. Associated with each class of widget (e.g., `Button`) is a command that creates instances of that class of widget. As the widget is created, a new Tcl command is defined that operates on that instance of the widget. Example 23-1 creates a button named `.hello`, and we can operate on the button using its name as a Tcl command. For example, we can cause the button to highlight a few times:

```
.hello flash
```

Or we can run the command associated with the button:

```
.hello invoke
=> Hello, World!
```

Tk has widget classes and instances, but it is not fully object oriented. It is not possible to subclass a widget class and use inheritance. Instead, Tk provides very flexible widgets that can be configured in many different ways to tune their appearance. The resource database can store configuration information that is shared by many widgets, and new classes can be introduced to group resources. Widget behavior is shared by using binding tags that group bindings. Instead of building class hierarchies, Tk uses composition to assemble widgets with shared behavior and attributes.

Naming Tk Widgets

The period in the name of the button instance, `.hello`, is required. Tk uses a naming system for the widgets that reflects their position in a hierarchy of widgets. The root of the hierarchy is the main window of the application, and its name is simply a dot (i.e., `.`). This is similar to the naming convention for directories in UNIX where the root directory is named `/`, and then `/` is used to separate components of a file name. Tk uses a dot in the same way. Each widget that is a child of the main window is named something like `.foo`. A child widget of `.foo` would be `.foo.bar`, and so on. Just as file systems have directories that are containers for files and other directories, the Tk window hierarchy uses frame widgets that are containers for widgets and other frames.

Each component of a Tk pathname must start with a lowercase letter or a number. Obviously, a component cannot include a period, either. The lower case restriction avoids a conflict with resource class names that begin with an upper case letter. A resource name can include Tk pathname components and Tk widget classes, and case is used to distinguish them. Chapter 31 describes resources in detail.

Store widget names in variables.

There is one drawback to the Tk widget naming system. If your interface changes enough it can result in some widgets changing their position in the widget hierarchy. In that case they may need to change their name. You can insulate yourself from this programming nuisance by using variables to hold the names of important widgets. Use a variable reference instead of widget pathnames in case you need to change things, or if you want to reuse your code in a different interface. The widget creating commands return the name of the widget:

```
set b [button .hello -text "Hello" -command {puts "Hello!"}]
```

You use `$b` as a command to operate on the button:

```
$b configure -background green
```

Configuring Tk Widgets

Example 23-1 illustrates a style of named parameter passing that is prevalent in the Tk commands. Pairs of arguments specify the attributes of a widget. The attribute names begin with –, such as `-text`, and the next argument is the value of that attribute. Even the simplest Tk widget can have a dozen or more attributes that can be specified this way, and complex widgets can have 30 or more attributes. However, the beauty of Tk is that you need to specify only the attributes for which the default value is not good enough. This is illustrated by the simplicity of the `Hello, World` example.

Practical Programming in Tcl and Tk, Fourth Edition By Ken Jones, Jeffrey Hobbs, Brent B. Welch ISBN:Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
0-13-038560-3 Publisher: Prentice Hall
Print Publication Date: 6/10/2003 User number: 628024 Copyright 2007, Safari Books Online, LLC.

Finally, each widget instance supports a `configure` operation, which can be abbreviated to `config`, that can query and change these attributes. The syntax for `config` uses the same named argument pairs used when you create the widget. For example, we can change the background color of the button to red even after it has been created and mapped onto the screen:

```
.hello config -background red
```

Widget attributes can be redefined any time, even the `text` and `command` that were set when the button was created. The following command changes `.hello` into a goodbye button:

```
.hello config -text Goodbye! -command exit
```

Widgets have a `cget` operation to query the current value of an attribute:

```
.hello cget -background
=> red
```

You can find out more details about a widget attribute by using `configure` without a value:

```
.hello config -background
=> -background background Background #ffe4c4 red
```

The returned information includes the command-line switch, the resource name, the class name, the default value, and the current value, which is last. The class and resource name have to do with the resource mechanism described in Chapter 31. If you only specify `configure` and no attribute, then a list of the configuration information for all widget attributes is returned. Example 23-2 uses this to print out all the information about a widget:

Example 23-2. Looking at all widget attributes

```
proc Widget_Attributes {w {out stdout}} {
   puts $out [format "%-20s %-10s %s" Attribute Default Value]
   foreach item [$w configure] {
      puts $out [format "%-20s %-10s %s" \
         [lindex $item 0] [lindex $item 3] \
         [lindex $item 4]]
   }
}
```

Tk Widget Attributes and the Resource Database

A widget attribute can be named three different ways: by its command-line option, by its resource name, and by its resource class. The command-line option is the format you use in Tcl scripts. This form is always all lowercase and prefixed with a hyphen (e.g., `-offvalue`). The resource name for the attribute has no leading hyphen, and it has uppercase letters at internal word boundaries (e.g., `offValue`). The resource class begins with an uppercase letter and has uppercase letters at internal word boundaries. (e.g., `OffValue`).

The tables in this book list widget attributes by their resource name.

You need to know these naming conventions if you specify widget attributes via the resource mechanism. The command-line option can be derived from the resource name by mapping it to all lowercase. The primary advantage of using resources to specify attributes is that you do not have to litter your code with attribute specifications. With just a few resource database entries you can specify attributes for all your widgets. In addition, if attributes are specified with resources, users can provide alternate resource specifications in order to override the values supplied by the application. For attributes like colors and fonts, this feature can be important to users. Resource specifications are described in detail in Chapter 31.

The Tk Manual Pages

This book provides summaries for all the Tk commands, the widget attributes, and the default bindings. However, for the absolute truth, you may need to read the on-line manual pages that come with Tk. They provide a complete reference source for the Tk commands. You should be able to use the UNIX *man* program to read them:

```
% man button
```

The *tkman* program provides a very nice graphical user interface to the UNIX manual pages. On the Macintosh platform, the manual pages are formatted into HTML documents that you

can find in the HTML Docs folder of the Tcl/Tk distribution. On Windows, the manual pages are formatted into Help documents. You can find the manual pages on the web at:

http://www.tcl.tk/man/

There are a large number of attributes that are common across most of the Tk widgets. These are described in a separate man page under the name options. Each man page begins with a STANDARD OPTIONS section that lists which of these standard attributes apply, but you have to look at the options man page for the description. In contrast, the tables in this book always list all widget attributes.

Summary of the Tk Commands

The following tables list the Tcl commands added by Tk. The page number in the table is the primary reference for the command, and there are other references in the index.

Widget Commands

Table 23-1 lists commands that create widgets. There are 18 different widgets in Tk, although 4 of them are variations on a button, and 5 are devoted to different flavors of text display.

Table 23-1. Tk widget-creation commands

Command	Pg.	Description
button	454	Create a command button.
canvas	557	Create a canvas, which supports lines, boxes, bitmaps, images, arcs, text, polygons, and embedded widgets.
checkbutton	458	Create a toggle button that is linked to a Tcl variable.

Command	Pg.	Description
entry	507	Create a one-line text entry widget.
frame	485	Create a container widget used with geometry managers.
label	490	Create a read-only, multiline text label.
labelframe	485	Create a container widget used with geometry managers that has extra label attributes. (Tk 8.4)
listbox	519	Create a line-oriented, scrolling text widget.
menu	462	Create a menu.
menubutton	462	Create a button that posts a menu.
message	493	Create a read-only, multiline text message.
panedwindow	429	Create a container widget that controls other widgets in a paned fashion. (Tk 8.4)
radiobutton	458	Create one of a set of radio buttons linked to one variable.
scale	495	Create a scale widget that adjusts the value of a variable.
scrollbar	499	Create a scrollbar that can be linked to another widget.
spinbox	511	Create a spinbox widget that is a composite entry widget with button controls for adjusting the value. (Tk 8.4)
text	531	Create a general-purpose, editable text widget.

Command	Pg.	Description
toplevel	485	Create a frame that is a new top level window.

Widget Manipulation Commands

Table 23-2 lists commands that manipulate widgets and provide associated functions like input focus, event binding, and geometry management.

Table 23-2. Tk widget-manipulation commands

Command	Pg.	Description
bell	497	Ring the terminal bell device.
bind	435	Bind a Tcl command to an event.
bindtags	437	Create binding classes and control binding inheritance.
clipboard	594	Manipulate the clipboard .
destroy	605	Delete a widget.
event	446	Define and generate virtual events.
focus	603	Control the input focus.
font	641	Set and query font attributes and measurements.
grab	604	Steal the input focus from other widgets.
grid	419	Arrange widgets into a grid with constraints.

Command	Pg.	Description
image	626	Create and manipulate images.
lower	409	Lower a window in the stacking order.
option	477	Set and query the resources database.
pack	409	Pack a widget in the display with constraints.
place	427	Place a widget in the display with positions.
raise	409	Raise a window in the stacking order.
selection	593	Manipulate the selection.
send	648	Send a Tcl command to another Tk application.
tk	669	Query or set the application name or global caret.
tkerror	202	Handler for background errors.
tkwait	605	Wait for an event.
update	608	Update the display by going through the event loop.
winfo	663	Query window state.
wm	657	Interact with the window manager.

Practical Programming in Tcl and Tk, Fourth Edition By Ken Jones, Jeffrey Hobbs, Brent B. Welch ISBN:Prepared for Ronald Fischer, Safari ID: ronald.fischer@fusshuhn.de
0-13-038560-3 Publisher: Prentice Hall
Print Publication Date: 6/10/2003 User number: 628024 Copyright 2007, Safari Books Online, LLC.

Support Procedures

Table 23-3 lists several support procedures that implement standard dialogs, option menus, and other facilities.

Table 23-3. Tk support procedures

Command	Pg.	Description
tk_bisque	621	Install bisque family of colors.
tk_chooseColor	602	Dialog to select a color. (Tk 4.2)
tk_chooseDirectory	600	Dialog to select a directory. (Tk 8.2)
tk_dialog	599	Create simple dialogs.
tk_focusFollowsMouse	603	Install mouse-tracking focus model.
tk_focusNext	604	Focus on next widget in tab order.
tk_focusPrev	604	Focus on previous widget in tab order.
tk_getOpenFile	600	Dialog to open an existing file. (Tk 4.2)
tk_getSaveFile	600	Dialog to open a new file. (Tk 4.2)
tk_messageBox	600	Message dialog. (Tk 4.2)
tk_optionMenu	465	Create an option menu.
tk_popup	465	Create a pop-up menu.

Command	Pg.	Description
tk_setPalette	621	Set the standard color palette. (Tk 4.2)

Other Widget Sets

This book describes the set of widgets provided by core Tk distribution. There are number of other widget sets for Tk. Some are implemented as Tcl procedures that compose the basic widgets into useful combinations (e.g., BWidgets). Others are C-based toolkits (e.g., Tix and BLT). A few of the more popular widget sets are listed here:

BLT

George Howlett created BLT. It includes a great graph widget that efficiently supports large datasets. It also includes a tabbed notebook and tree view widget. Its busy widget covers your application with a transparent widget that just displays a watch cursor, which is handy when the application is busy doing something and you don't want to accept mouse clicks. This is a C-based toolkit.

http://www.sourceforge.net/projects/blt/

Tix

Tix was created by Ioi Lam, and is now supported by a team of volunteers. It includes several widgets and an infrastructure for creating new widgets in Tcl. Notable features include balloon help, tabbed windows, paned window, and a hierarchy browser. This is a C-based toolkit, although it includes a number of compound widgets created in Tcl.

http://tix.sourceforge.net/

[incr Tk] and [incr Widgets]

[incr Tk] is a C-based framework for creating compound widgets using the [incr Tcl] object system. [incr Widgets] is the widget set created using that framework. It includes loads of widgets, from simple labeled-entry widgets up through HTML display widgets. These tools are described in Chad Smith's book, *[incr Tcl] from the Ground Up* (Osborne-McGraw Hill, 1999).

http://incrtcl.sourceforge.net

BWidgets

BWidgets is a set of Tcl-based widgets. It includes a variety of compound widgets, including a tabbed notebook, combobox, and hierarchy browser. It is hosted at the Standard Tcl Lib (tcllib) web site:

http://www.sourceforge.net/projects/tcllib

TkTable

TkTable is combination of a gridding geometry manager and several text-oriented widgets. It makes it easy to lay out tabular data like spreadsheets, and it also provides a large amount of control over the formatting of cells and their data.

http://www.sourceforge.net/projects/tktable