InDepth

GUI Scripting with Tcl/Tk

Tcl/Tk may not look very modern, but it has handy features such as variables that automatically take on the value of a widget.

by Derek Fountain

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 even was created. Supported by an enthusiastic community, Tcl/Tk quietly and efficiently has been providing cross-platform GUI scripting to UNIX, Windows and Macintosh developers for many years.

The language itself currently is up to version 8.4.5.0, and the Tcl/Tk application development tool of choice, Visual Tcl, recently has been updated to version 1.6 after two years of development. This article looks at the language, toolkit and Visual Tcl and describes 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 toward Perl when a scripting language was required to drive the emerging Internet, Tcl still is 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 that 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 see shortly, 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 and straightforward in nature. The entire syntax is described in exactly 11 rules, from which the whole language is built. Ironically, it's this simplicity that sometimes confuses people who are new to Tcl. An experienced programmer can learn to read Tcl scripts in ten 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 well written man pages. A complete HTML package of the documentation also is 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 is 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. 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 actually is 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 now is 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 throw a couple of pictures onto a Web page quickly so that friends and family could see them. A full-blown Web gallery application would be overkill;

I need only the ability to select one or two image files, add a few lines of text and then have a single Web page appear that I can upload to a Web server. Figure 1 shows an example of the sort of page I would like to be able to produce.

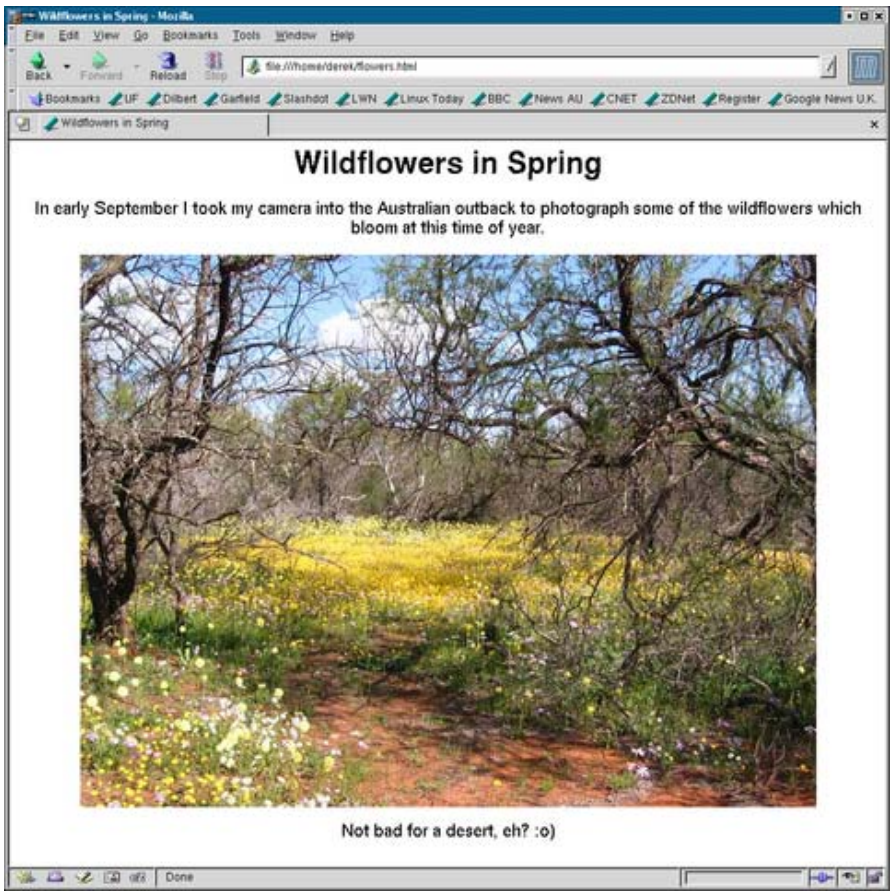

Figure 1. The task is to produce simple Web pages containing images and a small amount of text quickly, like this one.

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

The rest of this article looks at how the combination of Tcl/Tk and Visual Tcl were used to develop this little application rapidly. Figure 2 shows the final script in action; the code can be downloaded from the link provided at the end of this article.

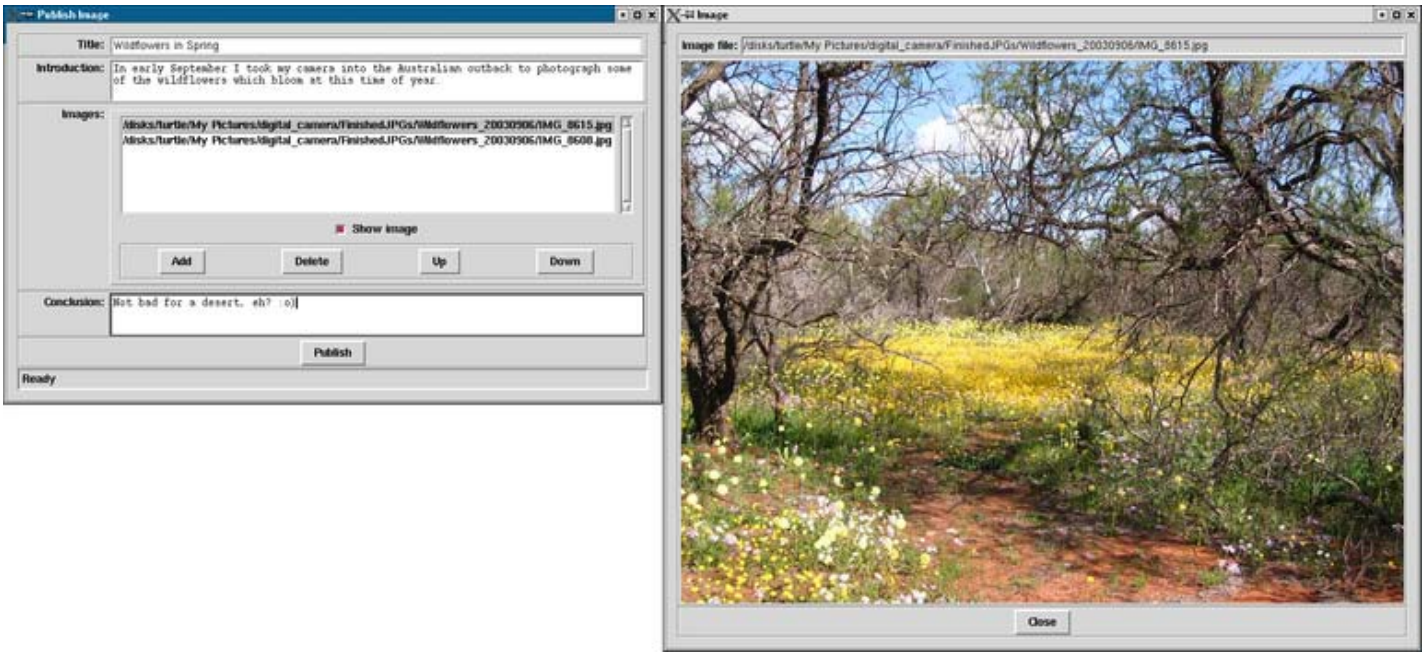

Figure 2. The script is running, with the image display window open.

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 up to date and professionally presented, it provides a standard Tcl package with a lot of useful extensions. If you know your users are using ActiveTcl, you know exactly which extensions they have on their machine and therefore can guarantee your script can run. I encourage anyone who wants to run the project in this article to download and install ActiveTcl-8.4.5.0 or later, as that's what I used for development. ActiveTcl comes with its own installer, and if you install it in, for example, /opt/ActiveTcl-8.4.5.0, it doesn'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 make sure 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 the application requires, and it produces a solid framework on which those features can be built. 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, such as my Web page, 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 that should 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). It occasionally offers a text editing window where the user can write the Tcl code that forms the actual application, rather than limiting itself to dealing with the development of the GUI. On the other hand, it doesn't offer a debugger or 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 that seem to get in my way (Figure 3).

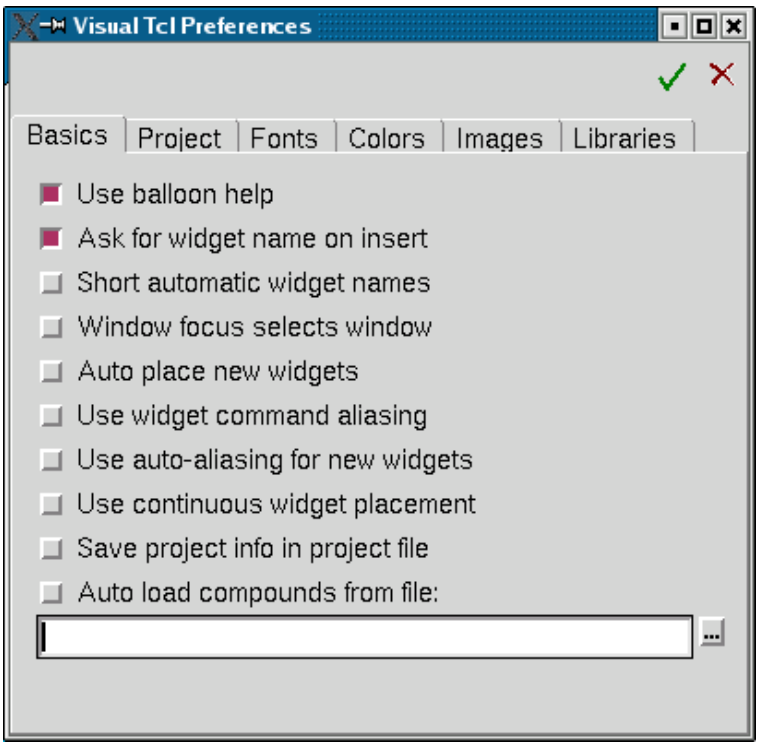

Figure 3. Visual Tcl is highly configurable.

Instead I chose to write the bulk of the application logic in my favoured environment (XEmacs) and simply used the output from Visual Tcl as a library that 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.

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

```
#!/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
#
```

Once I understood the way I wanted to work with the tool, it didn't take too long to produce the output I wanted. Widgets are placed using 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 also are 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 that load the Tcl/Tk code are surprisingly tolerant, which means the generated code can be edited and tuned independently 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 offers only the basic building blocks of widgets. Things like comboboxes and notebooks aren't available in Tk. Fortunately, a number of extensions to Tcl/Tk provide these mega widgets, and Visual Tcl supports them all. The drawback 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 set. 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 the excellent Qt Designer can tell you that Visual Tcl needs more work. It's slow on my dual PIII-500 machine to the point of being irritating, and it 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 Visual Tcl did the job I required. 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 that sets Tcl apart from more modern GUI scripting solutions is the way the Tk toolkit interacts with the Tcl code that does the work. Packages such as 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. Developers really need to understand the widgets with which they're working 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 of a peer-to-peer nature. The GUI toolkit operates at the same level as the language driving it, which makes the combination easy to work with. Take, for example, the listbox widget that 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 always are reflected in the listbox widget. If I add, move or delete an item in that list variable, the contents of the listbox widget are updated immediately and automatically to display its contents. The code that handles the image list doesn't access or interact with the GUI at all. It simply keeps a single list variable in the correct state, safe in the knowledge that Tcl/Tk does 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 sometimes is 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 changes as container widgets, such as frames, are added and removed. To prevent the script writer from having to keep track of the full names of the important widgets, Visual Tcl allows the user to specify an alias—a short, easily memorable name by which the widget always is known. 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, always is easy. For example, I gave the Introduction text widget the alias IntroText. To fetch the text currently in that widget, the code in Listing 2 can be used.

Listing 2. Fetching the Contents of an Aliased Widget

```
...
set introWidget $::widget(IntroText)
set text [$introWidget get 1.0 end]
...
```

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 1 character 0 to the end, also is easy.

The image display in the viewer window actually is 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 (Listing 3). 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.

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

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

When the user clicks the Publish button, a Tcl function is called that creates the Web page. The workings of this code aren't especially relevant here. Suffice it to say that Tcl allows generation of an XML DOM using the TclXML extension and then allows the callout 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

Although the Tcl/Tk script works nicely, it's hard to ignore the obvious gulf in quality between the appearance of a Tcl/Tk script and a more modern Qt or GTK 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 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 have. Python, coupled with GTK or Qt, now provides a more contemporary solution to many of the problems for which Tcl/Tk used to be the natural choice. Both Tcl/Tk and Visual Tcl have some ground to make up in terms of looks, features and desktop integration. Yet, the simplicity of application development offered by the mature and superbly integrated combination of the Tcl language and the Tk toolkit still is second to none. If you have a simple scripting task that would benefit from a GUI, where speed and cost of development are important, Tcl/Tk still should be near the top of the list of contenders for the job.

Resources

ActiveState Tcl Web Site: www.activestate.com/Products/ActiveTcl

Incr Tcl: incrtcl.sourceforge.net/itcl

Practical Programming in Tcl and Tk, 4th edition, by Brent Welch. Prentice-Hall PTR: www.beedub.com/book

The 11 Rules of the Tcl Syntax: www.tcl.tk/man/tcl8.4/TclCmd/Tcl.htm

Source for the Script Developed in This Article: ftp.ssc.com/pub/lj/listings/issue119/7225.tgz

The Tcler's Wiki: mini.net/tcl

Tcl/Tk Headquarters: www.tcl.tk

Tcl/Tk man Pages, On-line and Downloadable: www.tcl.tk/man

Visual Tcl: vtcl.sourceforge.net

XSLT for libxml2: www.xmlsoft.org/XSLT.html

Derek Fountain is a freelance software developer, specializing in UNIX and Linux. He strongly believes in the adage of “make it as simple as possible, but no simpler”. That's why he deploys scripting solutions wherever possible. He lives in Perth, Western Australia.
