

Notes from Well House Consultants

These notes are written by Well House Consultants and distributed under their Open Training Notes License. If a copy of this license is not supplied at the end of these notes, please visit

*<http://www.wellho.net/net/whcotnl.html>
for details.*

1.1 Well House Consultants

Well House Consultants provides niche training, primarily but not exclusively in Open Source programming languages. We offer public courses at our training centre and private courses at your offices. We also make some of our training notes available under our "Open Training Notes" license, such as we're doing in this document here.

1.2 Open Training Notes License

With an "Open Training Notes License", for which we make no charge, you're allowed to print, use and distribute these notes provided that you retain the complete and unaltered license agreement with them, including our copyright statement. This means that you can learn from the notes, and have others learn from them too.

You are NOT allowed to charge (directly or indirectly) for the copying or distribution of these notes, nor are you allowed to charge for presentations making any use of them.

1.3 Courses presented by the author

If you would like us to attend a course (Java, Perl, Python, PHP, Tcl/Tk, MySQL or Linux) presented by the author of these notes, please see our public course schedule at

<http://www.wellho.net/course/index.html>

If you have a group of 4 or more trainees who require the same course at the same time, it will cost you less to have us run a private course for you. Please visit our onsite training page at

<http://www.wellho.net/course/otc.html>

which will give you details and costing information

1.4 Contact Details

Well House Consultants may be found online at

<http://www.wellho.net>

graham@wellho.net

technical contact

lisa@wellho.net

administration contact

Our full postal address is

404 The Spa
Melksham
Wiltshire
UK SN12 6QL

Phone +44 (0) 1225 708225

Fax +44 (0) 1225 707126

Introduction to Tk

Would you like to access your Tcl script through a Graphic User Interface? The Tk GUI, which is distributed with most releases of Tcl, provides such a capability. In this module we introduce you to Tk and see a couple of short examples of Tk in use.

<i>Hello Tcl/Tk World.</i>	<i>2894</i>
<i>A real application</i>	<i>2899</i>
<i>A GUI front end to a data file.</i>	<i>2900</i>

Tk stands for "tool kit". It's a Graphic User Interface (GUI) that's a Tcl application; it adds commands to Tcl, which makes it more user-friendly.

Tk is a superb GUI for the programmer who needs to knock up an interface to something quickly; there's still a lot to learn about in Tk, but it's much quicker and easier to program than X Windows or Motif or other GUI systems. In fact, it's so good that it's one of the main reasons that Tcl is so popular.

222.1 Hello Tcl/Tk World

Let's write the simplest of Tk applications, a graphics "Hello World". Our program file contains the following:

```
#!/usr/bin/wish

# Tcl/Tk - Report Date Information

button .action -text "Press for timestamp" -command {
    puts -nonewline "It is "
    puts [clock format [clock seconds]]
}
button .final -text exit -command exit

pack .action .final
```

Each time you click your mouse on the "Press for Timestamp" button, a timestamp is printed in the window from which you ran the program.

Figure 987 *tk_hello displayed*

When you select the button labelled "exit", the graphic window exits and your prompt returns on the window from which you're running:

```
[graham@chapatti folder]#tk_hello
It is Sat Oct 21 17:57:24 BST 2000
It is Sat Oct 21 17:58:03 BST 2000
It is Sat Oct 21 17:58:19 BST 2000
[graham@chapatti folder]$
```

All the other good window stuff happens too. When you resize your window, the buttons are re-laid out and redrawn. When you move over a button, it's highlighted. And when you actually select a button, it's highlighted further. When you pull another window to the front, partly hiding your Tcl/Tk window, then bring your Tcl/Tk window to the front again, it comes up correctly.

Figure 988 An example of our tk window overlapping another window and being overlaid by a third



If you were programming in raw X Windows code, for example, you would have to program all these things! With Tcl/Tk, they're done for you using the power built into Tk, and the underlying facilities that Tk Calls.

Tcl/Tk supported platforms

Windows systems differ from platform to platform. With most graphics-based software, you'll find you need to have a specific executable file to work with the particular windowing system that you're using, or emulation software to allow you to work cross-platform. That's why you have products such as X-Ceed to allow you to run X Windows on a PC with a Microsoft operating system, and Virtual PC to run Microsoft Windows (or Dos or Linux) programs on a Macintosh.

Tcl/Tk is much more portable. You can run it on your system, provided that:

- Tcl and the Tk extensions have been implemented on that platform
- You have a copy of the appropriate executable for that platform
- Other necessary libraries are also present and available

Once you have the platform-specific requirements met, you can run most Tcl/Tk programs, even cross-platform, without hassle. You do need to take care in certain areas, however. For example:

- End-of-line characters in data files
- Operating system-specific calls which you make directly from your Tcl

Tcl/Tk is implemented and runs on the following modern systems:

- PCs running Windows (98, 2000, NT, XP) and no additions or options are required to the OS
- Macintosh running MacOS
- Most Unix systems; however, the system needs to be running and have available the X Windows System, and the Motif libraries on top of that
- Most Linux systems. The system needs to be running and have available the X Windows System, and the Motif libraries on top of that

Linux and Unix systems that are shipped these days include X Windows and Motif. In unusual circumstances, though, they may not have been loaded onto your system. For example, if your Linux box is purely a firewall system that's been customised for you, without keyboard or screen, X windows may not be present.

Sourcing Tcl/Tk

Whatever platform you're running on, you need to have a Tcl/Tk program. Such can be downloaded from the Web, or loaded from CDs from the back of various course text books. Tcl/Tk is distributed under an open source license, so there are no fees to pay and, provided that you retain the copyright, you can distribute it onwards

yourself. It comes as a standard part of the CD set with Linux distributions, but may not load if you choose the leanest of configurations.

As you learnt Tcl (as opposed to Tcl/Tk), you were probably using a program called Tcl. This is Dr. John Ousterhout's Tool Control Language – a library of commands which can be built into other applications, of which Tcl/Tk is one. The Tcl program itself does not support the Tk extensions:

```
[graham@chapatti folder]$tcl
tcl>button .final -text exit -command exit
Error: invalid command name "button"
tcl>
```

Neither does the expect application:

```
[graham@chapatti folder]$expect
expect1.1>button .final -text exit -command exit
invalid command name "button"
    while executing
    "button .final -text exit -command exit"
expect1.2>
```

For Tcl/Tk, you'll be looking for a program called "wish" – the Windowing Interactive Shell. If you're wanting the Expect application too, then you'll probably be looking for a program called "expectk".

Wish and Expectk include:

- The Tcl command set
- An operating system shell so that unknown commands are passed on to the OS
- The Tk commands

Expectk also includes the expect extension to Tcl.

Throughout this module, we've used wish but we could equally have used expectk.

Our sample program

Let's take a line-by-line look at the first "tk_hello" program that we showed you:

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

We're telling the operating system and controlling program that this file of commands is to be read into, and interpreted by, the wish program.

Additionally:

- The path set correctly
- File permissions set
- you can just type in `tk_hello` and have it run!

Figure 989 Running wish interactively

Let's run wish interactively to see what it does:

```
[graham@chapatti folder]$wish
%
```

And up pops a window (see Figure 989).

```
# Tcl/Tk - Report Date Information

The null command - a comment - Tcl.

button .action -text "Press for timestamp" -command {
    puts -nonewline "It is "
    puts [clock format [clock seconds]]
}
```

The `button` command defines a Tk button. The button will be called `.action` (notice the leading dot, which is necessary) and will have the text "Press for timestamp" on it when it's displayed. When the button is pressed, the command block will be run.

Figure 990 The button definition isn't actually displayed yet

Although the button definition is made, it's not actually displayed yet as Tk hasn't been told how and where to display it; the information is simply held within Tk.

Our display window is unchanged (see Figure 990).

```
button .final -text exit -command exit
```

A second button is defined; its name is `.final` it will bear the text "exit" when (or if?) it's displayed. If it's selected, it will run the `exit` command. Again, the graphics window display doesn't change.

Our interactive session now reads:

```
[graham@chapatti folder]wish
% button .action -text "Press for timestamp" -command {
    puts -nonewline "It is "
    puts [clock format [clock seconds]]
}

.action
% button .final -text exit -command exit
.final
%
```

The button commands have echoed back the name of the button they've created,¹ but otherwise, the buttons seem to have disappeared into thin air.

```
pack .action .final
```

The `pack` command tells Tk to display the buttons (and other widgets – widgets being a generic name for windows components that can have a visible presence in a similar way to a button).

¹ we'll save these names into variables later on

Each named button is packed into the window, starting at the top, in the order that they're listed, and the display is updated. You'll notice that the window has reduced in size. Once the `pack` command has been told what you want displayed, it's able to adjust the window to fit and that's what it has done. Had you requested different text on the buttons, the size of the overall window would have been different after packing.

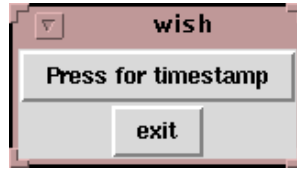


Figure 991 Updating the display

There is a subtle difference here between interactive and non-interactive modes, in that you won't see a large blank window flash up each time you start a wish program, just to see it shrink away again. The large window is an interactive feedback tool and, in a running program, all the Tcl/Tk requests get bundled up into a single display request. The final result is the same, but the way of getting there is more efficient.

That's it. That's the end of our Tcl/Tk commands. The application is running.

You can click on the "Press for timestamp" button and get a timestamp back in the window you're running from. You can press on the "exit" button, and exit from the program. There are no more Tcl/Tk commands to type in.

Here's the complete interactive session:

```
[graham@chapatti folder]wish
% button .action -text "Press for timestamp" -command {
    puts -nonewline "It is "
    puts [clock format [clock seconds]]
}
.action
% button .final -text exit -command exit
.final
% pack .action .final
% It is Sun Oct 22 08:51:55 BST 2000
It is Sun Oct 22 08:51:56 BST 2000
It is Sun Oct 22 08:52:12 BST 2000
It is Sun Oct 22 08:52:39 BST 2000
[graham@chapatti folder]$
```

You'll note that a `%` prompt came back after you packed, and you were able to type in a further command, although you chose not to.

On a non-interactive shell, the `pack` command truly is the last thing you write in this application. Once a wish or an expectk finishes running the commands you've given it, it drops into what's called an "event loop" where it waits for things to happen. When something does happen (an event!) it runs the command that you've associated with that event.

222.2 A real application

You've seen the elements of a Tk application:

- A series of elements (widgets) is defined.
 - A geometry manager (pack) is used to arrange, manage and display those widgets
- Event handlers collect requests and perform tasks necessary.

If we enhance our previous example to provide feedback via another widget, we're on to a real application.

```
#!/usr/bin/wish

# Tcl/Tk - Report Date Information

button .action -text "Press for timestamp" -command {
    set xx [format "It is %s" [clock format [clock seconds]]]
    .result config -text $xx
}

button .final -text exit -command exit
label .result -text "Answer appears here"

pack .action .result .final
```

Yes, it is that short. Such a program in Motif and C would be three times as long, and using pure X Windows it would be 10 times as long!

Feedback via the Tk window

Our very first Tk program reported its actions to the window from which it was run, rather than to the Tk window, but we've now corrected that.

We've added a label widget to our display. The label widget is a piece of text which can be changed from within the program. It doesn't look like a select-able button and indeed it's not select-able, but otherwise it has many similar features. For example, it's sized (by default) to take the text that's assigned to it.

Figure 992 *Running the program, and the result after clicking the timestamp button*

Did you notice how the size of the label widget (and thus the size of the whole Tk window) grew when we pressed the button?

Widgets as commands

When we created a label or a button (and this applies to other widget types too), Tk also created a new command. That command has the same name as the name of the button, and it has a number of subcommands associated with the particular widget type that we've used.

```
label .result -text "Answer appears here"
```

creates a label widget called `.result` and also a command (or perhaps a proc; at the user level we don't know) called `.result`.

Once the `.result` command exists, we can run it, which we do when we want to change the text:

```
.result config -text $xx
```

This changes the configuration of the widget called `.result`, replacing the old text it bore with the contents of the variable `xx`.

222.3 A GUI front end to a data file

Let's have a look at another Tcl/Tk example, this time using a simple GUI to look up information with a data file. We're taking a web server access log file as input. It contains one line for each time a client has asked for a file from a web server, and each line is of the form:

```
samosa - - [11/Aug/2000:09:28:28 +0100] "GET /mp/ HTTP/1.0" 200 713
```

We want to be able to enter a host computer name into a text box, and have a count of the number of times a particular host has called up pages be displayed (see Figure 993). Let's see the program.

The main program

In order to have code re-usable, and to keep it follow-able and maintainable too, we've split it down into a series of procs. The main program looks absurdly simple:

```
#!/usr/bin/expectk
```

The very first line of our code calls up `expectk`. We could have used `wish` just as easily, but perhaps we plan to use `expect` to FTP or Telnet log files across into our application later?

Our procs are defined next, but we'll look at them later. Let's look at the main program first.

```
setup
```

That's a call to a proc to read in and analyse the data file – nothing to do with Tk!

```
entry .word -width 20
```

```
button .action -text search! -command scanner
```

```
label .result -text "Answer appears here"
```

Create an entry box,¹ a button which when selected runs the proc called `scanner`, and a label into which we'll be putting the answer later.

```
pack .word .action .result
```

Pack and display the three widgets.

The setup proc

Opens and reads the incoming data file, which is called `seal_log` in this example. It uses an array to count up the number of times that each host has accessed the server.

We want the `count` array to be available elsewhere in our program, so it's been declared as global.



Figure 993 Adding an ability to enter a host computer name

¹ sized so that it can hold 20 characters of typing

The complete code of setup:

```
proc setup {} {
    global counter
    set fhandle [open seal_log r]
    while {[set line [gets $fhandle]] >0} {
        set hitlist [split $line]
        set hostname [lindex $hitlist 0]
        if {[info exists counter($hostname)]} {
            incr counter($hostname)
        } else {
            set counter($hostname) 1
        }
    }
}
```

The scanner proc

The proc called scanner is run whenever the **.action** button is pressed. Once again, the array counter is declared as global (and thus counter will apply to the same array in the main program, and in any other procs where it's so declared).

When the button labelled "search!" is pressed, we want to get the text off the widget labelled *.word*. Although variables are local, proc names are not; thus we can write:

```
.word get
```

to return to us the contents of the text box, which we save into a variable as we'll be using it a number of times:

```
set textval [.word get]
```

There are three possibilities we want to consider with the text we have read.

- If the word "quit", "end" or "exit" is entered, we want to terminate the application
- If a known hostname is entered, we want to display a count of the number of times that host has accessed our web pages
- If a host name isn't in our array, we want to report that fact.

Firstly, let's deal with the quit case:

```
if {[regexp -nocase ^\((exit|end|quit)\)\$ $textval]} exit
```

Match, case insensitive, against the contents of *textval*. If it contains the word "exit" or "end" or "quit" – case insensitive and anchored at both ends – then run the **exit** command.

Next, is there an element in the counter array corresponding to the host that's been called up?

```
if {[info exists counter($textval)]} {
```

If so, format our result string:

```
set rr [format "%s: %d accesses" $textval $counter($textval)]
```

and if not, set up a different result string:

```
set rr "No accesses from $textval"
```

Finally, we alter the text on our label:

```
.result config -text $rr
```

Figure 994 Example of an unknown system

At the end of running the proc, wish (or expectk) drops back into what's called the event loop. In other words, it goes back to waiting for the next user input (or for a resize, or focus change or any one of many other events). If an exiting command was given, the proc never gets to the end; this is the way out of the program.

The complete program

Below is a listing of the complete program:

```
#!/usr/bin/expectk

proc scanner {} {
    global counter
    set textval [.word get]
    if {[regexp -nocase ^\.(exit|end|quit)\.\$ $textval]} exit
    if {[info exists counter($textval)]} {
        set rr [format "%s: %d accesses" $textval $counter($textval)]
    } else {
        set rr "No accesses from $textval"
    }
    .result config -text $rr
}

proc setup {} {
    global counter
    set fhandle [open seal_log r]
    while {[set line [gets $fhandle]] >0} {
        set hitlist [split $line]
        set hostname [lindex $hitlist 0]
        if {[info exists counter($hostname)]} {
            incr counter($hostname)
        } else {
            set counter($hostname) 1
        }
    }
}

# Tcl/Tk - Access Log File counter!

setup

entry .word -width 20
button .action -text search! -command scanner
label .result -text "Answer appears here"

pack .word .action .result
```

Exercise

1. Create and display an entry box and a button.
2. Add code so that when the user types a colour name into the entry box and presses the button, the background colour of the button changes to that colour.

Note: use the **-background** option to config to change the colour; the new colour will be displayed once you move the mouse off the button after pressing it.

License

*These notes are distributed under the **Well House Consultants Open Training Notes License**. Basically, if you distribute it and use it for free, we'll let you have it for free. If you charge for its distribution of use, we'll charge.*

3.1 Open Training Notes License

Training notes distributed under the **Well House Consultants Open Training Notes License** (WHCOTNL) may be reproduced for any purpose PROVIDE THAT:

- This License statement is retained, unaltered (save for additions to the change log) and complete.
- No charge is made for the distribution, nor for the use or application thereof. This means that you can use them to run training sessions or as support material for those sessions, but you cannot then make a charge for those training sessions.
- Alterations to the content of the document are clearly marked as being such, and a log of amendments is added below this notice.
- These notes are provided "as is" with no warranty of fitness for purpose. Whilst every attempt has been made to ensure their accuracy, no liability can be accepted for any errors of the consequences thereof.

Copyright is retained by Well House Consultants Ltd, of 404, The Spa, Melksham, Wiltshire, UK, SN12 6QL - phone number +44 (1) 1225 708225. Email contact - Graham Ellis (graham@wellho.net).

Please send any amendments and corrections to these notes to the Copyright holder - under the spirit of the Open Distribution license, we will incorporate suitable changes into future releases for the use of the community.

If you are charged for this material, or for presentation of a course (Other than by Well House Consultants) using this material, please let us know. It is a violation of the license under which this notes are distributed for such a charge to be made, except by the Copyright Holder.

If you would like Well House Consultants to use this material to present a training course for your organisation, or if you wish to attend a public course is one is available, please contact us or see our web site - <http://www.wellho.net> - for further details.

Change log
Original Version, Well House Consultants, 2004

Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____
Updated by: _____ on _____

License Ends.