Perl/Tk Tutorial - Create GUI with Perl's Tk Module

Introduction

Perl/Tk (also known as pTk) is a collection of modules and code that attempts to wed the easily configured Tk 8 widget toolkit to the powerful lexigraphic, dynamic memory, I/O, and object-oriented capabilities of Perl 5 In other words, it is an interpreted scripting language for making widgets and programs with **Graphical User Interfaces (GUI)**

Perl or *Practical Extraction and Report Language* is described by Larry Wall, Perl's author, as follows: "Perl is an interpreted language optimized for scanning arbitrary text files, extracting information from those text files, and printing reports based on that information It's also a good language for any system management tasks The language is intended to be practical (easy to use, efficient, complete) rather than beautiful (tiny, elegant, minimal)" The perlintro man page has this to say

> Perl is a general-purpose programming language originally developed for text manipulation and now used for a wide range of tasks including system administration, web development, network programming, GUI development, and more

Tk, the extension(or module) that makes GUI programming in perl possible, is taken from **Tcl/Tk** Tcl(Tool Command Language) and Tk(ToolKit) was created by Professor John Ousterhout of the University of California, Berkeley Tcl is a scripting language that runs on Windows, UNIX and Macintosh platforms Tk is a standard add-on to Tcl that provides commands to quickly and easily create user interfaces Later on Tk was used by a lot of other scripting languages like Perl, Python, Ruby etc

Applications

Perl has been used since the early days of the web to write CGI scripts, and is now a component of the popular LAMP (Linux/Apache/MySQL/Perl) platform for web development Perl has been called "the glue that holds the web together" Large systems written in Perl include Slashdot, and early implementations of Wikipedia and PHP

Perl finds many applications as a glue language, tying together systems and interfaces that were not specifically designed to interoperate Systems administrators use Perl as an all-purpose tool; short Perl programs can be entered and run on a single command line

Philosophy

Perl has several mottos that convey aspects of its design and use One is There's more than one way to do it (TMTOWTDI - usually pronounced 'Tim Toady') Another is Perl: the Swiss Army Chainsaw of Programming Languages A stated design goal of Perl is to "make easy tasks easy and difficult tasks possible"

Perl is free software, and may be distributed under either the Artistic or the GPL License It is available for most operating systems but is particularly prevalent on Unix and Unix-like systems (such as Linux, FreeBSD, and Mac OS X), and is growing in popularity on Microsoft Windows systems

Perl/Tk Requirements

Before starting with the tutorial, make sure you have the following things If some are missing you still can learn *perl* - but you will not be able to use it to its full power

1. ActivePerl from http://wwwactivestatecom/ActivePerl/ for windows - for programming in Windows Linux don't need any special outside interpreter because it already has it in most of the distributions
2. A good text editor I would recommend Crimson Editor(http://wwwcrimsoneditorcom/) for Windows and XEmacs for Linux

Installing/Using Perl

In **Unix/Linux** you can execute your perl scripts by typing "*perl <filename>*" at command prompt But before you do that make sure you have **both** Perl and its Tk module Most linux distributions have perl - but quite a few don't have the Tk module Make sure that the system you are using have the Tk module If you don't have it, go to http://wwwcpanorg and download the perl module Or you can use the perl's CPAN module to install the Tk module To do this, open a terminal and enter the following command

```
perl -MCPAN -e shell
cpan> install Bundle::CPAN
cpan> reload cpan
cpan> install Tk
```

Another(and a much easier) way to do this is to get a rpm of Perl/Tk and installing it with the command

```
rpm -ivh FILENAME
```

If you are using Ubuntu, a easy way of installing Perl/Tk is using this command

```
sudo apt-get install perl-tk
```

If you are using Windows, download ActivePerl and install it Then you can execute any perl file by double clicking it

Two more things before we begin the tutorial I will be teaching perl/tk and I expect you to know how to program in perl I may ignore some of the perl coding conventions like including `use strict;`, `-w` or `use warnings;` in my examples The examples have only one purpose - to demonstrate the feature that will be taught in that part of the tutorial Sorry about that - but I have to keep my tutorial's example scripts short and to the point

Finally, this is a *tutorial for Perl/Tk only* - I will not be teaching perl here So if you know perl, continue But if you are a beginner to perl, I would recommend that you read my **perl tutorial**

Hello World

Let us begin, as all other tutorials begin, with the "Hello World" program Create a file called "Hellopl" and enter the following into it

```
#!/usr/local/bin/perl
use Tk;
# Main Window
my $mw = new MainWindow;
my $label = $mw -> Label(-text=>"Hello World") -> pack();
my $button = $mw -> Button(-text => "Quit",
                -command => sub { exit })
        -> pack();
MainLoop;
```

The first line - `#!/usr/local/bin/perl` is not needed in windows In Linux, it tells the name of the script language processor In our case it is perl Don't understand what that means? Don't worry your gray cells over it Just put it at the top of the file

The second line - `use Tk;` tells the interpreter that our program will use the Tk module This line is an absolute must

in all GUI programs you make using perl When the interpreter encounters this line, it will load the Tk components that we will be using to create our program

The third line - This is a comment Any line that starts with a '#' char is a comment Comments are not of any use in the program It is used by programmer to talk to themselves A programmer cannot be expected to remember every thing a script does So he uses a comment to write it down Next time he edits the script, he can read the comment and understand what the program is for It is good practice to make as much comments as possible

The fourth line, `my $mw = new MainWindow;`, will create a window into which the GUI elements will be placed The variable $mw is a object of type 'MainWindow' We will have to use this element when we want to place any widget inside it

The fifth line - `$mw -> Label(-text=>"Hello World") -> pack();` makes a label and writes "Hello world" in it You can change the text to any thing you like Note the structure of the command -

`$label` - This variable assigned to that particular widget Ever widget must have a UNIQUE variable This name will be used when ever that widget must be accessed

`$mw ->` - $mw is the MainWindow's object We will be placing our label widget inside this window

`Label(-text=>"Hello World")` - 'Label' is the name of the widget A widget is a user interface object in X graphical user interfaces Confused? Lets just say that it is the name of the object that appears on screen There are many other widgets too If you want to display a button, you use the button widget For text, you use the text widget For entry, you guessed it, the entry widget If you want, you can see more about widgets

-text=>"Hello World" - The option for this widget This option says that this widget must be given the text "Hello World" Options change according to the widgets - a button widget will not have all the options of the label widget and vise versa But there will be many common ones
Please note that operator used here is '=>' as opposed to the one used earlier '->' in `$mw ->` One uses the minus(-) sign while the other uses the equals(=) sign Do not confuse between these two
You can keep writing other options can also be written here For example, let us make a label for showing the text "Hello World" The other lines are same as the Hello World program

```
$mw -> Label(-text=>"Hello World",-font=>"courierfont",-relief=>"raised") ->
pack();
```

In this example, a lot more options are used The font option is used to tell which font must be used to make the text and the relief option tells whether the text should appear raised, sunken, flat etc To know all the options for a particular widget, read the manual that comes with Perl It lists every widget and every option they have If you are going to program in Perl, you will find your self peeking into the manual every few minutes The most important and most commonly used options are listed here

All options must separated by a comma But as you have noted, this line is a little difficult to read As the number of options increase, the more difficult to read it So a more readable version is

```
$mw -> Label(-text=>"Hello World",
                -font=>"courierfont",
                -relief=>"raised")
        -> pack();
```

Next comes the `-> pack();` This will pack the widget '$label' into the window '$mw' 'pack' is a geometry manager Another geometry manager is 'grid' Personally, I like grid better Once again, putting all this in one line is an eye sore - so you can put this part in the next line

```
my $label = $mw -> Label(-text=>"Hello World")
        -> pack();
```

In this case, pack has no options within it But that is not always the case

```
my $label = $mw -> Label(-text=>"Hello World")
        -> pack(-side=>"left",
                -anchor=>'w');
```

You don't have to pack the widget in the same line of creating it - but it is convenient in small programs You can pack the widget later using the widget's variable For example

```
my $label = $mw -> Label(-text=>"Hello World"); #We created the widget
$label -> pack(-side=>"left", -anchor=>'w'); #We pack it in another line
```

So we have the final syntax of how to create and display a widget

```
my $WidgetVariable = $Window -> WidgetType(?Option 1=>Value 1, ?Option 2=>Value 2 ??) -> pack();
```

The next three lines

```
my $button = $mw -> Button(-text => "Quit",
                -command => sub { exit })
        -> pack();
```

will create and display a button Here the widget variable is '$button' When we look at the options, we will find two options - 'text' and 'command' The given text is Quit - so the button will have the text "Quit" on it The command option determines what should happen when the user click on the button You can specify a function to execute when the user clicks on the button In this case the program will exit when this button is pressed One can also call functions that you have created from here

```
#!/usr/local/bin/perl

use Tk;
# Main Window
my $mw = new MainWindow;
my $label = $mw -> Label(-text=>"Hello World") -> pack();
my $button = $mw -> Button(-text => "Quit",
                -command =>\&exitProgam)
        -> pack();
MainLoop;

sub exitProgam {
        $mw->messageBox(-message=>"Goodbye");
        exit;
}
```

The next line - `MainLoop;` is the Main Loop or the Event Loop Its job is to invoke callbacks in response to events such as button presses or timer expirations If this line is missing, the program will run and exit with out waiting for the user to do any thing This is another one of those 'absolute musts' of Perl/Tk programming

Now Perl puritans will raise a great hue and cry and say that this is not the way to print "Hello World" The "pure" method is the following

```
#!/usr/local/bin/perl
print "Hello World"
```

Putting things in perspective, I am teaching Perl/Tk - not Perl The above is the Perl method of doing it My method is the pTk method of doing it

Widgets 1 : Button, Entry, Label

A widget is a user interface object in X graphical user interfaces Confused? Lets just say that it is the name of the object that appears on screen There are many types widgets If you want to display a button, you use the button widget For text, you use the text widget For entry, you guessed it, the entry widget

Syntax:

my $WidgetVariable = $Window -> WidgetType(?Option 1=>Value 1, ?Option 2=>Value 2 ??) -> pack();

Three things need to be said about widgets First is the **widget variable** This I have explained earlier The widget variable of all widgets must be unique and will be used whenever that widget needs to be accessed Second is the **options** Each widget has some options which can be used to configure it This is usually done when the widget is declared, but it can be done afterward also The final thing is **commands** Each widget has some commands which also

can be used to configure it or make it do some thing

But before we begin, we need to know a little about the pack command I have explained this earlier but just doing it one more time so that you don't have to push the back button Pack is a geometry manager Another geometry manager is 'grid' - we will explore that latter Pack is much more simpler than grid
The line *$hello -> pack;* tells the interpreter to pack the widget called "$hello"

Button

This will make a button It can be configured to execute some code when the button is pushed This will usually refer to a function so when the button is pushed, the function will run An button is shown below This button is created using HTML input tag

Some Options

-text=>"TEXT"	TEXT will be the text displayed on the button
-command=>CALLBACK	CALLBACK will be the code that is called when the button is pushed

```
#!/usr/local/bin/perl
use Tk;

# Main Window
my $mw = new MainWindow;

my $but = $mw -> Button(-text => "Push Me",
                  -command =>\&push_button);
$but -> pack();

MainLoop;

#This is executed when the button is pressed
sub push_button {
         whatever
}
```

You may have noticed that I used a slash(\) in the command callback (`-command =>\&push_button);`) Make sure that the slash stays there - to see why, go to the Most common mistakes by Perl/Tk beginners

---

Entry

An entry is a widget that displays a one-line text string and allows the user to input and edit text in it When an entry has the input focus it displays an insertion cursor to indicate where new characters will be inserted An entry element is shown using HTML

Some Options

-width=>NUMBER	Width of the input field NUMBER should be an integer
-textvariable=>\$VARIABLE	The contents of the variable VARIABLE will be displayed in the widget If the text in the widget is edited, the variable will be edited automatically
-state=>STATE	The state of the input field It can be **normal**, **disabled**, or **readonly** If it is **readonly** the text can't be edited

Some Commands

Syntax	**Description**	**Example**
$widget -> get();	The text inside input field can be taken by this command	$name = $ent -> get();
$widget -> delete(FIRST?,LAST?);	Delete one or more elements of the entry FIRST is	$ent -> delete(0,'end');

	the index of the first character to delete, and LAST is the index of the character just after the last one to delete If last isn't specified it defaults to FIRST+1, ie a single character is deleted This command returns an empty string	
$widget -> insert(*index*,"STRING");	Insert the characters of STRING just before the character indicated by *index* Index is 0 for the first character The word "end" can be used for the last character	$ent -> insert('end',"Hello");

Example

```
#!/usr/local/bin/perl
use Tk;

# Main Window
my $mw = new MainWindow;

#GUI Building Area
my $ent = $mw -> Entry() -> pack();
my $but = $mw -> Button(-text => "Push Me",
                  -command =>\&push_button);
$but -> pack();

MainLoop;

#This is executed when the button is pressed
sub push_button {
        $ent -> insert('end',"Hello");
}
```

Label

This widget display text messages

Some Options

-text => "TEXT"	TEXT will be the text displayed on the button
-font => FONT	Specifies the font to use when drawing text inside the widget You can specify just the font or you can give it in this format "FONTNAME SIZE STYLE" The STYLE can be bold, normal etc

Example

```
#!/usr/local/bin/perl
use Tk;

my $mw = new MainWindow; # Main Window

my $lab = $mw -> Label(-text=>"Enter name:") -> pack();
my $ent = $mw -> Entry() -> pack();
my $but = $mw -> Button(-text => "Push Me",
                  -command =>\&push_button);
$but -> pack();

MainLoop;

#This is executed when the button is pressed
sub push_button {
        $ent -> insert(0,"Hello, ");
}
```

Widgets 2 : Frame, Text, Scrollbar, Scale

Frame

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 Frame can be created just like any other widget -

```
my $frm = $mw -> Frame();
```

To place other widgets in this frame, you should use the frame widget variable as its parent Normally the parent is '$mw' or the MainWindow But if we wish to put a widget inside a frame, use the frame variable('$frm' in this case) in place of '$mw' Like this

```
my $lab = $frm_name -> Label(-text=>"Name:") -> pack();
```

Some Options

-relief=>STYLE	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

Example

```
#!/usr/local/bin/perl
use Tk;

my $mw = new MainWindow; # Main Window

my $frm_name = $mw -> Frame() -> pack(); #New Frame
my $lab = $frm_name -> Label(-text=>"Name:") -> pack();
my $ent = $frm_name -> Entry() -> pack();

my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button) -> pack();

MainLoop;

#This function will be executed when the button is pushed
sub push_button {
        $ent -> insert(0,"Hello, ");
}
```

Text

A text widget displays one or more lines of text and allows that text to be edited Similar to the entry widget but a larger version of it

Some Options

-xscrollcommand => COMMAND	This is to enable communication between a text widget and a scroll bar widget There is a -yscrollcommand similler to this one
-font => FONTNAME	Specifies the font to use when drawing text inside the widget
-width => NUMBER	Specifies the width of the widget
-height => NUMBER	Specifies the, you guessed it, height of the widget

Syntax	Description	Example

$widget -> get(*index1, ?index2 ?*);	Return a range of characters from the text The return value will be all the characters in the text starting with the one whose index is *index1* and ending just before the one whose index is *index2* (the character at *index2* will not be returned) If *index2* is omitted then the single character at *index1* is returned Note that the index of text is different from that of the entry widget The index of text widget is in the form LINE_NOCHARECTER_NO This means that 10 means the first character in the first line	$contents = $txt -> get(10,'end');
$widget -> insert(*index*,DATA);	Inserts all of the chars arguments just before the character at index If index refers to the end of the text (the character after the last newline) then the new text is inserted just before the last newline instead	$txt -> inset('end',"Hello World");

Example

```
#!/usr/local/bin/perl
use Tk;

my $mw = new MainWindow; # Main Window

my $frm_name = $mw -> Frame() -> pack();
my $lab = $frm_name -> Label(-text=>"Name:") -> pack();
my $ent = $frm_name -> Entry() -> pack();

my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button) -> pack();
#Text Area
my $txt = $mw -> Text(-width=>40, -height=>10) -> pack();

MainLoop;

#This function will be executed when the button is pushed
sub push_button {
        my $name = $ent -> get();
        $txt -> insert('end',"Hello, $name");
}
```

Scrollbar

A scroll bar is a widget that displays two arrows, one at each end of the scroll bar, and a slider in the middle portion of the scroll bar It provides information about what is visible in an associated window that displays an document of some sort (such as a file being edited or a drawing) The position and size of the slider indicate which portion of the document is visible in the associated window For example, if the slider in a vertical scroll bar covers the top third of the area between the two arrows, it means that the associated window displays the top third of its document It is made to work with other widgets like text **Some Options**

-orient=>DIRECTION	For widgets that can lay themselves out with either a horizontal or vertical orientation, such as scroll bars, this option specifies which orientation should be used DIRECTION must be either horizontal or vertical or an abbreviation of one of these
-command => COMMAND	This command gets executed when the scroll bar is moved This option almost always has

	a value such as t xview or t yview, consisting of the name of a widget and either xview (if the scroll bar is for horizontal scrolling) or yview (for vertical scrolling) All scrollable widgets have xview and yview commands that take exactly the additional arguments appended by the scroll bar

Example

```
#!/usr/local/bin/perl
use Tk;

my $mw = new MainWindow; # Main Window

my $frm_name = $mw -> Frame();
my $lab = $frm_name -> Label(-text=>"Name:");
my $ent = $frm_name -> Entry();

my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button);

my $textarea = $mw -> Frame(); #Creating Another Frame
my $txt = $textarea -> Text(-width=>40, -height=>10);
my $srl_y = $textarea -> Scrollbar(-orient=>'v',-command=>[yview => $txt]);
my $srl_x = $textarea -> Scrollbar(-orient=>'h',-command=>[xview => $txt]);
$txt -> configure(-yscrollcommand=>['set', $srl_y],
                  -xscrollcommand=>['set',$srl_x]);

$lab -> grid(-row=>1,-column=>1);
$ent -> grid(-row=>1,-column=>2);
$frm_name -> grid(-row=>1,-column=>1,-columnspan=>2);

$but -> grid(-row=>4,-column=>1,-columnspan=>2);

$txt -> grid(-row=>1,-column=>1);
$srl_y -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x -> grid(-row=>2,-column=>1,-sticky=>"ew");
$textarea -> grid(-row=>5,-column=>1,-columnspan=>2);

MainLoop;

#This function will be executed when the button is pushed
sub push_button {
        my $name = $ent -> get();
        $txt -> insert('end',"Hello, $name");
}
```

grid

As you can see I have used 'grid' here Grid is NOT a widget It is a geometry manager like pack but more advanced Lets take a closer look at the commands -

`$widget -> grid(-row=>1, -column=>1);`

This line will tell the interpreter to put the widget called '$txt' in the first row of the first column of its parent widget The below digram will help you understand

	Column 1	Column 2
Row 1	'$txt' widget will be here	'$srl_y' widget's place
Row 2	'$srl_x' widget's position	

Some Options

-sticky => STYLE	This option may be used to position (or stretch) the widget within its cell STYLE is a string that contains zero or more of the characters n, s, e or w Each letter refers to a side (north, south, east, or west) that the slave will "stick" to If both n and s (or e and w) are specified, the slave will be stretched to fill the entire height (or width) of its cavity
-ipadx => AMOUNT	The AMOUNT specifies how much horizontal internal padding to leave on each side of the slave(s) This is space is added inside the slave(s) border
-ipady => AMOUNT	The AMOUNT specifies how much vertical internal padding to leave on each side of the slave(s) Options same as -ipadx
-padx => AMOUNT	The amount specifies how much horizontal external padding to leave on each side of the slave(s), in screen units AMOUNT may be a list of two values to specify padding for left and right separately
-pady => AMOUNT	The amount specifies how much vertical external padding to leave on the top and bottom of the slave(s), in screen units Options same as -padx
-row => N	Insert the slave so that it occupies the Nth row in the grid Row numbers start with 0 If this option is not supplied, then the slave is arranged on the same row as the previous slave specified on this call to grid, or the first unoccupied row if this is the first slave
-column => N	Insert the slave so that it occupies the N'th column in the grid Options same as -row
-rowspan => N	Insert the slave so that it occupies N rows in the grid The default is one row
-columnspan => N	Insert the slave so that it occupies N columns in the grid

Using grid requires a bit of experience - but if you know HTML it would help a lot The rows and columns are just like those in HTML tables - although the codes are very different

---

Scale

Makes a slider that can be adjusted by the user to input a variable

Some Options

-from => NUMBER	Starting Number
-to => NUMBER	Ending Number
-tickinterval => NUMBER	Determines the spacing between numerical tick marks displayed below or to the left of the slider
-varable => NAME	Specifies the name of a global variable to link to the scale Whenever the value of the variable changes, the scale will update to reflect this value Whenever the scale is manipulated interactively, the variable will be modified to reflect the scale's new value

Syntax	**Description**	**Example**
$widget -> get();	Get the current value of the scale	my $age = $scl -> get();
$widget -> set(*value*);	Give the scale a new value	$scl -> set(20);

Example

```
#!/usr/local/bin/perl
use Tk;

#Global Variables
my $age = 10;
```

```
# Main Window
my $mw = new MainWindow;

#GUI Building Area
my $frm_name = $mw -> Frame();
my $lab = $frm_name -> Label(-text=>"Name:");
my $ent = $frm_name -> Entry();
#Age
my $scl = $mw -> Scale(-label=>"Age :",
         -orient=>'v',          -digit=>1,
         -from=>10,             -to=>50,
         -variable=>\$age,      -tickinterval=>10);

my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button);

#Text Area
my $textarea = $mw -> Frame();
my $txt = $textarea -> Text(-width=>40, -height=>10);
my $srl_y = $textarea -> Scrollbar(-orient=>'v',-command=>[yview => $txt]);
my $srl_x = $textarea -> Scrollbar(-orient=>'h',-command=>[xview => $txt]);
$txt -> configure(-yscrollcommand=>['set', $srl_y],
                -xscrollcommand=>['set',$srl_x]);

#Geometry Management
$lab -> grid(-row=>1,-column=>1);
$ent -> grid(-row=>1,-column=>2);
$scl -> grid(-row=>2,-column=>1);
$frm_name -> grid(-row=>1,-column=>1,-columnspan=>2);

$but -> grid(-row=>4,-column=>1,-columnspan=>2);

$txt -> grid(-row=>1,-column=>1);
$srl_y -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x -> grid(-row=>2,-column=>1,-sticky=>"ew");
$textarea -> grid(-row=>5,-column=>1,-columnspan=>2);

MainLoop;

## Functions
#This function will be exected when the button is pushed
sub push_button {
        my $name = $ent -> get();
        $txt -> insert('end',"$name is $age years old");
}
```

Now our little example is becoming more and more like a program We have added the comments to it as it has grown big and is difficult to understand Now we have added a slider with which age can be inputed

Dialogs

Dialogs can be called the elements in a program that detaches itself from the main window This is a VERY general definition and has many problems But for the moment, it will do Tk provides many dialogs

messageBox

This procedure creates and displays a message window with an application-specified message, an icon and a set of buttons Each of the buttons in the message window is identified by a unique symbolic name (see the -type options) After the message window is popped up, messageBox waits for the user to select one of the buttons Push the below button to see an example of messageBox

Some Options

-default=>*name*	Name gives the symbolic name of the default button for this message window ('ok', 'cancel', and so on) See -type for a list of the symbolic names If this option is not specified, the first

	button in the dialog will be made the default
-icon>*iconImage*	Specifies an icon to display IconImage must be one of the following: error, info, question or warning If this option is not specified, then the info icon will be displayed
-message>*string*	Specifies the message to display in this message box
-title>*String*	Specifies a string to display as the title of the message box The default value is an empty string
-type>*predefinedType*	Arranges for a predefined set of buttons to be displayed The following values are possible for predefinedType: **abortretryignore** Displays three buttons whose symbolic names are abort, retry and ignore **ok** Displays one button whose symbolic name is ok **okcancel** Displays two buttons whose symbolic names are ok and cancel **retrycancel** Displays two buttons whose symbolic names are retry and cancel **yesno** Displays two buttons whose symbolic names are yes and no **yesnocancel** Displays three buttons whose symbolic names are yes, no and cancel

Example

```
#!/usr/local/bin/perl
use Tk;
use strict;

# Main Window
my $mw = new MainWindow;
my $button = $mw-<Button(-text=<"Show Quit Dailog", -command =< \&exitTheApp)-<pack();

sub exitTheApp {
        my $response = $mw -< messageBox(-message=<"Really quit?",-type=<'yesno',-
icon=<'question');

        if( $response eq "Yes" ) {
                exit
        } else {
                $mw -< messageBox(-type=<"ok", -message=<"I know you like this application!");
        }
}

MainLoop;
```

---

chooseColor

chooseColor pops up a dialog box for the user to select a color

Some Options

-initialcolor>COLOUR	Specifies the color to display in the color dialog when it pops up

---

getOpenFile

The procedures **getOpenFile** and **getSaveFile** pop up a dialog box for the user to select a file to open or save The getOpenFile command is usually associated with the Open command in the File menu Its purpose is for the user to select an existing file only If the user enters an non-existent file, the dialog box gives the user an error prompt and requires the user to give an alternative selection If an application allows the user to create new files, it should do so by

providing a separate New menu command

The getSaveFile command is usually associated with the Save as command in the File menu If the user enters a file that already exists, the dialog box prompts the user for confirmation whether the existing file should be overwritten or not

Some Options

-initialdir=>DIRNAME	Specifies that the directories in directory should be displayed when the dialog pops up If this parameter is not specified, then the directories in the current working directory are displayed If the parameter specifies a relative path, the return value will convert the relative path to an absolute path
-defaultextension=>EXTENSION	Specifies a string that will be appended to the filename if the user enters a filename without an extension The default value is the empty string, which means no extension will be appended to the filename in any case
-filetypes=>filePatternList	If a File types listbox exists in the file dialog on the particular platform, this option gives the filetypes in this listbox When the user choose a filetype in the listbox, only the files of that type are listed If this option is unspecified, or if it is set to the empty list, or if the File types listbox is not supported by the particular platform then all files are listed regardless of their types This is a little tricky - see manual for information
-initialfile=>FILENAME	Specifies a filename to be displayed in the dialog when it pops up
-multiple	Allows the user to choose multiple files from the Open dialog

---

Toplevel

toplevel is a widget This can be used to create custom dialog boxes A toplevel is similar to a frame except that it is created as a top-level window: its X parent is the root window of a screen rather than the logical parent from its path name The primary purpose of a toplevel is to serve as a container for dialog boxes and other collections of widgets The only visible features of a toplevel are its background color and an optional 3-D border to make the toplevel appear raised or sunken

One can use toplevel to create new windows The widgets can be packed inside it in the same way widgets are packed inside a frame An example

```
#!/usr/local/bin/perl
use Tk;
# Main Window
$mw = new MainWindow;

my $lab = $mw -> Label(-text=>"This is the root window",
                -font=>"ansi 12 bold") -> pack;
my $but = $mw -> Button(-text=>"Click to Create Toplevel",
                -command=>\&makeTop)  -> pack;

MainLoop;

#A function to make a toplevel window
sub makeTop {
        my $top = $mw -> Toplevel(); #Make the window
        #Put things in it
        my $top_lab = $top -> Label(-text=>"This is the Toplevel window",
                        -font=>"ansi 12 bold") -> pack;
        my $txt = $top -> Text() -> pack;
        $txt -> insert('end', "Widgets can be packed in this window");

        #An option to close the window
        my $but_close = $top -> Button(-text=>"Close",
                -command => sub { destroy $top; } ) -> pack;
}
```

Widgets 3 : Radiobutton, Checkbutton

Radiobutton

Radiobutton is an input where any one of many choices MUST be chosen If one is chosen and another button is clicked, the last chosen will lose its state and the clicked button will be chosen A graphic example(in HTML) is given below

Choices ⦿ 1 | ○ 2 | ○ 3

Some Options

-command=>COMMAND	Specifies a command to associate with the button This command is typically invoked when mouse button 1 is released over the button window
-variable => \$VARIABLE	Specifies name of global variable to set to indicate whether or not this button is selected
-value => VALUE	Specifies value to store in the button's associated variable whenever this button is selected

Syntax	Description	Example
$widget -> deselect();	Deselects the checkbutton and sets the associated variable to its "off" value	$rdb_m -> deselect();
$widget -> select()	Selects the checkbutton and sets the associated variable to its "on" value	$rdb_m -> select();

Example

```
#!/usr/local/bin/perl
use Tk;

#Global Variables
my $age = 10;
my $gender = "Male";

# Main Window
my $mw = new MainWindow;

#GUI Building Area
my $frm_name = $mw -> Frame();
my $lab = $frm_name -> Label(-text=>"Name:");
my $ent = $frm_name -> Entry();
#Age
my $scl = $mw -> Scale(-label=>"Age :",
         -orient=>'v',          -digit=>1,
         -from=>10,             -to=>50,
         -variable=>\$age,      -tickinterval=>10);

#Gender
my $frm_gender = $mw -> Frame();
my $lbl_gender = $frm_gender -> Label(-text=>"Sex ");
my $rdb_m = $frm_gender -> Radiobutton(-text=>"Male",
                -value=>"Male",  -variable=>\$gender);
my $rdb_f = $frm_gender -> Radiobutton(-text=>"Female",
                -value=>"Female",-variable=>\$gender);


my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button);

#Text Area
my $textarea = $mw -> Frame();
```

```
my $txt = $textarea -> Text(-width=>40, -height=>10);
my $srl_y = $textarea -> Scrollbar(-orient=>'v',-command=>[yview => $txt]);
my $srl_x = $textarea -> Scrollbar(-orient=>'h',-command=>[xview => $txt]);
$txt -> configure(-yscrollcommand=>['set', $srl_y],
                -xscrollcommand=>['set',$srl_x]);

#Geometry Management
$lab -> grid(-row=>1,-column=>1);
$ent -> grid(-row=>1,-column=>2);
$scl -> grid(-row=>2,-column=>1);
$frm_name -> grid(-row=>1,-column=>1,-columnspan=>2);

$lbl_gender -> grid(-row=>1,-column=>1);
$rdb_m -> grid(-row=>1,-column=>2);
$rdb_f -> grid(-row=>1,-column=>3);
$frm_gender -> grid(-row=>3,-column=>1,-columnspan=>2);

$but -> grid(-row=>4,-column=>1,-columnspan=>2);

$txt -> grid(-row=>1,-column=>1);
$srl_y -> grid(-row=>1,-column=>2,-sticky=>"ns");
$srl_x -> grid(-row=>2,-column=>1,-sticky=>"ew");
$textarea -> grid(-row=>5,-column=>1,-columnspan=>2);

MainLoop;

## Functions
#This function will be executed when the button is pushed
sub push_button {
        my $name = $ent -> get();
        $txt -> insert('end',"$name\($gender\) is $age years old");
}
```

This time the program is subjected to even more change - the geometry manager is fully grid now There is no instances of pack You will find this necessary when the layout becomes more complicated I hope you can stay with me in such trying times

---

Checkbutton

Checkbotton is a input with two options - Off or On - it has to be either one The state can be changed by clicking on it An example is shown below

check box

Some Options

-offvalue=>VALUE	Specifies value to store in the button's associated variable whenever this button is deselected Defaults to ``0''
-onvalue=>VALUE	Specifies value to store in the button's associated variable whenever this button is selected Defaults to ``1''
-command=>CALLBACK	Specifies a command to associate with the button This command is typically invoked when mouse button 1 is released over the button window
-variable=>\$VARABLE	Specifies name of global variable to set to indicate whether or not this button is selected

Syntax	**Description**	**Example**
$widget -> deselect();	Deselects the checkbutton and sets the associated variable to its ``off'' value	$chk -> deselect();
$widget -> select();	Selects the checkbutton and sets the associated variable to its ``on'' value	$chk -> select();

$widget -> toggle();	Toggles the selection state of the button, redisplaying it and modifying its associated variable to reflect the new state	$chk -> toggle();

Example

```
#!/usr/local/bin/perl
use Tk;

#Global Variables
my $age = 10;
my $gender = "Male";
my $occupied = 1;

# Main Window
my $mw = new MainWindow;

#GUI Building Area
my $frm_name = $mw -> Frame();
my $lab = $frm_name -> Label(-text=>"Name:");
my $ent = $frm_name -> Entry();
#Age
my $scl = $mw -> Scale(-label=>"Age :",
         -orient=>'v',          -digit=>1,
         -from=>10,             -to=>50,
         -variable=>\$age,      -tickinterval=>10);

#Jobs
my $chk = $mw -> Checkbutton(-text=>"Occupied",
        -variable=>\$occupied);
$chk -> deselect();

#Gender
my $frm_gender = $mw -> Frame();
my $lbl_gender = $frm_gender -> Label(-text=>"Sex ");
my $rdb_m = $frm_gender -> Radiobutton(-text=>"Male",
                -value=>"Male",  -variable=>\$gender);
my $rdb_f = $frm_gender -> Radiobutton(-text=>"Female",
                -value=>"Female",-variable=>\$gender);

my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button);

#Text Area
my $textarea = $mw -> Frame();
my $txt = $textarea -> Text(-width=>40, -height=>10);
my $srl_y = $textarea -> Scrollbar(-orient=>'v',-command=>[yview => $txt]);
my $srl_x = $textarea -> Scrollbar(-orient=>'h',-command=>[xview => $txt]);
$txt -> configure(-yscrollcommand=>['set', $srl_y],
                -xscrollcommand=>['set',$srl_x]);

#Geometry Management
$lab -> grid(-row=>1,-column=>1);
$ent -> grid(-row=>1,-column=>2);
$frm_name -> grid(-row=>1,-column=>1,-columnspan=>2);

$scl -> grid(-row=>2,-column=>1);
$chk -> grid(-row=>2,-column=>2,-sticky=>'w');

$lbl_gender -> grid(-row=>1,-column=>1);
$rdb_m -> grid(-row=>1,-column=>2);
$rdb_f -> grid(-row=>1,-column=>3);
$frm_gender -> grid(-row=>3,-column=>1,-columnspan=>2);

$but -> grid(-row=>4,-column=>1,-columnspan=>2);

$txt -> grid(-row=>1,-column=>1);
$srl_y -> grid(-row=>1,-column=>2,-sticky=>"ns");
```

```
$srl_x -> grid(-row=>2,-column=>1,-sticky=>"ew");
$textarea -> grid(-row=>5,-column=>1,-columnspan=>2);

MainLoop;

## Functions
#This function will be executed when the button is pushed
sub push_button {
        my $name = $ent -> get();
        $txt -> insert('end',"$name\($gender\) is $age years old");
}
```

Widgets 4 : Listbox

Listbox

A listbox is a widget that displays a list of strings, one per line When first created, a new listbox has no elements Elements may be added or deleted using widget commands described below

Some Options

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

Some Commands

Syntax	**Description**	**Example**
$widget -> curselection();	Returns a list containing the numerical indices of all of the elements in the listbox that are currently selected If there are no elements selected in the listbox then an empty string is returned	$sel = $lst -> curselection();
$widget -> delete(*first,?last?*);	Deletes one or more elements of the listbox First and last are indices specifying the first and last elements in the range to delete If last isn't specified it defaults to first, ie a single element is deleted	$lst -> delete(5);
$widget -> get(*first,?last?*);	If last is omitted, returns the contents of the listbox element indicated by first, or an empty string if first refers to a non-existent element If last is specified, the command returns a list whose elements are all of the listbox elements between first and last, inclusive	$lst -> get(5,end);
$widget -> index(*index*);	Returns the integer index value that corresponds to *index* If *index* is end the return value is a count of the number of elements in the listbox (not the index of the last element)	$lst -> index(5);
$widget -> insert(*index,? element element ?*);	Inserts zero or more new elements in the list just before the element given by *index* If *index* is specified as end then the new elements are added to the end of the list Returns an empty string	$lst -> insert('end',"me");
$widget -> size();	Returns a decimal string indicating the total number of elements in the listbox	$count = $lst -> size();

Example

```
#!/usr/local/bin/perl
use Tk;

#Global Variables
my $age = 10;
my $occupied = 1;
my $gender = "Male";

# Main Window
my $mw = new MainWindow;

#GUI Building Area
my $frm_name = $mw -> Frame();
my $lab = $frm_name -> Label(-text=>"Name:");
my $ent = $frm_name -> Entry();
#Age
my $scl = $mw -> Scale(-label=>"Age :",
         -orient=>'v',          -digit=>1,
         -from=>10,                     -to=>50,
         -variable=>\$age,      -tickinterval=>10);

#Jobs
my $frm_job = $mw -> Frame();
my $chk = $frm_job -> Checkbutton(-text=>"Occupied",
        -variable=>\$occupied);
$chk -> deselect();
my $lst = $frm_job -> Listbox(-selectmode=>'single');

#Adding jobs
$lst -> insert('end',"Student","Teacher","Clerk","Business Man",
        "Militry Personal","Computer Expert","Others");

#Gender
my $frm_gender = $mw -> Frame();
my $lbl_gender = $frm_gender -> Label(-text=>"Sex ");
my $rdb_m = $frm_gender -> Radiobutton(-text=>"Male",
                -value=>"Male",  -variable=>\$gender);
my $rdb_f = $frm_gender -> Radiobutton(-text=>"Female",
                -value=>"Female",-variable=>\$gender);

my $but = $mw -> Button(-text=>"Push Me", -command =>\&push_button);

#Text Area
my $textarea = $mw -> Frame();
my $txt = $textarea -> Text(-width=>40, -height=>10);
my $srl_y = $textarea -> Scrollbar(-orient=>'v',-command=>[yview => $txt]);
my $srl_x = $textarea -> Scrollbar(-orient=>'h',-command=>[xview => $txt]);
$txt -> configure(-yscrollcommand=>['set', $srl_y],
                -xscrollcommand=>['set',$srl_x]);

#Geometry Management
$lab -> grid(-row=>1,-column=>1);
$ent -> grid(-row=>1,-column=>2);
$scl -> grid(-row=>2,-column=>1);
$frm_name -> grid(-row=>1,-column=>1,-columnspan=>2);

$chk -> grid(-row=>1,-column=>1,-sticky=>'w');
$lst -> grid(-row=>2,-column=>1);
$frm_job -> grid(-row=>2,-column=>2);

$lbl_gender -> grid(-row=>1,-column=>1);
$rdb_m -> grid(-row=>1,-column=>2);
$rdb_f -> grid(-row=>1,-column=>3);
$frm_gender -> grid(-row=>3,-column=>1,-columnspan=>2);

$but -> grid(-row=>4,-column=>1,-columnspan=>2);

$txt -> grid(-row=>1,-column=>1);
$srl_y -> grid(-row=>1,-column=>2,-sticky=>"ns");
```

```
$srl_x -> grid(-row=>2,-column=>1,-sticky=>"ew");
$textarea -> grid(-row=>5,-column=>1,-columnspan=>2);

MainLoop;

## Functions
#This function will be executed when the button is pushed
sub push_button {
        my $name = $ent -> get();
        $txt -> insert('end',"$name\($gender\) is $age years old and is ");

        my $job = "";
        #See whether he is employed
        if ( $occupied == 1 ) {
                my $job_id = $lst -> curselection(); #Get the no of selected jobs
                if ( $job_id eq "" ) { #If there is no job
                        $job = "a Non worker";
                }
                else {
                        $job = $lst -> get($job_id) ;#Get the name of the job
                        $txt -> insert('end',"a $job");
                }
        }
        else {
                $txt -> insert('end',"unemployed");
        }
}
```

Wow! Our 'little' example is a big (and utterly pointless) program now I am going to stop 'exampling' from now on This is quite complicated isn't it? Why don't you run the script and see what a beautiful script we made Copy the above script and paste it in a file called "infopl" and double click the file Voila! We are Perl/Tk programmers

Widgets 5 : Menubutton, Menu, Optionmenu

Menubutton

A menubutton is a widget that displays a textual string, bitmap, or image and is associated with a menu widgetIn normal usage, pressing left-clicking the menubutton causes the associated menu to be posted just underneath the menubutton

Some Options

-direction => DIRECTION	Specifies where the menu is going to be popup up above tries to pop the menu above the menubutton below tries to pop the menu below the menubutton left tries to pop the menu to the left of the menubutton right tries to pop the menu to the right of the menu button flush pops the menu directly over the menubutton
-menu => NAME	Specifies the path name of the menu associated with this menubutton The menu must be a child of the menubutton

---

Menu

A menu is a widget that displays a collection of one-line entries arranged in one or more columns There exist several different types of entries, each with different properties Entries of different types may be combined in a single menu Menu entries are not the same as entry widgets In fact, menu entries are not even distinct widgets; the entire menu is one widget

Some Options

-tearoff => BOOLEAN	This option must have a proper boolean value, which specifies whether or not the menu should include a tear-off entry at the top If so, it will exist as entry 0 of the menu and the other entries will number starting at 1 The default menu bindings arrange for the menu to be torn off when the tear-off entry is invoked
-title => STRING	The string will be used to title the window created when this menu is torn off If the title is NULL, then the window will have the title of the menubutton or the text of the cascade item from which this menu was invoked
-type => OPTION	This option can be one of menubar, tearoff, or normal, and is set when the menu is created While the string returned by the configuration database will change if this option is changed, this does not affect the menu widget's behavior

Some Commands

<table>
<tr><th>Syntax</th><th>Description</th></tr>
<tr><td>$widget -> TYPE(?option=>value,option=>value,?);</td><td>Add a new entry to the bottom of the menu The new entry's type is given by TYPE and must be one of cascade, checkbutton, command, radiobutton, or separator, or a unique abbreviation of one of the above If additional arguments are present, they specify any of the following options:
-accelerator => VALUE — Specifies a string to display at the right side of the menu entry Normally describes an accelerator keystroke sequence that may be typed to invoke the same function as the menu entry This option is not available for separator or tear-off entries
-columnbreak => VALUE — When this option is zero, the appears below the previous entry When this option is one, the menu appears at the top of a new column in the menu
-label => VALUE — Specifies a string to display as an identifying label in the menu entry Not available for separator or tear-off entries
-compound => VALUE — Specifies whether the menu entry should display both an image and text, and if so, where the image should be placed relative to the text Valid values for this option are bottom, center, left, none, right and top
-image => VALUE — Specifies an image to display in the menu instead of a text string or bitmap The image must have been created by some previous invocation of image create This option overrides the -label and -bitmap options but may be reset to an empty string to enable a textual or bitmap label to be displayed This option is not available for separator or tear-off entries
-underline => VALUE — Specifies the integer index of a character to underline in the entry This option is used to make keyboard shortcuts 0 corresponds to the first character of the text displayed in the entry, 1 to the next character, and so on</td></tr>
<tr><td>$widget -> delete(index1, ?index2?);</td><td>Delete all of the menu entries between index1 and index2 inclusive If index2 is omitted then it defaults to index1 Attempts to delete a tear-off menu entry are ignored (instead, you should change the tearOff option to remove the tear-off entry)</td></tr>
<tr><td>$widget -> insert(index,type, ? option=>value ?);</td><td>Same as the add widget command except that it inserts the new entry just before the entry given by index, instead of appending to the end of the menu The type, option, and value arguments have the same interpretation as for the add widget command It is not possible to insert new menu entries before the tear-off entry, if the menu has one</td></tr>
</table>

Example

```
#!/usr/local/bin/perl
use Tk;
# Main Window
my $mw = new MainWindow;

#Making a text area
my $txt = $mw -> Scrolled('Text',-width => 50,-scrollbars=>'e') -> pack ();

#Declare that there is a menu
my $mbar = $mw -> Menu();
$mw -> configure(-menu => $mbar);

#The Main Buttons
my $file = $mbar -> cascade(-label=>"File", -underline=>0, -tearoff => 0);
my $others = $mbar -> cascade(-label =>"Others", -underline=>0, -tearoff => 0);
my $help = $mbar -> cascade(-label =>"Help", -underline=>0, -tearoff => 0);

## File Menu ##
$file -> command(-label => "New", -underline=>0,
                -command=>sub { $txt -> delete('10','end');} );
$file -> checkbutton(-label =>"Open", -underline => 0,
                -command => [\&menuClicked, "Open"]);
$file -> command(-label =>"Save", -underline => 0,
                -command => [\&menuClicked, "Save"]);
$file -> separator();
$file -> command(-label =>"Exit", -underline => 1,
                -command => sub { exit } );

## Others Menu ##
my $insert = $others -> cascade(-label =>"Insert", -underline => 0, -tearoff => 0);
$insert -> command(-label =>"Name",
        -command => sub { $txt->insert('end',"Name : Binny V A\n");});
$insert -> command(-label =>"Website", -command=>sub {
        $txt->insert('end',"Website : http://wwwgeocitiescom/binnyva/\n");});
$insert -> command(-label =>"Email",
        -command=> sub {$txt->insert('end',"E-Mail : binnyva\@hotmailcom\n");});
$others -> command(-label =>"Insert All", -underline => 7,
        -command => sub { $txt->insert('end',"Name : Binny V A
Website : http://wwwgeocitiescom/binnyva/
E-Mail : binnyva\@hotmailcom");
        });

## Help ##
$help -> command(-label =>"About", -command => sub {
        $txt->delete('10','end');
        $txt->insert('end',
        "About
----------
This script was created to make a menu for a\nPerl/Tk tutorial
Made by Binny V A
Website : http://wwwgeocitiescom/binnyva/code
E-Mail : binnyva\@hotmailcom"); });

MainLoop;

sub menuClicked {
        my ($opt) = @_;
        $mw->messageBox(-message=>"You have clicked $opt
This function is not implanted yet");
}
```

Create the main buttons as cascade menus and create the menus as their slaves For more information see the manual

---

Optionmenu

Makes a button, which when clicked on shows a list with available options Useful when user has to make one choice when multiple choices are given Below is a options menu in HTML A word of caution though - Perl/Tk's option menu has a very different appearance

Go

Syntax

```
my $widget = $mw -> Optionmenu(?option=>value,option=>value,?);
```

Options

Syntax	Description
-options=>OPTIONS	(Re)sets the list of options presented
-command=>CALLBACK	Defines the callback that is invokes when a new option is selected
-variable=>\$VARIABLE	Reference to a scalar that contains the current value of the selected option

Methords

Syntax	Description	Example
$widget -> addOptions(*[Option1=>Value1], ? [Option2=>Value2]?*);	Adds newly given options to the already available options	$opt->addOptions([May=>5], [June=>6], [July=>7], [Augest=>8]);

Example

```
#!/usr/local/bin/perl
use Tk;
# Main Window
$mw = new MainWindow;

my $var;
my $opt = $mw -> Optionmenu(-options => [qw(January February March April)],
        -command => sub { print "got: ", shift, "\n" },
        -variable => \$var,
        )->pack;
$opt->addOptions([May=>5],[June=>6],[July=>7],[Augest=>8]);

$mw->Label(-textvariable=>\$var, -relief=>'groove')->pack;
$mw->Button(-text=>'Exit', -command=>sub{$mw->destroy})->pack;

MainLoop;
```

Some more Widgets - Canvas, Message, Adjuster, Scrolled

Canvas

The canvas widget is a very important widget as all points are addressable graphical drawing area Canvas widgets implement structured graphics A canvas displays any number of items, which may be things like rectangles, circles, lines, and text Items may be manipulated (eg moved or re-colored) and commands may be associated with items So if you don't like the paint program in windows, you can make your own program using this widget

The command *$widget* `-> create` *type options* is used to make different structures A few examples are given below For more information read the manual

Example

```
#!/usr/local/bin/perl
use Tk;
# Main Window
my $mw = new MainWindow;

my $cns = $mw -> Canvas(-relief=>"sunken", -background=>"blue");
$cns -> create('polygon',5,100,50,5,150,5,200,100,5,100,
        -joinstyle=>"bevel", -fill=>"red", -outline=>"white", -width=>5);
$cns -> create('oval',200,100,300,200, -fill=>"green");
$cns -> create('oval',100,150,300,100, -fill=>"white", -width=>0);
$cns -> create('rectangle',10,150,100,250, -dash=>[6,4,2,4,2,4]);
$cns -> pack;

MainLoop;
```

Message

A message is a widget that displays a textual string Much like the label widget but this can be used to make a multi-line text

The *-justify* option specifies how to justify lines of text Must be one of **left**, **center**, or **right** Defaults to **left** This option works together with the anchor, aspect, padX, padY, and width options to provide a variety of arrangements of the text within the window

---

Adjuster

An adjuster acts like the frame widget - with one notable exception The borders can be dragged and expended This widget contains any number of panes, arranged horizontally or vertically, according to the value of the *-orient* option Each pane contains one widget, and each pair of panes is separated by a movable sash Moving a sash can be done by dragging it This causes the widgets on either side of the sash to be resized

Some Options

-side=>DIRECTION	Specifies the side on which the managed widget lies relative to the Adjuster In conjunction with the pack geometry manager, this relates to the side of the master against which the managed widget and the Adjuster are packed Must be **left, right, top**, or **bottom** Defaults to **top**

Some Methods

$adjuster -> packAfter($widget, ?*pack_options*?)	This command configures the Adjuster's -widget and -side options respectively to '$widget' and the -side value specified in pack_options (top if not specified) It then packs the Adjuster after '$widget', with -fill set to **x** or **y** as appropriate

Example

```
use Tk;
use Tk::Adjuster;
my $mw = new MainWindow;

my $adj = $mw -> Adjuster();
my $lst = $mw -> Listbox();
```

以上内容仅为本文档的试下载部分，为可阅读页数的一半内容。如要下载或阅读全文，请访问：https://d.book118.com/478106025075006063