;login:

THE MAGAZINE OF USENIX & SAGE

June 2002 volume 27 • number 3

inside:

PROGRAMMING

THE TCLISH 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

There seems to be a universal rule that no matter how large a container is, what you need to put in it is larger.

This applies to screens and graphic displays just as much as it applies to my bookshelves. No matter how large the display, there will be an application that needs to display more information than you can fit on it.

The GUI solution to this problem is the scrollbar, which lets us create an image that's larger than the viewing area, and then move the viewable window to the subset of the image we're interested in. (If someone develops a scrollbar to put on bookcases they'll make a fortune.)

Syntax: scrollbar *scrollbarName ?options?*

scrollbar		Create a scrollbar widget.
scrollbarName		The name for this scrollbar.
options		This widget supports several options. The -command option is required.
	-command *"procName ?args?"*	This defines the command to change the state of the scrollbar. Arguments that define the changed state will be appended to the arguments defined in this option
	-orient *direction*	Defines the orientation for the scrollbar. The *direction* may be horizontal or vertical. Defaults to vertical.
	troughcolor *color*	Defines the color for the trough below the slider. Defaults to the default background color of the frames.

A scrollbar interacts with a Tk widget via callback procedures registered with the scrollbar and the associated widget. When the widget changes configuration (for example, more text is added to a text widget), it evaluates a script to update the scrollbar. When the scrollbar is modified (a user moves a slider), it evaluates a script that will update the appropriate widget.

Several Tk widgets (listbox, entry, text, and canvas) have built in support for a scrollbar. Each of these widgets supports a yview and/or xview widget command that moves the viewable window, and can be invoked by a scrollbar.

The scrollbar supports a set widget command that changes the size and location of the slider and can be invoked by the widget associated with the scrollbar.

To make a canvas and scrollbar combination you'd use the -xscrollcommand option to register the appropriate scrollbar's set command to the canvas, and the scrollbar -command option to register the canvas's xview command to the scrollbar.

This code will create a small (50x50) window into a larger (200x50) canvas with a horizontal scrollbar to reposition the displayed window, looking a lot like figure 1:

```
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
    -xscrollcommand {.xsb set}
scrollbar .xsb -orient horizontal -command {.c xview}
```

Figure 1

```
grid .c -row 0 -column 0
grid .xsb -row 1 -column 0 -sticky ew

.c create rectangle 100 20 120 40
```

This is OK, but the viewable window in this example is a fixed size. It would be nice to enable the user to resize the window. The previous Tclsh Spot article described techniques for making resizable windows. This example uses the grid columnconfigure command to allow a widget to be resized.

Figure 2

```
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
      -xscrollcommand {.xsb set}
scrollbar .xsb -orient horizontal -command {.c xview}
grid .c -row 0 -column 0 -sticky nsew
grid .xsb -row 1 -column 0 -sticky ew

grid columnconfigure . 0 -weight 1

.c create rectangle 100 20 120 40
```

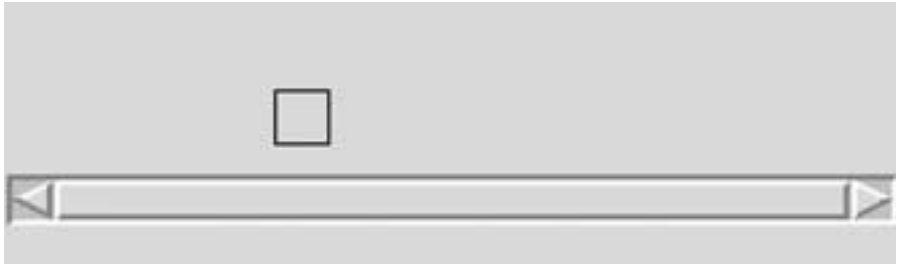

Figure 3

This makes a window that we can expand to look like figure 2, or even stretch to display the entire viewable area of the canvas (figure 3).

Of course, when we can see the entire canvas we don't need the scrollbar. It would be nice to remove the scrollbar when it's not needed.

The most common way to use a scrollbar/widget combination is to invoke a scrollbar directly from the widget and vice versa using the xview and set methods, as we've done here. However, Tcl does not require this style. You can easily write your own scripts to be invoked by the scrollbar and widgets.

When a widget evaluates the script to modify a scrollbar it appends two values for the start and end position of the slider. Both of these are numeric fractions between 0 and 1. If the start position is 0, and the end position is 1, that indicates that the entire viewable area of the widget is displayed, and we don't need a scrollbar.

We can use those values in a procedure that would check to see if the scrollbar were still needed, and remove it when the associated widget is scaled to be fully viewable. This procedure is derived from one of the Tk Tips on the Tcler's Wiki (*http://mini.net/tcl/*). The idea is credited to Brent Welch.

```
# Define a grid command to be sent to the modifyScrollbar proc.
set cmd [list grid .xsb -row 1 -column 0 -sticky ew]

# Create and grid a canvas using the modifyScrollbar procedure to
#  control the scrollbar.
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
      -xscrollcommand [list modifyScrollbar .xsb $cmd]

grid .c -row 0 -column 0 -sticky nsew

# Create the scrollbar. Do not grid. That will be done in
# modifyScrollbar when necessary.

scrollbar .xsb -orient horizontal -command {.c xview}
```

```
# Configure the column to expand when possible.
grid columnconfigure . 0 -weight 1

# Put something in the canvas. Just to add interest.
.c create rectangle 100 20 120 40
```

Figure 4


```
#################################################################
#    proc modifyScrollbar {scrollbar cmd startFract endFract}—
#    Control the scrollbar appearance.
# Arguments
#    scrollbar :  The name of the scrollbar widget.
#    cmd       :  The command to display the scrollbar (grid, pack, place).
#    startFract:  Fraction for the start edge of the slider. PROVIDED BY TCL.
#    endFract:    Fraction for the end edge of the slider. PROVIDED BY TCL.
#
# Results
#    If necessary, the scrollbar is displayed in (or removed from) the
#    parent frame.
#    The slider is modified to reflect new values.

proc modifyScrollbar {scrollbar cmd startFract endFract} {
    if {($endFract < 1.0) || ($startFract > 0)} {
        eval $cmd
        $scrollbar set $startFract $endFract
    } else {
        eval [lindex $cmd 0] forget $scrollbar
    }
}
```

You could hardcode a grid command in the modifyScrollbar procedure, but since the place, pack, and grid window managers all support a forget subcommand, this trick of passing the command to eval allows the modifyScrollbar procedure to be used with any geometry manager.

This technique is fine for a single canvas, text, or listbox widget, but the more common problem is just plain having too many widgets to fit on the screen. Having a scrollable frame would make constructing a scrollable display easy, but frame widget doesn't support the necessary xview and yview widget commands.

However, what we can do is place a frame inside a canvas, and then scale and scroll the canvas as necessary.

Mark Harris and Michael McClennan describe making a scrollable canvas and frame in Effective Tcl/Tk. The big trick is that the frame should be a child window of the canvas you are going to place it into.

```
# Define the grid commands for X and Y scrollbars.

set xCmd [list grid .xsb -row 1 -column 0 -sticky ew]
set yCmd [list grid .ysb -row 0 -column 1 -sticky ns]

# Create a canvas and grid it.
canvas .c -height 50 -width 50 -scrollregion {0 0 200 50} \
    -xscrollcommand [list modifyScrollbar .xsb $xCmd] \
    -yscrollcommand [list modifyScrollbar .ysb $yCmd]
```

```
grid .c -row 0 -column 0 -sticky nsew

# Create the scrollbars; they'll be gridded when needed.
scrollbar .xsb -orient horizontal -command {.c xview}
scrollbar .ysb -orient vertical -command {.c yview}

# Allow the canvas to resize when the parent resizes.
grid columnconfigure . 0 -weight 1
grid rowconfigure . 0 -weight 1

# Create a frame as a child of the canvas, place it in the
# canvas, and bind resizing the canvas to frame size changes.
frame .c.f
.c create window 0 0 -window .c.f -anchor nw
bind .c.f <Configure> {.c configure -scrollregion [.c bbox all]}
```

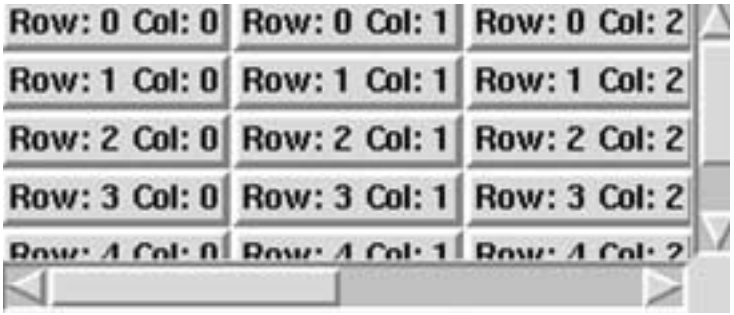

Figure 5

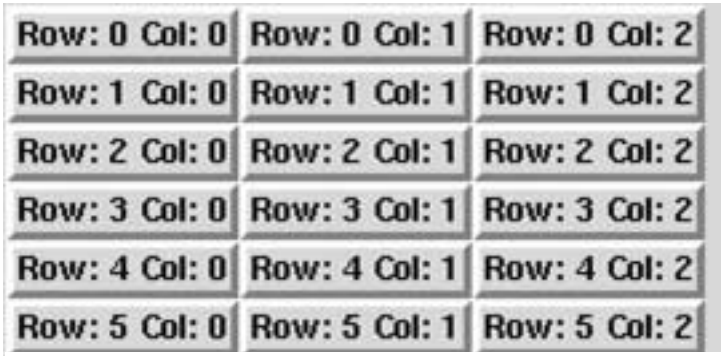

Figure 6

```
# Build some label widgets in the frame for a demo.

for {set i 0} {$i < 3} {incr i} {
 for {set j 0} {$j < 6} {incr j} {
    label .c.f.l_$i,$j -text "Row: $j Col: $i" \
-relief raised -borderwidth 3
    grid .c.f.l_$i,$j -row $j -column $i
}
}
```

This set of code will create a window that looks like figure 5.

If the main window is expanded, the scrollbars go away, as in figure 6.

There are a couple of lines to take note of in this code:

```
.c create window 0 0 -window .c.f -anchor nw
```

This places the frame inside the canvas, with the upper left corner of the frame at the top left corner of the canvas.

```
bind .c.f <Configure> {.c configure -scrollregion
 [.c bbox all]}
```

This line causes the canvas scrollregion to be updated whenever the size of the frame is modified. The frame size will be modified when a user resizes the window or a new widget is added to the frame.

The bind command binds an event/window pair to a script.

Syntax: bind *window event script*

Causes script to be evaluated if event occurs while window has focus.
window The name of the window to which this script will be bound.
event The event to use as a trigger for this script.
script The script to evaluate when the event occurs.

The .c bbox all command is a canvas command that returns the bounding rectangle for a set of canvas objects. The all option tells the canvas to select all objects it is displaying.

The bounding rectangle for all the objects in the canvas (in this case, just the one frame) is the total displayable area of the canvas. We can then configure the -scrollregion to that area, to allow all the objects in the canvas to be scrolled to.

When the canvas is resized by the configure command, it automatically invokes its -xscrollcommand and -yscrollcommand scripts.

This is a useful set of code, but it would be more useful to be able to create scrollable frames when needed.

This code pulls together these ideas into a pair of procedures to create scrollable frames with scrollbars that appear and vanish as needed.

```
package provide scrollFrame 1.1

#################################################################
# proc scrollingFrame {name args}—
#    scrollingFrame - Returns the name of a frame within a canvas
#    attached to vanishing scrollbars.
# Arguments
#    name        The name of the parent frame.
#      NOTE  ::   NON-CONVENTIONAL RETURN – RETURNS THE INTERNAL
#                 NAME, NOT THE NAME OF THE PARENT WINDOW!!!
#    args        Arguments to be passed to frame and canvas
#
# Results
#    Creates 2 frames, a canvas and a two scrollbars.
#    |-------------|   <- Outer holding frame
#    | ccccccccc ^|   Canvas within outer frame
#    | cfffffffc ||   Frame within canvas
#    | cf       fc ||
#    | cfffffffc ||   <— Vertical Scrollbar within outer frame
#    | ccccccccc v|
#    | <--------> |   Horizontal Scrollbar within outer frame
#    |-------------|
#
proc scrollingFrame {outerFrame args} {
    if {[string first "." $outerFrame] != 0} {
        error "$outerFrame is not a legitimate window name -
            must start with '.'"
    }

    # Create the outer frame (or not if it already exists).
    catch {frame $outerFrame}

    # Build the scrollbar commands for the X and Y scrollbar.

    set cmdy [list modifyScrollBar $outerFrame.sby \
        [list grid $outerFrame.sby -row 0 -column 1 -sticky ns]]

    set cmdx [list modifyScrollBar $outerFrame.sbx \
        [list grid $outerFrame.sbx -row 1 -column 0 -sticky ew]]

    # Create and grid the canvas.
    set cvs [canvas $outerFrame.c -yscrollcommand $cmdy
          -xscrollcommand $cmdx]
    grid $outerFrame.c -row 0 -column 0 -sticky news

    # Create the scrollbars. Do not grid. They'll be gridded when
    # needed.
        scrollbar $outerFrame.sby -orient vertical -command "$cvs yview"
        scrollbar $outerFrame.sbx -orient horizontal -command "$cvs xview"
```

```
    # Configure the canvas to expand with its holding frame.
    grid rowconfigure       $outerFrame 0 -weight 1
    grid columnconfigure  $outerFrame 0 -weight 1

    # Create a frame to go within the canvas. The various frame
    #  options are applied here.
    eval frame $cvs.f $args

    # Place the new frame within the canvas.
    $cvs create window 0 0 -window $cvs.f -anchor nw

    # Bind frame changes to modify the canvas scrollregion.
    bind $cvs.f <Configure> "$cvs configure -scrollregion \[$cvs bbox all\]"

    return $cvs.f
}

#################################################################
#
#    proc modifyScrollbar {scrollbar cmd startFract endFract}—
#        Control the scrollbar appearance.
# Arguments
#    scrollbar    :  The name of the scrollbar widget.
#    cmd          :  The command to display the scrollbar (grid, pack, place).
#    startFract   :  Fraction for the start edge of the slider. PROVIDED BY TCL.
#    endFract     :  Fraction for the end edge of the slider.  PROVIDED BY TCL.
#
# Results
#    If necessary, the scrollbar is displayed in (or removed from) the
#    parent frame.
#    The slider is modified to reflect new values.

proc modifyScrollBar {scrollbar cmd startFract endFract} {
    if {($startFract > 0) || ($endFract < 1.0)} {
        eval $cmd
        $scrollbar set $startFract $endFract
    } else {
        eval [lindex $cmd 0] forget $scrollbar
    }
}
```

The frame returned by the scrollingFrame procedure can be used just like any other frame; you can place new widgets into it using pack, place, or grid; set the relief or background color; and so on. However, if the frame is too small to display all the widgets, it will suddenly acquire a set of scrollbars.

Here's a short example that uses the scrollingFrame procedure to create a frame and then populates that frame with a bunch of labels.

```
set ff [scrollingFrame .f2 -background yellow -height 30 -width 30]
grid .f2 -row 0 -column 0 -sticky news

for {set i 0} {$i < 200} {incr i} {
    label $ff.l_$i -text $i
    grid $ff.l_$i -row [expr $i / 10] -column [expr $i % 10]
}
grid rowconfigure . 0 -weight 1
grid columnconfigure . 0 -weight 1
```

As usual, this code is available at *http://www.noucorp.com*.