# ;login:

THE MAGAZINE OF USENIX & SAGE

April 2002 • Volume 27 • Number 2

inside:

**PROGRAMMING**

The Tclsh Spot
BY CLIF FLYNT

USENIX & SAGE

The Advanced Computing Systems Association &
The System Administrators Guild

# the tclsh spot

**by Clif Flynt**

Clif Flynt is president of Noumena Corp., which offers training and consulting services for Tcl/Tk and Internet applications. He is the author of *Tcl/Tk for Real Programmers* and the *TclTutor* instruction package. He has been programming computers since 1970 and a Tcl advocate since 1994.

*clif@cflynt.com*

This article starts a short series describing Tk techniques to make applications more professional looking.

When I'm doing a quick and dirty Tk GUI, I usually just use pack or grid to display the widgets in a usable manner. I don't worry about resizing the windows, providing hints, or any of the other features that I consider minimal requirements in a commercial package.

As soon as this package gets shown to a client, my lapses become obvious and embarrassing.

These features are actually pretty easy to put into a Tk application, and the nice thing is that they can be added after the GUI layout has been tuned. This helps keep the code small and manageable during the main rework stage of development.

In this article, I'll describe how to configure the geometry managers and widgets to allow individual widgets to be resized when the main window is resized.

Tk uses a paradigm similar to other windowing systems in that generating and displaying graphic objects are two separate events. The graphics objects are created with Tk widget commands like label, button, entry, text, and canvas. Mapping the objects onto the display is handled with one of the geometry managers – place, pack, and grid (and, of course, whatever the system window manager is).

Tk's three geometry managers provide different levels of abstraction in defining how a display should appear. The place command gives you the most control and requires the most thought to use. The pack and grid commands let your script describe the layout in more abstract terms, and the geometry manager will define the exact positions of the widgets.

The place command takes arguments that describe the location of each widget to be displayed and the size of the widget. The syntax is:

Syntax: place *.widgetName ?-option1 value1? ?-option2 value2? ...*

| | |
|---|---|
| place | Place a widget at a particular location in a frame. |
| *.widgetName* | The name of the widget to be placed. |
| *-option value* | Key/value pairs that define the behavior of the widget being placed. |

| | |
|---|---|
| -x/-y pixels | Describe the location of the widget in pixels, numbering from the upper-left-hand corner. |
| -relx/-rely float | Describe the location of the widget as a fraction of the size of the window. 0.0 is the left (for -relx) or top (for -rely) edge, and 1.0 is the right or bottom. |
| -anchor location | Tells whether the widget should be anchored on the north, south, east or west edge (nsew), or center. By default, this is nw, the upper-left corner. |
| -width/-height size | An integer that defines the size of the widget in pixels. By default, this is the natural size for the widget. |
| -relwidth/-relheight float | Defines what fraction of the parent window should be devoted to this widget. |

Using the place command, it's fairly easy to make a display that would have two text widgets, and corresponding labels above them, and have the widgets expand to be wider when you resize the window:

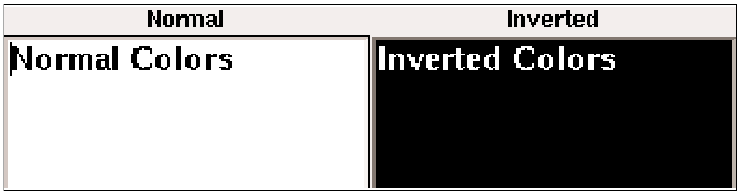
```
label .l1 -text "Normal"
label .l2 -text "Inverted"

text .t1 -background white -foreground black -font "helvetica 16 bold'
text .t2 -background black -foreground white -font "helvetica 16 bold"

place .l1 -x 0 -y 0 -relwidth .5 -height 15
place .l2 -relx .5 -y 0 -relwidth .5 -height 15

place .t1 -x 0 -y 15 -relwidth .5 -relheight .99
place .t2 -relx .5 -y 15 -relwidth .5 -relheight .99
```

The display for this code would resemble the window below. When the window is resized, the text widgets would expand or shrink, while the labels would stay 15 pixels tall.



For simple displays like this, or free form displays (perhaps a simulation of a car instrument panel), place works well. However, if you change the font in the labels, you'd have to change the height of the labels and the upper location of the text widgets. If the display were complex, tuning the appearance could become very tedious.

The pack command doesn't support locating widgets at specific coordinates. The pack command allows you to declare that a widget should be as close as possible to one edge or another of the window it's contained in.

Syntax: pack *.windowName ?-option1 value1? ?-option2 value2?*

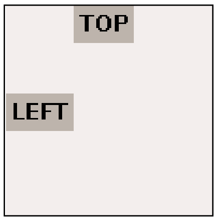
The options for pack include:

| | |
|---|---|
| *-expand boolean* | If this is true (true, 1 or Y), then leftover space after the windows have been placed will be given to this window. If multiple windows are true for being expanded, the space will be divided evenly. |
| *-fill x/y/none/both* | If the -expand option is true, this tells the geometry manager that this window should expand in the x, y, or both dimensions. By default the value is none. |
| -anchor location | Tells whether the widget should be anchored on the north, south, east, or west edge (n, s, e, w), or center of the available space. By default, this is center. |
| *-side left/top/right/bottom* | Tells the packer which side of the open space to place this widget against. |

The packer can be conceptualized as starting with a large open square. As it receives windows to pack, it places them on the requested edge of the remaining open space.

For example, if the first window is a label with the -side left option set, pack would place the left edge of the label against the left side of the empty frame. The left edge of the empty space is now the right edge of the label, even though there may be empty space above and below this widget.

If the next item is to be packed -side top, it will be placed above and to the right of the first widget.



The following code shows how these two widgets would appear. Note that even though the anchor on the top label is set to west, it only goes as far to the west as the east-most edge of the first widget packed.

```
label .l1 -background gray50 -text LEFT
label .l2 -background gray50 -text TOP
pack .l1 -side left
pack .l2 -side top -anchor w
```

This makes it impossible to build a display like the one we built with place from windows that are first-level children of the primary window.

The trick to building rectangular GUIs with the place command is to use multiple frames.

A frame is a (mostly) invisible holder window that can be used to group other windows.

Syntax: frame .frameName ?-option value?

The options supported by the frame command include:

| | |
|---|---|
| -height *numPixels* | Height in pixels. |
| -width *numPixels* | Width in pixels. |
| -background *color* | The color of the background. |
| -relief ?sunken?raised? | Defines how to draw the border edges – can make the frame look raised or sunken. The default is to not display relief borders. |
| -borderwidth *width* | Sets the width of the decorative borders. |

Building a complex GUI with pack and frame can get confusing. When the display is not what you expected, and you can't quite figure out why, it's sometimes useful to use the -background or -relief options to make it obvious which frame is where.

Simple GUIs, however, are fairly easy with pack. For instance, to build the same GUI as was done with place, we could use two frames. One frame would hold the left-most label and text widget, while the other would hold the right-most label and text widget.

```
frame .left
frame .right

label .left.label -text Normal
label .right.label -text Inverted

text .left.text -background white -foreground black \
      -font "helvetica 16 bold" -width 5 -height 1
text .right.text -background black -foreground white \
      -font "helvetica 16 bold" -width 5 -height 1

pack .left.label -side top
pack .right.label -side top
pack .left.text -side top -expand y -fill both
pack .right.text -side top -expand y -fill both

pack .left -side left -expand y -fill both
pack .right -side left -expand y -fill both
```

One difference between the pack and place command is that with the place command, we could specify the percentage of the main window a widget could use, and Tk would scale the widget accordingly. With pack, Tk places the windows in their default sizes, and then expands them where it can.

The default size for a text widget is 80 columns by 23 lines – like an old-style monitor, or a default-sized x-term window. In order to allow pack to use small text windows, we need to define them with a small initial size, and let the pack algorithm expand them.

One advantage of this code over place is that we can change the size of the widgets (like defining a larger font in the labels) without having to recalculate the locations of the widgets.

The pack command is very useful for asymmetric layouts or simple GUIs. Using pack to arrange buttons in a frame is the easiest way to display a row of buttons.

However, it's difficult to make a gridded display like a spreadsheet with the pack command, and that's actually a very common pattern.

The third geometry manager is the grid command, which is oriented around a gridded, spreadsheet-like layout.

The grid command arranges the widgets in the rows and columns your script defines, and expands or shrinks the row/column to fit the largest widget in that row/column.

Like the pack command, the grid command can expand a widget to fit a cell, but it does not shrink them below their requested size.

Syntax: grid *widgetName -option1 value1 ?-option2 value2?* ...

The *grid* options include:

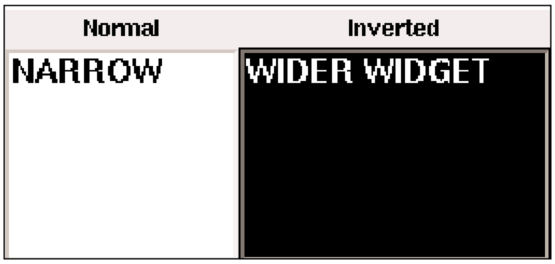
| | |
|---|---|
| -column *columnNumber* | The column position for this widget. |
| -row *rowNumber* | The row for this widget. |
| -columnspan *columnsToUse* | How many columns to use for this widget. Defaults to 1. |
| -rowspan *rowsToUse* | How many rows to use for this widget. Defaults to 1. |
| -sticky *side* | Which edge of the cell this widget should "stick" to. Values may be n, s, e, w, or a combination of sides. |

One shortcoming of the pack command is that it distributes extra space evenly between the windows that are allowed to expand. With the place command, you could define one widget to expand at a different rate from another.

Having widgets grow at different rates can be done with the grid command. The grid columnconfigure and grid rowconfigure commands let you define the behavior of a row or column in the display. There are a number of options including:

| | |
|---|---|
| *-minsize pixels* | The minimum size for this widget, in pixels. |
| *-weight integer* | The weighting to assign to this column/row. The default value is 0; never change the size of the widget. |

| | |
|---|---|
| *-pad pixels* | A number of pixels to use as padding around widgets in this column/row. |



This code will create a pair of text widgets and labels, as we did above, but will make the right-hand side half again as wide as the left.

```
label .l1 -text "Normal"
label .l2 -text "Inverted"

text .t1 -background white -foreground black \
  -font "helvetica 16 bold" -width 5 -height 1

text .t2 -background black -foreground white \
  -font "helvetica 16 bold" -width 5 -height 1

grid .l1 -row 0 -column 0 -sticky news
grid .l2 -row 0 -column 1 -sticky news

grid .t1 -row 1 -column 0 -sticky news
grid .t2 -row 1 -column 1 -sticky news

grid columnconfigure . 0 -weight 2
grid columnconfigure . 1 -weight 3
grid rowconfigure . 1 -weight 1
```

To summarize the features of the three geometry managers:

*place*

- Complete control of widget location.
- Can overlay one widget on top of another.
- Script must define all details.
- Individual widgets can be resized.
- Can resize widgets at different rates.

*pack*

- Supports a space-filling abstraction for locating widgets.
- Widgets will not overlap each other; all widgets will be displayed.
- Individual widgets can be resized.
- Available space is apportioned to widgets evenly.

*grid*

- Defines widget layout as a grid.
- Widgets will not overlap each other; all widgets will be displayed.
- All widgets in a row or column can be resized.
- Rows and columns can resize at different rates.

In these examples, the windows were always wide enough to display all of the widgets. We added support for someone expanding a main GUI and expanding the appropriate widgets to look right, but not to shrink the display below the default minimum.

What do you do if there might be more elements to display than you've got space for on the screen? For instance, someone might want to run your application on the IBM Linux watch with an LCD that isn't wide enough to display a full button bar.

The obvious answer to this is to put a scrollbar on the edges of the primary window, so the user can scroll to the widgets they want to see.

The next Tclsh Spot will describe some tricks for making scrolled windows.