Menus

The following example shows how to develop a simple menu, with three options. When the menu button is activated by a button 1 press, the menu drops down ready for selection.

If the menu button is activated by a button 2 press, the menu drops down, and can be “torn off”, and positioned anywhere on the screen. To revert to normal menu mode, the menu button should be re-activated by a button 1 press.

```
#First make a menu button
menubutton .menu1 -text “Unix commands” -menu .menu1.m
    -underline 0

#Now make the menu, and add the lines one at a time
menu .menu1.m
.menu1.m add command -label “List Files” -command {ls}
.menu1.m add command -label “Get date” -command {date}
.menu1.m add command -label “Start calendar” -command {xcalendar}

pack .menu1
```

Scrollbars

The following example shows how to control a listbox with two buttons, one to scroll the data in the listbox up, and one to scroll the data down.

```
#Set up a listbox to show European cities
listbox .listbox2
#The following is an example of a widget specific command -this is
#the insert operation for listboxes, which inserts a list of
#items starting at line 0
.listbox2 insert 0 Aberdeen Alicante Athens Basel Berne
     Bilbao Cannes Edinburgh Glasgow Geneva
     Lisbon London Madrid
     Munich Paris Rome
     Rouen Venice Zurich
#The only way this can be scrolled at present is by
#using Button 2 to grab the text. Now we'll try to attach
#some buttons to control it.
#Set up a new frame with two buttons
frame .fr -relief raised
button .fr.button1 -text "UP" -command {.listbox2 yview 1}
button .fr.button2 -text "DOWN" -command  {.listbox2 yview 10}
#The buttons are associated with listbox commands to move the
#data which is in view.
#Now pack everything together
pack .listbox2 .fr -side left -padx 1m -fill x
pack .fr.button1 .fr.button2 -padx 1m -pady 2m -fill x -fill y
```

Scrolling and scrollbar widgets

A simple way to see this is to run the earlier listbox example - with the rainbow colours.

Now issue the commands

```
.listbox1 yview 3
.listbox1 yview 10
.listbox1 yview 20
.listbox1 yview 1
.listbox1 yview end
```

and watch what happens. You should see the contents of the box move up and down in response to the commands.

To provide the Load facility, a Load button is provided

```
button .load -text Load -command {LoadData}
```

and a new procedure

```
proc LoadData {} {
        set infile [open inputfile r]
        gets $infile $firstname1
        gets $infile $surname1
        gets $infile $address1
        gets $infile $telephone
        gets $infile $age1
        gets $infile $sex
        }
```

Modifying the application to display pre-stored data

The application as developed so far can be used to input data and save it to a file. Two useful modifications can be made - firstly to clear all the data, and secondly to initialise the data to data stored in an appropriate input file - which for simplicity we shall call ÒinputfileÓ.

To provide the clear data facility, we shall simply add a new button

```
button .clear -text Clear -command {ClearData}
```

and a new procedure

```
proc ClearData {} {
       set firstname1 ""
       set surname1 ""
       set address1 ""
       set telephone ""
       set age1 ""
       set sex Male
         #or female if you prefer!
       }
```

Using check buttons

With a little more ingenuity, it is possible to use a single check button to allow the value for the variable sex to be set.

```
#set gender "Female"  - initialisation - if needed
checkbutton .sex1 -width 15 -textvariable "gender" -variable sex
   -onvalue Male -offvalue Female
   -command {set gender $sex}
```

When the checkbutton is set on, the value of the variable sex is set to Male, and when the button is off, the value is set to Female.

Using radio buttons

In the application developed here, the entry for sex has not been carried out according to our original specification, which suggested that a radiobutton be used.

The following is a possible implementation of a pair of radio buttons which will achieve the desired effect:

```
radiobutton .male –width 10 –text “Male”  –variable sex –value “Male”
radiobutton .female –width 10 –text “Female” –variable sex –value “Female”
```

Since both radio buttons are bound to the same variable, sex, the mutual exclusivity is determined by the system. Note that initially neither button is set, so strict mutual exclusivity only applies once one of the buttons has been selected. If an initial setting is required, this can be handled by the use of an appropriate set command, such as

```
set sex Female
```

Clearly these radio buttons will need to be packed into the application appropriately.

Data entry example

With the widgets so far described, it is possible to make a small data entry example.

The following personal data is to be collected:

Surname, First names, Address, Telephone number, Sex, Age

For the Surname, First names, Telephone number and Age, entry widgets will be used. For the Address, a text widget will be used, and for the Sex input, a radiobutton will be used.

The layout of the display will have a column of labels to the left, with a column of entry and other widgets to the right.

This can be implemented using two frames, one for each column.

Additionally, when all the data is entered, a Save button will store the data in a file, for future use, either by a TCL program, or by an external program, and a Quit button wll cause the program to finish.

Procedures

TCL code fragments can be useful for avoiding the need to specify a large number of widgets explicitly, as in the calendar example in the previous section. However, it is almost certainly better to make such code fragments into TCL procedures, as shown:

```
proc makecalendar {daysinmonth} {
       for {set i 1} {$i <= $daysinmonth} {incr i 1}  {
         button .button$i -width 16 -text “$i” -command “daydetails $i”
         pack   .button$i
         }
}
```

To execute this.. type

```
makecalendar 31
```

Using TCL code to generate widgets

Widgets can be generated explicitly, as in the example here:

```
label .lab1 -text "This is label 1"
label .lab2 -text "This is label 2"
label .lab3 -text "This is label 3"
pack  .lab1 .lab2 .lab3
```

but they can also be generated by code, as in

```
for {set i 1} {$i<=3} {incr i 1}  {
         label .lab$i -text "This is label $i"
         pack  .lab$i
}
```

Here, the for command is rather like the for command of C, and is of the form

```
for {initialisation} {condition} {action} {
          body
}
```

This idea could be used for a calendar application, as in

```
for {set i 1} {$i <=31} {incr i 1} {
         button .button$i -width 20 -text "$i" -command "daydetails $i"
         pack   .button$i
}
```

Packing

TCL provides a means of packing widgets together, and laying them out on the screen. This enables widgets to be placed on the screen in a flexible,and useful way. The operation of the packer is slightly complicated, and it is perhaps best to use it only for horizontal or vertical packing.

For example

```
button .button1 -text “Button1” -command “cmd1”
button .button2 -text “Button2” -command “cmd2”
button .button3 -text “Button3” -command “cmd3”
pack .button1 .button2 .button3
```

simply creates three buttons, and packs them vertically, while.

```
button .button1 -text “Button1” -command “cmd1”
button .button2 -text “Button2” -command “cmd2”
button .button3 -text “Button3” -command “cmd3”
pack .button1 .button2 .button3 -side left
```

Command binding

WARNING - THIS EXAMPLE NEEDS CHECKING - SOMETHING IS WRONG!
Some widgets, such as button widgets, can have a command associated with them. This command is normally executed when button 1 is pressed, as in:

```
#Set up a button with a command”
button .button1 –text “Press Here” –command “exec echo Hello there”
pack .button1
```

Care must be taken when using variables within a command to execute, as follows:

```
#Set up a button with a command”
button .button2 –text “Press Here” –command “exec echo Hello $Name”
pack .button2
```

Widget Binding

The following example is more complex, and shows how selections can be made from a listbox, using the mouse.

```
#Set up a listbox to show rainbow colours
#and demonstrate selection
listbox .listbox2 -exportselection 1
pack .listbox2

#The following is an example of a widget specific command -this is
#the insert operation for listboxes, which inserts a list of
#items starting at line 0
.listbox2 insert 0 red orange yellow green blue indigo violet

#There are no pre-declared bindings for the listbox
#Now bind a button release event on button 1 to the widget
#This will NOT interfere with the selection process!
bind .listbox2 <ButtonRelease-1> {showindices}

#Now define procedure to show selected indices
proc showindices {} {
       puts [.listbox2 curselection]
}
```

Widget binding

Widgets such as entry and text widgets have an implicit binding to their text variable, so that as the data changes within the widget, the text variable is automatically kept up to date. It is also possible to explicitly bind together a widget with an event to produce an action.

The instruction to bind a widget to an action is the bind command. An example is given here using a frame widget:

```
frame .frame1 -width 100 -height 100
     bind .frame1 <Button-1> {puts "frame1: Button 1 pressed"}
     bind .frame1 <Button-2> {puts "frame1: Button 2 pressed"}
     bind .frame1 <Button-3> {puts "frame1: Button 3 pressed"}
     bind .frame1 <ButtonRelease-1> {puts "frame1: Button 1 released"}
     bind .frame1 <Double-Button-1>
                    {puts "frame1: Double click Button 1"}
     bind .frame1 <Any-Motion> {puts "frame1: pointer at %x,%y"}
     pack .frame1
```

Configuration options

The following options are available in all widgets currently supported:

The foreground colour
The background colour

The following options are commonly available;

geometry: Specifies height and width of widget - widgets:

width: Specifies width of widget - widgets: entry

height: specifies height of widget - widgets: listbox

text: specifies text associated with widget - widgets:

textvariable: specifies text variable associated with widget - widgets: entry

```
button .b -text “Press Me” -command exit
pack .b
.b configure -background Red
```

Scale widgets

Scale widgets allow direct data entry of numerical data, by means of a slider. The following example creates a scale widget, and a label, and the value set by the scale widget is displayed in the label. The scale is set to show an initial value.

```
#Make a scale, oriented vertically, going down from 50 to 0.
#Tick marks are places alongside the scale every 10 units.
scale .scale1 -width 10 -orient vertical -length 280
            -from 50 -to 0  -command “set var1”
            -tickinterval 10 -relief raised

label .lab1 -width 10 -textvariable var1 -relief sunken
pack .scale1 .lab1 -padx 1m -pady 2m  -fill x

#Now set the value of scale to 25 using a widget command
.scale1 set 25
```

Text widgets

Text widgets are complex widgets, which allow many lines of text to be entered, and edited. Editing can be performed by moving the cursor into the text with the mouse, and typing or deleting characters as appropriate. Text widgets are complex in their operation, and support a number of different features, including the ability to mark and tag included text for highlighting purposes, and to all the use of displayed tact characters for selection by an event such as a mouse press.

```
text .text1 -width 45 -height 5 -relief sunken
pack .text1

#The following is another example of a widget specific command -
#this is the insert operation for text widgets, which
#inserts a list of characters starting at line 1, character 0.

.text1 insert 1.0 "     Twas brillig and the slithy toves
   did gyre and gimble in the wabe
   All mimsy were the borogoves
   and the mome raths outgrabe"
```

Listbox widgets

```
#Set up a listbox to show rainbow colours
listbox .listbox1
pack .listbox1

#The following is an example of a widget specific command -this is
#the insert operation for listboxes, which inserts a list of
#items starting at line 0
.listbox1 insert 0 red orange yellow green blue indigo violet
```

An example of how mouse clicks can be used to select items from a listbox will be discussed under Widget Binding.

Entry widgets

Entry widgets provide entry boxes for data entry, and are defined as shown:

```
entry .entrybox1  -textvariable variable
pack  .entrybox1
```

to make it easier to see, try:

```
entry .entrybox2 -relief ridge -textvariable variable
pack  .entrybox2
```

The textvariable configuration option associates the entry box with a TCL variable variable.

try doing a

```
puts $variable
```

or

```
echo $variable
```

after you have entered something in the dialog box.

Message widgets

Message widgets are similar to label widgets, since they display text, but the text may extend over several lines:

```
message .msg –text “When the lights turn
     green you may proceed with caution – when the
     lights are red you must stop – when
     the lights turn  to red you should stop, unless
     it is dangerous to do so, and would endanger
     following traffic – when the lights are amber, you
     should prepare to stop, and when the
     red and amber lights are on, you may prepare to
     move off.”

pack .msg
```

It is allowable to insert continuation characters, tabs (\t), end of line (\n).

you can use the -relief, sunken options with message widgets as with labels.

Different forms of labels:

```
#This example shows different forms for the label widget
     label .lab1 -text “This is text”
     label .lab2 -text “This is coloured text” -foreground red
                 -background blue
     label .lab3 -text “Sunken text”  -relief sunken
     label .lab4 -text “Grooved text” -relief groove
     label .lab5 -text “Flat text”    -relief flat
     label .lab6 -text “Ridged text”  -relief ridge
     label .lab7 -text “Raised text”  -relief raised
     label .lab8 -text “Times font”
                  -font *-times-medium-r-normal--*-100-*
     pack  .lab1 .lab2 .lab3 .lab4 .lab5 .lab6 .lab7 .lab8
                    -padx 2m -pady 1m -fill x
```

Label widgets

Create text labels:

```
label .label1  -text "This is a text label"
pack  .label1
```

The label command causes the label to be generated, while the pack command is needed to cause the label to be displayed.

```
label .label2 -text "More text" -relief sunken
pack  .label2
```

gives a boxy feel to the label text, while

```
label .label3 -text "Coloured text" -background red -foreground blue
pack  .label3
```

gives coloured text.

```
set text "Hello there"
label .label4 -text $text
pack  .label4
```

Introduction to TCL/TK

Widgets

These are window objects that can be used for a variety of purposes.

E.g.

toplevel:	a sort of a container for other widgets. Typically interacts with the window manager.
frame:	a container for widgets that rests under the top level.
label:	text label
entry:	box for data entry
message:	displaying multiple line messages
text:	inputting and editing text
checkbutton:	displays the button position
scale:	easy way of setting numeric values
listbox:	displays items for selection
scrollbar:	horizontal / vertical scrollbars for other widgets
canvas:	displays graphics.

CS590C: Introduction to Computational Science

Lecture 7: Tools for developing User Interfaces: Tcl/Tk.

This lecture is intended to familiarize you with:

- ❏ Tcl/Tk
- ❏ Developing Applications with Tcl/Tk