Tclhttpd

Adding a web interface to an application is an increasingly common requirement. Remote access to applications has obviously been essential in the server space since the beginning, but it is now becoming popular in the desktop space as well. Many applications now benefit from having some sort of interface which allows them to share data and status information across the corporation, and sometimes across the wider Internet as a whole.

The way most programmers address this issue is to interface their code up to a web server using PHP, CGI or similar technology. While this can always be made to work, it often introduces problems of installation and administration, and requires privileges the regular desktop user doesn't have.

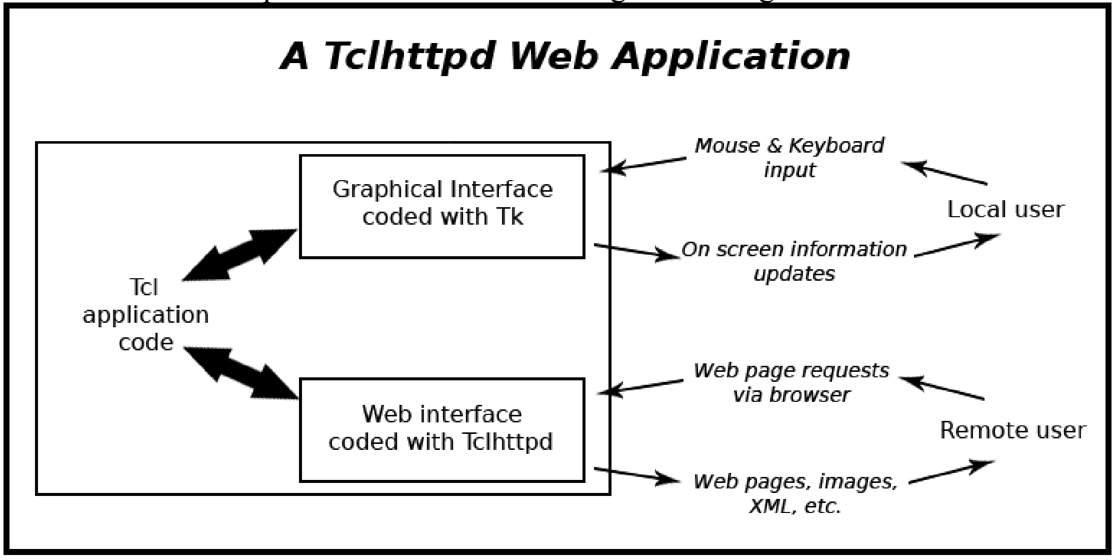
There is an alternative approach, however. It is perfectly possible for an application to open a communications port and for it to accept and handle requests from a web browser itself. The reason this is not a particularly popular approach is because of the complexity involved. If the application is going to field web requests itself, it needs an implementation of the HTTP protocol inside it, plus support for standard web facilities such as authentication, cookies, session management, and so on. Writing a complete web server in such a way that it can be embedded into an application is a difficult task.

There is, however, a readymade, free software solution to this problem. Tclhttpd is a Tcl library written by Brent Welch, which provides a complete HTTP server implemented in script. It is specifically designed to be embedded into an application. This article gives an overview of Tclhttpd, and examines the facilities it provides for connecting application logic up to a web interface.

Tclhttpd as a web application base

Tclhttpd runs out of the box as a well featured, multithreaded web server, and is occasionally found behind web sites which attract tens, sometimes hundreds of thousands of hits per day. The Tcl/Tk website at http://www.tcl.tk is built on Tclhttpd, and the demands on the server are graphically demonstrated on the status page at http://www.tcl.tk/status. But acting as an Apache replacement is not really the idea of the package. Tclhttpd is a piece of code which is designed to be built on and extended into something much more than a web server. In fact, the idea is really for it to be embedded into a application of which the web interface is just a small and incidental part.

The application can be as simple or complex as necessary. It can have a Tk Graphical User Interface if a local desktop user is to be able to use or manage it alongside the remote web based users. As far as the application is concerned, a web server connection should only really be seen as one way to get information from the application. Requests from the server can trigger calls to the application code in the same way requests from the GUI can, with responses being formed as HTML rather than updates to the on screen widgets. See Figure 1.


A Tclhttpd Web Application
Graphical Interface coded with Tk
Tcl application code
Web interface coded with Tclhttpd
Mouse & Keyboard input
On screen information updates
Local user
Web page requests via browser
Web pages, images, XML, etc.
Remote user

A Tclhttpd based application is set up like any other. It will be started from the command line as normal, then it will read its configuration and data files and then build and present its user interface. It will also make a few calls to the Tclhttpd Application Program Interface (API) which configure the embedded web server and set it listening for incoming page requests on the required port. The application will then enter its event loop from where it will dynamically respond to page requests in much the same way as it will dynamically respond to button clicks, timer expirations and all the other events which it needs to deal with as it runs.
The building of the actual application, whatever it might be, is beyond the scope of this article. What follows is a brief look at the mechanisms the Tclhttpd package uses to link page requests from the web server to the application logic which knows how to generate the required responses.

Serving web pages

The simplest and most obvious use of any web server is to serve static web pages. Tclhttpd can do this, of course, and with its multi-threaded design is plenty fast enough to meet the requirements of a busy website on fairly modest hardware. Legacy features like CGI are also supported if you need to make use of existing code.
But it's dynamic page generation where Tclhttpd gets interesting. When a request comes in for information which needs to be supplied by the host application the server is embedded in, the web server code needs to be connected directly to that application logic. For a developer to get some idea of how he might attempt this task, he first has to consider the three mechanisms Tclhttpd makes available for attaching the server request to the content generating application code.

Application Direct URL

The simplest method of connecting a server request to the application is known as an Application Direct URL. The principle here is that at application start up the programmer adds an entry to a table which links a URL to a Tcl procedure. When an HTTP request is received for that URL, the Tcl procedure associated with it is called automatically by the server code. The procedure returns a string (normally some HTML code), which is passed directly back to the requesting browser.
For example, suppose we write a Tcl procedure called handleLogin, which will handle user logins. We can configure the embedded server to pass a request for the login page to this procedure with a call like:

```
Direct_Url "/login" handleLogin
```

Direct_Url is a procedure in the Tclhttpd API which is used to make additions to the web server's Application Direct URL table. Making this call ensures that when a web browser asks the server for http://localhost/login the handleLogin procedure will be called automatically. Any query parameters in the request are passed into the procedure which then returns its result as an HTML string.
Direct URLs are actually configured with URL and procedure name prefixes. This means that, for the example above, a request for the URL http://localhost/login/guest will result in a call to the Tcl procedure "login/guest". If this looks like a strange procedure name, remember that slashes happen to be valid characters in Tcl procedure names. This is convenient because it means whole groups of Tcl procedures can be set up to deal with whole groups of URLs with just one line of Tcl code to configure the server.

Document Types

An alternative way of configuring the server is to set it up so that requests for any documents of a specific type are handled by a single Tcl procedure. This is done by registering a document type, then linking that document type to a specified Tcl procedure.

At start-up, the server reads its mime.types file which maps the usual file suffixes like .html and .jpg to their respective mime types: text/html and image/jpeg for these examples. Additional entries can be added to this file to create new mime types if necessary.
The Document Type association between a specific mime type and the Tcl procedure which handles it is made by giving the Tcl procedure the right name. The right name is simply the string "Doc_" prepended to the mime type. For example, to install a procedure which will handle all requests for any files of mime type image/jpeg, the developer simply creates a procedure called "Doc_image/jpeg". The server automatically detects procedures which are correctly named to handle mime types, so nothing more is required.
As with Application Direct URLs, Document Type handling procedures receive details of the actual request in their arguments. However, because the data returned often isn't simple HTML, they can't just return a text string. The HTTPD library in the Tclhttpd API contains a selection of routines to assist with returning binary data or the contents of files to the browser.

Domain Handlers

A good part of a web based application interface can be implemented using the simple Application Direct URL and Document Type handling facilities described above. Sometimes, however, things are a little more complex than those features can handle, so the developer will need to call upon the most powerful method of handling server requests. This is known as a Domain Handler, and it presents the most flexible interface between the server and the Tcl code which is going to handle the request.
The idea here is that any area of the website can be configured to be handled by one piece of Tcl code. For instance, a developer might set up the server such that any URL which looks like http://localhost/products/* is handled by a single procedure called productEnquiry:

```
Url_PrefixInstall "/products" productEnquiry
```

As usual, the productEnquiry procedure is passed a set of parameters which allows it to examine the URL requested and thus construct its response.
Domain handlers need to be used carefully since their design allows one Tcl procedure to be in charge of one complete area of the website. Should that area of the website start to grow, the domain handling procedure could rapidly grow out of control.

Using embedded Tcl script

Tclhttpd also supports a mechanism which allows script to be embedded into the HTML of a web page, which the server runs before the page is served. This facility is referred to as HTML+Tcl Templates, and is the equivalent of embedded PHP or any other server side scripting language.
The principle is identical to the use of PHP: a page of otherwise normal HTML contains pieces of code which the server locates and runs. The results of those code pieces are then entered directly into the HTML, in place of the code itself. The only significant difference between HTML+Tcl Templates and PHP is the way the feature is normally used. The flexibility provided by the URL handlers described above means complex code is generally kept out of the web pages themselves. Embedded code is mostly used for HTML templates which give a site a consistent look and feel.
The use of HTML+Tcl Templates is based around a "template file", which has a standard suffix of .tml. When a request for an .html file is received, the server looks to see if there is an equivalent .tml file. If there is, the timestamps of the two files are checked. If the .html file is the newer one, its contents are served as normal. If the .tml file is newer, the Tcl scripts inside it are executed and the result is saved in the .html file. Put simply, the .html file is used as a cache for the result of the code in the .tml file. In order to update the look and feel of an entire website, only the .tml files need to be changed. The new .html files will be regenerated by the server as they are requested, with no server restart necessary.

Conclusion

Tclhttpd provides a full featured, highly scalable web server which is designed to be embedded inside an application in order to provide that application with a web interface. It deals with all the complexities of the HTTP protocol, leaving the developer free to concentrate on the application itself. It provides several methods for connecting web page requests to the application code which can fulfil them. It also has a rich API which simplifies all aspects of coding for the web. It is implemented in pure Tcl script, which makes it portable across platforms.

Anyone with a problem which would normally call for a combination of Apache and a scripting language should at least pause for a moment and consider whether a single application with an embedded web server might be a quicker and simpler solution.

Sidebar: minihttpd

The Tclhttpd project started out as an HTTP server implemented in about 175 lines of Tcl script. Over a number of years it has grown into a large piece of code with a lot of features, and now requires considerable effort to understand, administer and write code around. However, at the same time as Tclhttpd was starting to expand, another programmer, Steve Uhler, decided there was still plenty of merit in keeping a much smaller version without the bells, whistles and complexity, so the code forked and minihttpd was born.

The original goal of minihttpd was to provide a basic web serving facility in 250 lines of script and the present version still meets that requirement. Without comments, the current version of minihttpd is 246 lines, and is still supplied with the main Tclhttpd package.

minihttpd retains the feel of the first version of Tclhttpd. It implements the very basics of the HTTP/1.0 protocol, and offers very little in the way of extras. You get basic error handling and logging, and that's about it. There are no dynamic facilities of any sort, no CGI and nothing clever like threading or access control.

What you do get, however, is remarkable simplicity. The code comes in one file containing about a dozen procedures. If all your application requires is the delivery of the occasional static web page, minihttpd is all you need. Because the code is so simple, it is also trivial to work on. This means simple, dynamic applications can be built on top of minihttpd very quickly.

minihttpd exists to fill a very real niche. The fact it is small and almost completely devoid of features makes it just as useful in its own way as the full blown Tclhttpd package itself.