GUI Scripting with Tcl/Tk

Despite the emergence of new graphical toolkits like GTK and Qt, the combination of Tcl/Tk is still the tool of choice for many script writers.

Although many Linux developers are only now discovering the combination of a scripting language and a Graphical User Interface (GUI) toolkit, this sort of development environment is not new. The largely unsung forerunner to projects like PyQt and pyGTK is Tcl/Tk, the first footprints of which can be traced back to before Linux was even created. Supported by an enthusiastic community, Tcl/Tk has been quietly and efficiently providing cross platform GUI scripting to Unix, Windows and Mac developers for many years.

The language itself is currently up to version 8.4.4.0, and the Tcl/Tk application development tool of choice, Visual Tcl, has recently been updated to version 1.6 after 2 years of development. This article looks at the language, toolkit and Visual Tcl, and shows how they can be used to produce a neat solution to a real requirement.

An Overview of Tcl/Tk

Although somewhat trampled in the stampede script writers made towards Perl when a scripting language was required to drive the emerging Internet, Tcl is still a technical match for Perl, Python or any other comparable language. Often described as "the best kept secret of the Internet", it is a free (in all the best senses of the word), full featured language driven by a byte code compiler which produces performance on a par with any of its peers. It is used in all the places other scripting languages are used: system administration, task automation, server back ends, and, as we shall shortly see, application development.

As a programming language, Tcl is exceptionally easy to learn. In contrast to the complicated feature sets and syntaxes of Python and Perl, Tcl is procedural in nature, and very straightforward. The entire syntax is described in exactly 11 rules, from which the whole language is built. Ironically, it's this simplicity which sometimes confuses people who are new to Tcl, but, no, really, it *is* that simple! An experienced programmer can learn to read Tcl scripts in 10 minutes, and write them inside an hour. A beginner doesn't take much longer.

Documentation is top rate, coming in the form of comprehensive, and very well written *man* pages. A complete HTML package of the documentation is also available.

If *man* pages are a little intimidating for the new user, a decent selection of books exist for Tcl/Tk, the pick of which probably Brent Welch's recently updated Practical Programming in Tcl and Tk from Prentice Hall PTR. Also worth a mention is the Tcler's Wiki, which is one of the largest and best supported wikis anywhere on the Internet.

Tcl philosophy centers on one idea: it's an extendable language. Most languages allow a developer to write functions and procedures, but Tcl goes much further than that. Tcl allows developers to extend the entire language with new commands and functionality, up to and including adding fundamental language structures such as object orientation. The Tk toolkit is actually just another optional extension to the Tcl language which happens to provide a whole set of Tcl commands to create, drive and control GUI widgets. Like dozens of other extensions, Tk has long been included in the Tcl core distribution and is now seen more as a part of the language than an extension of it.

The Project

In order to test drive the latest versions of Tcl/Tk and Visual Tcl, I needed a small project to develop. A personal requirement provided just the thing. Since getting a digital camera I've often wanted to quickly throw a couple of pictures onto a web page in order that friends and family can see them. A full blown web gallery generating application would be overkill; I just need the ability to select one or two image files, add a few lines of text, then have a single web page appear which I can upload to a web server. Figure 1 shows an example of the sort of page I would like to be able to quickly produce.
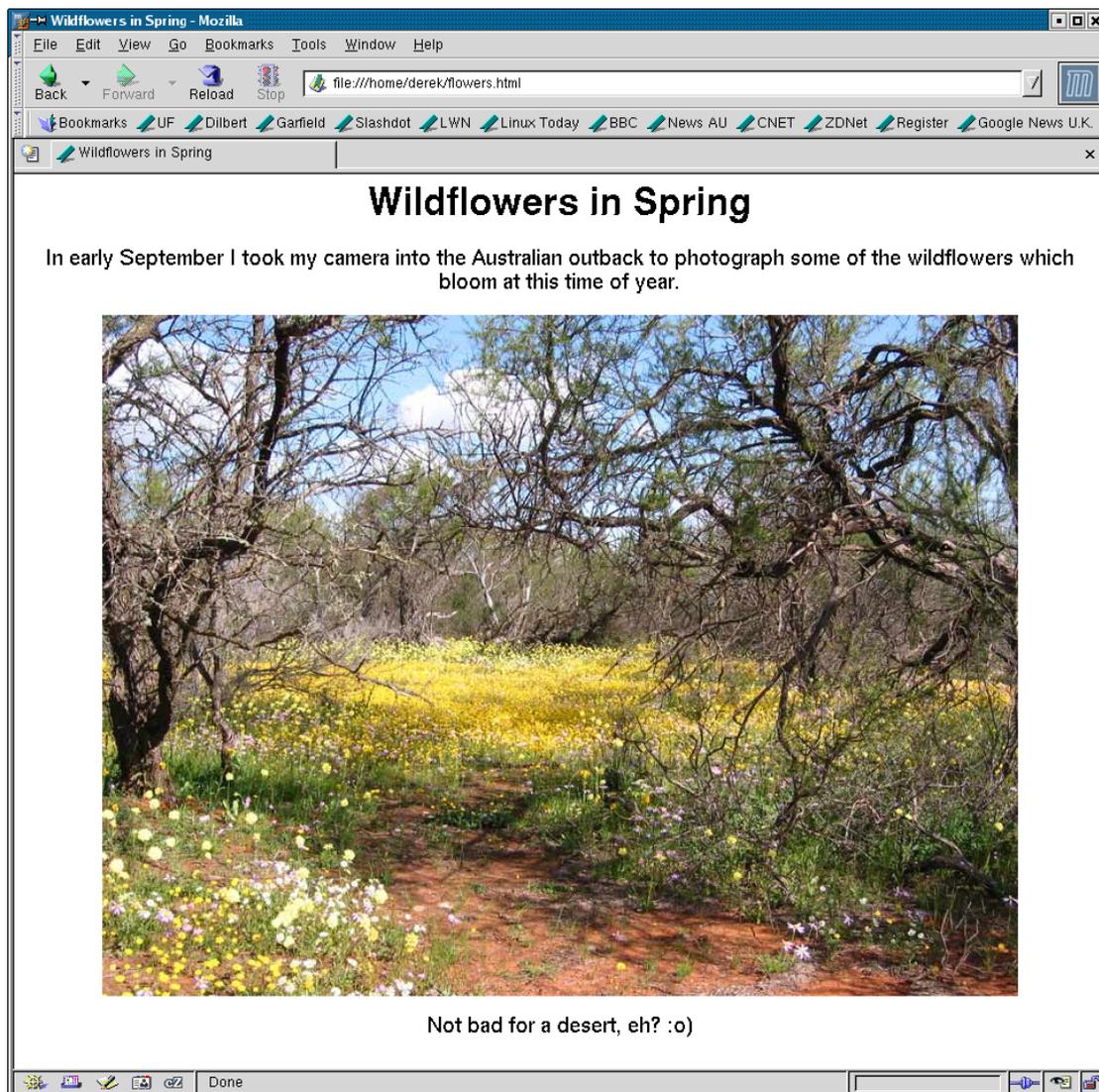
Figure 1: The task is to quickly produce simple web pages containing images and a small amount of text, like this one.

This sort of project is an ideal candidate for a GUI based script. It's a fairly simple task which isn't dependent on speed, but which will clearly benefit from having a graphical user interface. The function of the GUI is simple: present the user with an interface where they select some image files, viewing them if necessary, and collect a few lines of accompanying text. The script can then use a standard tool to produce the HTML page. In this case that tool is the XSLT processor from the libxml2 package found on just about every modern Linux system.

The rest of this article looks at how the combination of Tcl/Tk and Visual Tcl were used to rapidly develop this little application. Figure 2 shows the final script in action; the code can be downloaded from the link given at the end of this article.
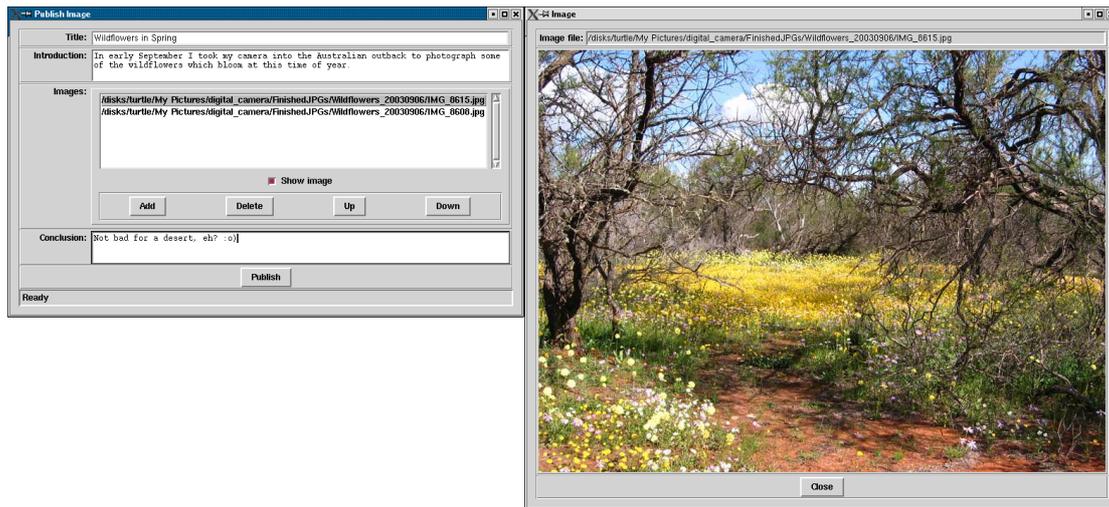
Figure 2: The script running, with the image display window open.

Getting the Software

Most Linux distributions come with Tcl/Tk. However, I always install and use the latest version of ActiveTcl from ActiveState Inc. Apart from being right up to date and professionally presented, it provides a standard Tcl package with lots of useful extensions. If you know your users are using ActiveTcl, you know exactly which extensions they have on their machine, and can therefore guarantee your script will run. I encourage anyone who wants to run the project in this article to download and install ActiveTcl-8.4.4.0 or later, since that's what I used for development. ActiveTcl comes with its own installer, and if you install it in, for example, */opt/ActiveTcl-8.4.4.0,* it won't interfere with any existing Tcl/Tk installation. If you already have a Tcl/Tk package in */usr/bin*, ensure you set an early entry in your user account's PATH to point to the ActiveTcl *bin* directory.

Visual Tcl is available from Sourceforge, and also comes with its own installer. Many Linux distributions include it, but ensure you have the latest version.

Developing a Tcl/Tk Script

A common approach to Tcl/Tk scripting is to start by designing the GUI. This process allows the developer to think through all the features which the application requires, and produces a solid framework which those features can be built upon. When things start getting complicated this approach breaks down and something more formal like a "Model, View, Controller" pattern is required. But for small applications like my one, or for rapid prototyping, getting a GUI together is a good starting point. So I'll start with Visual Tcl.

A Look at Visual Tcl

The days when developers would sit at a text editor manually arranging buttons, listboxes and other widgets by brain power alone are pretty much gone. This is the sort of job which should now be done with a graphical tool. Dragging and dropping widgets makes development much quicker, especially for beginners.

Visual Tcl provides exactly these sorts of facilities, and then some. In fact, it doesn't seem too sure whether to behave like a cut down integrated development environment (IDE) or not. It occasionally offers a text editing window where the user can write the Tcl code which forms the actual application, rather than just limiting itself to dealing with the development of the GUI. On the other hand it doesn't offer a debugger or some other traditional IDE features, so it's difficult to justify calling it a real IDE.

I dealt with this confusion of personality by going into the configuration dialog for the application and switching off many of the "features" which just seemed to get in my way. See Figure 3.
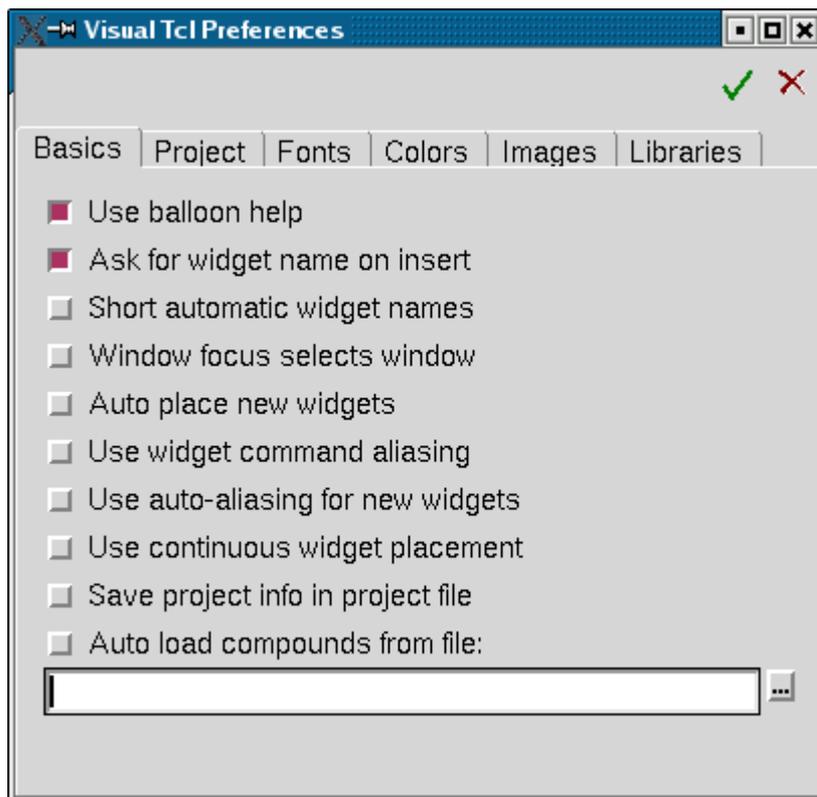

Figure 3: Visual Tcl is highly configurable.

Instead I chose to write the bulk of the application logic in my favoured environment (XEmacs) and just used the output from Visual Tcl as a library which creates the GUI for my script. Credit goes to Visual Tcl for being flexible enough to be used in the

way of my choosing. Listing 1 shows my "wrapper" script, which is the starting point for the application code itself.

```
#!/bin/sh
# the next line restarts using wish \
exec wish "$0" "$@"

#
# My own procedures and "pre-gui" code will go here
#

# Load and run the GUI code created by Visual Tcl
#
source gui.tcl

#
# Any "post-gui" code I need can go here
#
```

Listing 1: A simple wrapper to keep the Visual Tcl code (in gui.tcl) separate from the main script. The she-bang line weirdness is a very common way of starting a Tcl/Tk script.

Once I'd got to grips with the way I wanted to work with the tool, it didn't take too long to produce the output I wanted. Widgets are placed via a simple point and click interface, and a separate "Attribute Editor" window allows for the fine detail of widget behavior to be tweaked and fiddled with to the heart's content. Tk widget layout devices are also easy to control when you understand them. Figure 4 shows the Visual Tcl development environment.
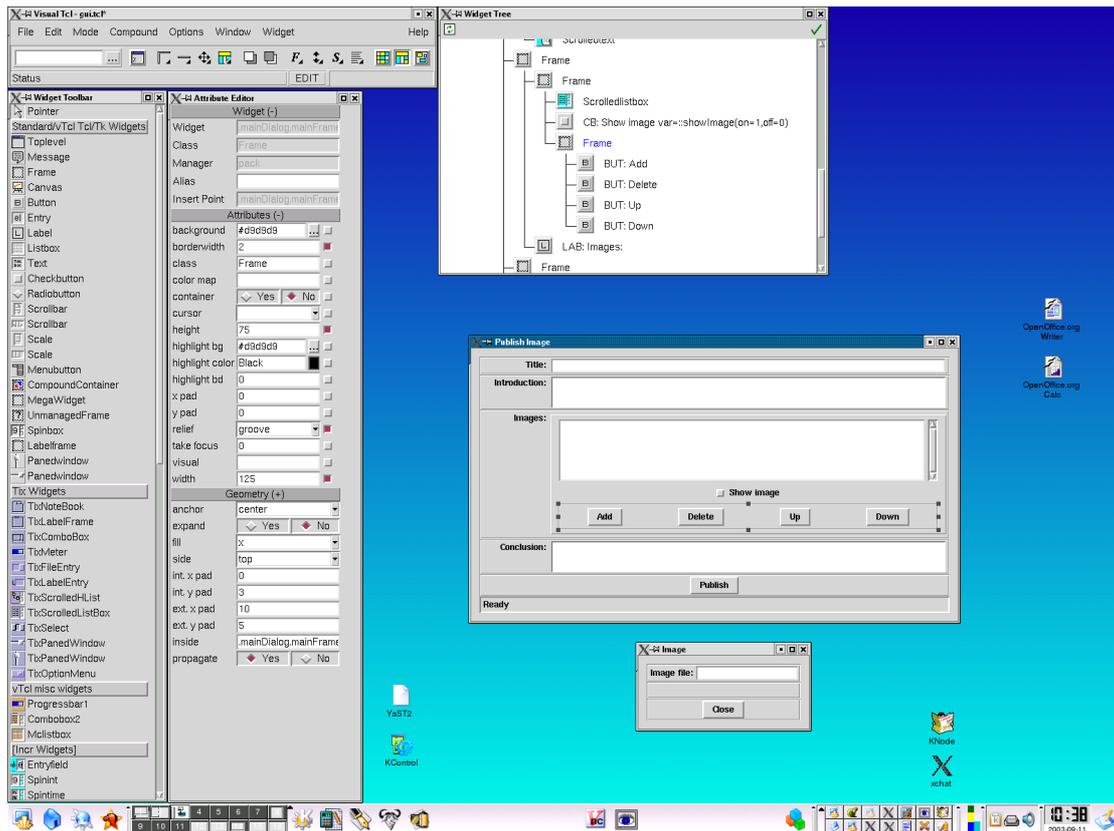
Figure 4: Visual Tcl appears rather cluttered even on a large screen. It's not too hard to use though.

Visual Tcl produces executable Tcl/Tk code which is loaded and edited directly. The routines which load the Tcl/Tk code are surprisingly tolerant, which means the generated code can be independently edited and tuned by the developer before being returned to Visual Tcl for further work.

Visual Tcl's biggest problem is the dated nature of the toolkit behind it. Tcl/Tk only offers the basic building blocks of widgets. Things like comboboxes and notebooks aren't available in Tk. Fortunately there are a number of extensions to Tcl/Tk which provide these "mega widgets", and Visual Tcl supports them all. The drawback with this is that, for the final script to run correctly, the target machine needs the mega widget extensions installed. For this project I made use of the "incr tcl" widget set, and the Tcl/Tk installed as part of most Linux distributions may not contain this. Hence my recommendation of the ActiveTcl Tcl/Tk distribution. In fact, my SUSE-8.1 system does include "incr tcl" but strangely doesn't include the extension required to load JPEG images – a rather glaring omission on the part of SUSE I'd have thought.

Anyone who has used a really slick GUI builder tool like Qt's excellent *designer* will tell you that Visual Tcl needs more work. It's slow on my dual PIII-500 machine to the

point of irritating, and has more than its share of usability issues and bugs, although these should be cleared up in the point-one release. The bottom line, though, is that Visual Tcl did the job I required of it. The script it generates is readable enough to be fine tuned by hand, and anything the code does can be overridden by more specific code in the main application. My GUI completed, I moved on to the application development side of the project.

Building the Application

The thing which still sets Tcl apart from more modern GUI scripting solutions is the way the Tk toolkit interacts with the Tcl code which does the work. Packages like GTK or Qt are low level libraries, written in C or C++. The script level bindings to them work well enough, but there's always a big step down from the scripting language into the API of the GUI toolkit. The developer needs to really understand the widgets he's working with, and must know how to configure and interrogate them using low level calls directly to the widgets themselves.

The relationship between Tcl and Tk is much more peer to peer. The GUI toolkit operates at the same level as the language driving it, which makes the combination very easy to work with.

Take, for example, the listbox widget which contains the list of images to put in the web page. In Visual Tcl an attribute of the listbox widget called the "listvar" is presented, and I set it to a variable called "::imageList". "::imageList" is a list variable in my Tcl code, and Tcl/Tk ensures that its contents are always reflected in the listbox widget. If I add, move or delete an item in that list variable, the contents of the listbox widget are immediately and automatically updated to display its contents. The code which handles the image list doesn't access or interact with the GUI at all. It just keeps a single list variable in the correct state, safe in the knowledge that Tcl/Tk will do the rest. Figure 5 shows this relationship.
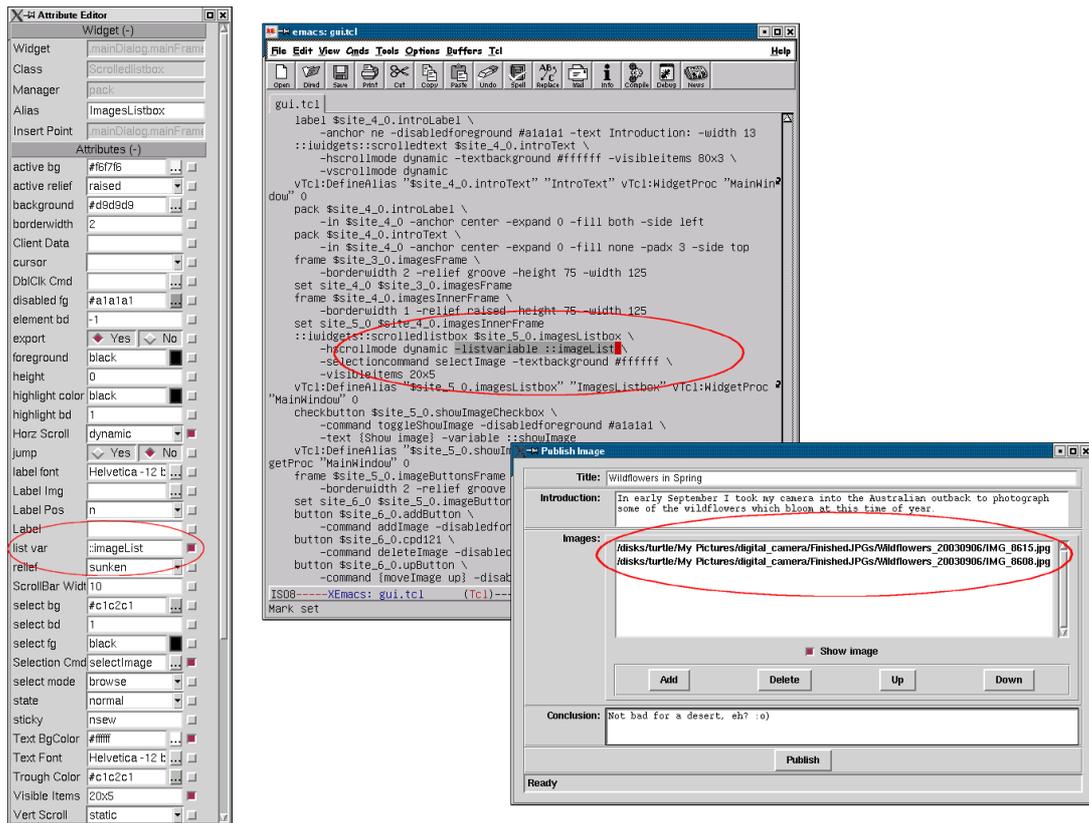
Figure 5: Setting the "listvar" attribute in Visual Tcl (left) ensures the generated code (middle) causes the onscreen widget (right) to respond immediately to any changes made in the named variable.

More direct access to the widgets is sometimes required. Under these circumstances, Visual Tcl makes use of *aliasing*. In Tcl/Tk, the name of a widget depends on where it is in the widget tree. That name will change as container widgets like frames are added and removed. To prevent the script writer having to keep track of the full names of the important widgets, Visual Tcl allows the user to specify an alias – that is a short, easily memorable name the widget is always known by. These short names can be looked up in a global associative array (also known as a hash or dictionary) so access to the widgets, wherever they might end up, is always easy. For example, I gave the "Introduction" text widget the alias "IntroText", so to fetch the text currently in that widget, the code in Listing 2 can be used.

```
...
set introWidget $::widget(IntroText)
set text [$introWidget get 0.0 end]
...
```

Listing 2: Fetching the contents of an aliased widget

The *::widget* array is provided automatically by the Visual Tcl generated code, so fetching the real name of the text widget is simple. Asking the widget to provide its current text, from line 0 character 0 to the end, is then easy.

The image display in the viewer window is actually just a label widget in the center of the dialog. Tk can load an image from disk and create a pixmap from it with one line of code. When the user selects a new image file, a pixmap is created from it and a single command is used to set the label widget to show that image. See Listing 3.

```
…
set loadedImage [image create photo –file $filename]
$::widget(ImageLabel) configure –image $loadedImage
…
```

Listing 3: The image is loaded from the disk, then the label widget is configured to show that image (Tk labels show images as well as text). The image appears on screen immediately.

In the actual script I store the loaded pixmaps in a cache. This makes switching from one image to another and back again much sharper.

When the user clicks the "Publish" button, a Tcl function is called which creates the web page. The workings of this code aren't especially relevant here; suffice to say that Tcl allows generation of an XML DOM using the TclXML extension, then allows the call out to the libxml2 XSLT processor which generates the HTML. Getting a specialist package to do the hard work is, of course, the ace up the script writer's sleeve.

The Shortcomings of Tcl/Tk

While the Tcl/Tk script works nicely, it's hard to ignore the obvious gulf in quality between the appearance of a Tcl/Tk based script and a more modern Qt or GTK based one. Qt and GTK based programs look much sharper than those using the Motif style of Tk widgets, plus they are themeable, whereas Tk isn't. Also compare "built in" features such as the file selector dialog – Tk's is no better than GTK's, and both are totally embarrassed by Qt's. Work continues in the Tcl community regarding these sorts of issues, but, as with many mature technologies, improvements are slow in coming for fear of breaking existing code.

Conclusion

Tcl/Tk is the oldest of the GUI enabled scripting languages in common use today, but it doesn't enjoy the monopoly position it used to. Python, coupled to GTK or Qt, now provides a more contemporary solution to many of the problems Tcl/Tk used to be the

natural choice for, and both Tcl/Tk and Visual Tcl have some ground to make up in terms of looks, features and desktop integration.

However, the simplicity of application development offered by the mature and superbly integrated combination of the Tcl language and the Tk toolkit is still second to none. If you have a simple scripting task which would benefit from a GUI, and where speed and cost of development are important, Tcl/Tk should still be near the very top of the list of contenders for the job.

References

Source to the script developed in this article:

    http://

Tcl/Tk headquarters:

    http://www.tcl.tk

The Tcler's Wiki:

    http://mini.net/tcl/

Tcl/Tk *man* pages, online and downloadable:

    http://www.tcl.tk/man/

ActiveState Tcl website:

    http://www.activestate.com/Products/ActiveTcl/

Visual Tcl:

    http://vtcl.sourceforge.net

Incr Tcl

    http://incrtcl.sourceforge.net/itcl/

The 11 rules of the Tcl syntax:

    http://www.tcl.tk/man/tcl8.4/TclCmd/Tcl.htm

Practical Programming in Tcl and Tk (4th ed.), by Brent Welch. Prentice Hall PTR:

    http://www.beedub.com/book/

XSLT for libxml2:

    http://www.xmlsoft.org/XSLT.html