The following paper was originally published in the
Proceedings of the Sixth Annual Tcl/Tk Workshop
San Diego, California, September 14–18, 1998

NeoWebScript: Enabling Webpages with Active Content Using Tcl

Karl Lehenbauer
NeoSoft, Inc.

For more information about USENIX Association contact:

1. Phone: 510 528-8649
2. FAX: 510 548-5738
3. Email: office@usenix.org
4. WWW URL: http://www.usenix.org/

NeoWebScript: Enabling Webpages with Active Content Using Tcl

Karl Lehenbauer
NeoSoft, Inc.
karl@NeoSoft.com

Abstract

NeoWebScript marries the world's most popular webserver, Apache, with the Tcl programming language to create a secure, efficient, server-side scripting language that gives webpage developers simple-yet-powerful tools for creating and serving webpages with active content.

The ability to embed NeoWebScript code into existing webpages, without requiring a URL name change, leverages work done with webpage creation tools such as Netscape Communicator and Net Objects Fusion, while not disturbing links from remote sites and search engines.

A mature application, NeoWebScript-equipped webservers are currently in production on the Internet, serving real-world loads of millions of webpage "hits" per day.

This paper describes the driving forces behind the creation of NeoWebScript, and how those forces shaped its design and evolution into its current-day form. NeoWebScript's software architecture is described, and its capabilities are demonstrated using numerous examples, including webpage "visitor counters", rotating banner ads, queuing email, posting news, storing form submissions into Berkeley-style "dbopen" databases, and creating graphical images on the fly.

Finally, NeoWebScript's current status is summarized, our near-term plans are detailed, and conclusions are drawn.

1. Why Create NeoWebScript?

Our company, NeoSoft, Inc., is a large regional Internet Service Provider (ISP). NeoSoft has been running a webserver since 1992, and has been providing web-related services for customers for several years. The mix of customers using our web services spans the widest level of abilities, covering a broad range of applications. Customers use every platform and development tool, and every web publishing technique has limitations that affect how the developer interfaces with specific server-side capabilities. While much attention has been given to Java and JavaScript for client-side scripting, many forms of active content require that data be maintained, accessed and updated on the server. Such applications include electronic commerce, shared databases… even simple things like hit counters.

As our customers became more sophisticated and creative, they began asking for *active content* features such as access counters and rotating banner ads. Many had CGI scripts that they had written, or obtained, and wanted to run them on our webserver. Others wanted us to write them. We found that CGI programs had a number of problems:

- The overhead of the CGI approach is fairly high. For each CGI executed, the webserver must set up, start, manage, and communicate with a child process that executes the CGI.

- Redirecting existing URLs to scripts required reconfiguring the webserver. Redirected URL names tended to be unwieldy, and the rewritten web page addresses were confusing to users.

- Letting untrusted users run CGIs with the same user ID as the webserver creates a security problem; likewise there are security problems with running the webserver as superuser so that it can obtain permissions of the user when running a script as the user. A user could, in either case, accidentally or intentionally compromise their own files; in the latter case, an intruder may be able to directly gain superuser privileges.

- CGI programs typically emit the entire webpage programmatically, either rendering HTML authoring tools unusable or requiring "hand-stitching" to weave the tool-created HTML into the program, a fairly expert task that must be per-

formed every time the tool-authored HTML is altered.

We began by modifying the webserver in C to add support for hit counters and banner ads. This approach was successful in that it provided some capabilities customers were asking for, and had lower overhead and was clearly more secure than the CGI approach, but it was immediately clear that we needed a more general solution than modifying the webserver on an ad-hoc basis. We wanted to create and make available to our users, and others, a simple and easy-to-learn tool for scripting active content.

To address these needs, we decided to integrate our own solution by mating an existing webserver with an embeddable scripting language.

We chose the *Apache webserver*[1] because we were already using it successfully in production to provide virtual web services for hundreds of domains. We had some skill with it, and we knew it to be capable, under active development, and in widespread use -- indeed, Apache is the world's most popular webserver[2] and comes with full source code. Since Apache is freely redistributable, including for commercial resale, it kept our options open with regard to creating a commercial version if we chose to. Derived from the widely used *NCSA http* server[3], among Apache's enhancements were a modular architecture, designed and documented for the purpose of accepting third-party "plug ins" such as the one we hoped to create.

For the scripting language we chose the Tool command language (Tcl)[4]. An explicit design goal of Tcl was that it be easy to integrate into other applications. Tcl is known for being easy to learn, and has strong text processing capabilities. We already had substantial experience with Tcl, which lowered the technical risk. Tcl's developer community had produced many excellent tools and packages, usually distributing them under the permissive Berkeley copyright, which we could utilize to add capabilities to the server that were far beyond our own resources to create and maintain. Finally, "Safe Tcl", by then supported as part of the Tcl core, looked very promising for providing users with a way to program the webserver with far less risk than the traditional server-side programming technologies.

2. Requirements

Fundamentally, NeoWebScript had to be reliable. It had to be able to serve millions of hits per day without failure. Accidental overwrites of adjacent data, memory leaks, etc, by the Tcl interpreter would have more serious consequences than for a CGI program, as the interpreter would run within the Apache server's child's address space, and a single Apache child process handles many webpage requests before terminating. Such problems wouldn't always manifest themselves until a subsequent page was served, which could significantly complicate debugging. Likewise, bugs in the Apache server could corrupt Tcl. Fortunately, both Apache and Tcl were (and are) robust and reliable – We have not had any significant problems with Tcl-enabled Apache processes malfunctioning or dumping core.

Another requirement was that, as our code progressed from an experiment to a production web scripting system, we embrace Apache's configuration files and configuration technology, which we did. All NeoWebScript-specific capabilities are configured and controlled through compliant configuration file extensions, using Apache's configurable module architecture.

One requirement greatly influenced the evolution of NeoWebScript. As an ISP, we have thousands of users who are not employees, all of whom are potential NeoWebScript developers, where *their* scripts are running on *our* servers. Compare this to a more traditional organization where, for example, a handful of employee-developers produce the organization's Internet and/or Intranet content. This is a critical distinction. As a result, NeoWebScript was designed from its inception to maintain the webserver's security while operating with an *untrusted user base*. This led to a design choice between providing webpage developers with the full power of a normal Tcl interpreter and our need to protect the server from those same developers, most of whom we have never met face-to-face.

Therefore, at every decision point, we opted for security over unencumbered power. By default, NeoWebScript's user files are kept out-of-band from the users' webpage files, protecting those files from being overwritten by a script, at the cost of not being able to script to create or manage files in the user's home directory. External programs cannot be executed except for certain specific applications (posting news and sending electronic mail, for instance) which are handled through tightly controlled interfaces that must be custom-developed for each supported program.

3. Design Philosophy

Our philosophy was that we would strive to make it easy to do "90%" of the things people wanted to do. This approach had served Mark Diekhans and I well in the development of Extended Tcl (TclX)[5], where, for example, we invented a simple way to create both client and server TCP sockets (which provided the model for what later became the Tcl core's *socket* command). This let users connect with or write Tcl code to talk to mail, news, the web, etc, but left the creation and interpretation of the less commonly used datagram (UDP) and more exotic multicast packets to other extensions such as Tcl-DP[6].

While our plan was to build in lots of capability, we wanted something that a fledgling webpage developer, with decent HTML skills but little or no prior programming experience, would be able to use to create simple active content features. A number of demos would be included, enabling web developers to cut and paste a number of interesting active-content elements into their webpages, all the while leaving their choice of HTML development tools completely open. An example of NeoWebScript usage appears in Figure 1.

```
<html>
<body bgcolorwhite>
<h1>Welcome to my webpage</h1>
…
You are visitor number
<nws> incr_page_counter </nws>
…
</html>
```

Figure 1 - NeoWebScript HTML code fragment to produce a webpage "hit" counter that automatically increments every time the page is retrieved.

A major goal was to make it easy to receive, store, locate and recall data entered via forms. An example page that obtains and stores data sent in through a form is shown in Figure 2.

```
<title>Simple Form Result</title>
<h1>Simple Form Result</h1>
<nws>
load_response response
dbstore simple $response(name) response
</nws>
Response stored.  Thanks!
```

Figure 2 - The minimal NeoWebScript page to store the results of a form submission. By setting a form's action to point to a page containing this code, the key-value pairs comprising the form are stored in a Berkeley-style "dbopen" file[7], using the field "name" as the key.

Finally, for experienced developers, we planned to provide interfaces to database back-ends, a way to make and/or modify graphical images from data, provide access to the environment variables normally available to CGIs, and do whatever else users could do with the general-purpose programmability of a Safe Tcl interpreter. This would allow developers to customize content based on browser type and version, date, address, host name of the client, etc.

3.1 Commerce Servers

We also wanted NeoWebScript to have a secure socket layer (SSL) capability. This would allow NeoWebScript applications to support the encrypted sessions required of commerce applications, etc. The commercial *Stronghold* commerce server (http://www.c2.net) adds a secure socket layer (SSL) encryption capability to Apache, and would be used to create a compatible, secure version of NeoWebScript.

Another commerce server option is the freely redistributable *Apache-SSL*.[8] Although at one point no Netscape or Explorer-recognized certificating authorities would sign digital certificates for Apache-SSL, Thawte (http://www.thawte.com/) has been doing so for some time. With Verisign's (http://www.verisign.com/) recent decision to sign certificates for Apache-SSL, Apache-SSL represents another viable commerce server platform that is NeoWebScript-compatible.

4. How It Works

When a webpage containing NeoWebScript code is requested, an interpreter is created and loaded up with services and an array of information about the connection.

The embedded code is evaluated within the interpreter. A number of services are provided:

- A Safe Tcl interpreter
- Form elements included with pages sent as GET and POST requests can be imported into an array of key-value pairs. (An example is shown in Figure 2.)

- Arrays of key-value pairs can be written to and read from btree-indexed disk files with a single statement.

- Arrays of key-value pairs can be translated to SQL statements and written to Oracle[9] and Postgres[10] databases, on the local machine or across the network. SQL queries and cursor walks produce data as arrays of key-value pairs.

- Creation and/or modification of GIF-format graphic images from a Tcl script, using Thomas Boutelle's *graphics draw* (gd) package.[11] Examples of on-the-fly graphic image generation can be seen in Figure 9 and Figure 11.

- Access to the environment variables normally available to CGI programs through the *webenv* array.

4.1 Apache Module Architecture

The Apache webserver is written in C. It has a modular architecture that splits the process of serving a webpage into eight stages. The stages are shown in **Table 1**.

URL-to-filename translation
Authentication
Authorization
Permissions
MIME Type Determination
Last-Stage Fixups
Emitting the Page
Logging the Results

Table 1 - Apache webpage pipeline modular stages

Apache provides a number of services to modules, controlled by a "switch" structure. Modules have the option of having initialization code executed at webserver startup time, of receiving configuration information when retrieving a page from config files in that page's directory, or merged among zero or more higher-level directories. Modules can opt to receive configuration information from the server config files, and can define handlers for one or more MIME data types.

Apache modules can install their code into a "chain" of handlers that can accept, pass on, or reject the aforementioned filename translations, user authentication and authorization checks, etc.

4.2 Processing Webpages

The NeoWebScript module is activated only when a request for a file matches the administrator-configured NeoWebScript file extension, which can be **.html** to enable any existing HTML page to include NeoWebScript code without a URL change, or a special extension such as **.nhtml**. If the file being requested does not match the special NeoWebScript MIME type, the file is processed by Apache's standard handlers, without any intervention by our code. If the page matches NeoWebScript's MIME type, it is handled in the same manner as Apache's server-side include handler (The NeoWebScript module is, in fact, based on Apache's *mod_include* module.)

Upon the first occurrence of a NeoWebScript tag, a safe interpreter is created and populated with variables, procedures and exported commands for use by the Tcl code contained in the webpage.

The embedded Tcl code is then evaluated, and its output is merged with any static content (standard HTML) that may be present in the page.

The same interpreter that executes the first chunk of NeoWebScript code executes any subsequent NeoWebScript code contained within that page.

After the page has been completely emitted, the interpreter is destroyed, as it cannot safely be reused. Note that the cost of creating, configuring and destroying a Tcl interpreter is far less than starting up and running a separate CGI program.

If no NeoWebScript tag is found in a page, the page is emitted without creating an interpreter or executing any Tcl code. This makes a NeoWebScript-enhanced Apache server perform at just a tick under the performance of an unenhanced server. When the NeoWebScript module is added to Apache, the server's memory footprint grows by the size of the NeoWeb-

Script module, plus a Tcl interpreter and associated code. The processing overhead increases slightly due to the small overhead of looking for NeoWebScript tags.

As NeoWebScript's capabilities have evolved, our use of Apache's services has increased. Currently we support per-directory and merged-directory configuration, general configuration through the server config files, and, of course, handlers to parse and send pages by executing the NeoWebScript code embedded in the page and merging it with any static text present in the HTML file.

4.3 Supervisor Mode Pages

After we had NeoWebScript up and running for a while, we saw two things. One was that sites with a trusted user base – a relatively small number of developers who were trusted by the server administrators – were unnecessarily inconvenienced by not having all of the capabilities of a full Tcl interpreter. Another was that users were running into the need to develop their own *control pages*, i.e. pages that made it possible for them to see things like what db files they had, how big they were, etc.

We wanted to provide a way to allow trusted developers to have the full capability of the Tcl shell, even on a server where most developers were untrusted and hence would only have the safe functionality. Also, we wanted to be able to create special pages that could be accessed by many different users, where each user would use the same page to look at their files, but not at one another's.

Supervisor Mode gives pages in specified directories full Tcl capabilities (not just safe ones) and allows those pages to assume the identity of other users (with respect to the server-maintained user identities – to the operating system, all the NeoWebScript files are owned by the webserver). This allows construction of site-wide control pages, multi-user data upload and download pages, etc.

We added the ability for supervisor mode pages to do password authentications against the UNIX password file, allowing users to login to services provided by supervisor mode pages and using those pages to access and manage their NeoWebScript files.

4.4 Tcl-oriented logging module

We created a new logging module, *mod_log_neo*, to augment or replace the default logging module, *mod_log_config*. Our logging module combines access logs and browser-type logs. It avoids the relatively costly per-hit time and date conversions by logging times in UNIX integer-since-1970 format. It also writes logfile entries as Tcl lists, making them easier to parse by Tcl-based reporting tools. (This also makes it harder to spoof or trick the logfile processor with bogus URL requests.) An example of a message logged by *mod_log_neo* appears in Figure 3.

```
901018069 203.162.3.234 {} {} 304  0
vnmusicawards.com
{GET /summit2.jpg HTTP/1.0}
{Mozilla/4.05 [en] (Win95; I)}
```

Figure 3 – An example log entry produced by mod_log_neo (broken into multiple lines for readability). Entries are logged as Tcl lists consisting of the time, the IP address of the requesting host, the hostname of the requesting host (empty if IP-to-hostname DNS lookups are disabled), and the user ID (empty if none was specified). Next comes the HTTP status code and number of bytes returned, followed by the virtual host and HTTP request serviced, followed by the browser identification string.

The *NeoWebStats* application creates summary data from the logfiles, which average more than 100 megabytes per day for a site serving a million hits per day. Summaries can be combined to produce summaries spanning multiple days. NeoWebScript's on-the-fly graphics generation capability is used to produce pie charts showing the proportion of hits at various "depths" within the webserver. An example of NeoWebStats output is shown in Figure 11.

5. Examples

5.1 Retrieving Data from a Client Request

There are two common ways for web clients to send data to a webserver. One way is to use the HTTP[12] GET method, in which key-value pairs are coded into the URL request. The other is to use the POST method, in which the key-value pairs are sent following the HTTP request line and the key-value pairs (browser type, cookies, etc.) that are always sent, regardless of the type of request.

In either case, NeoWebScript parses the data in the request and stores it into a global Tcl array using the NeoWebScript command **load_response.** An example using **load_response** is shown in Figure 2.

5.2 Sending Email from a Webpage

An example of the use of **open_outbound_mail** to send an Internet email message from a webpage appears in Figure 4.

```
set fp [open_outbound_mail \
        "Moving to Montana?" $toWhom]
puts $fp "Do you think I could interest you in"
puts $fp "a pair of zircon-encrusted tweezers?"
close $fp
```

Figure 4 - Sending email from within NeoWebScript

Open_outbound_mail returns a Tcl filehandle. The embedded code uses **puts** and any other relevant file-oriented Tcl commands to create the message body. The email sender's address is automatically constructed from the username of the owner of the webpage that's being interpreted and the name of the server that did the serving. If *to* is not specified, the recipient is also set to be the user name of the owner of the webpage.

5.3 Posting News

Open_post_news starts a Usenet news posting, returning a Tcl filehandle to be used with **puts**, etc, to specify the contents of the body of the news posting. The message always comes "from" the user name of the owner of the webpage being interpreted, coupled with the name of the server doing the serving. Example code for posting news is shown in Figure 5.

After writing out the body of the news article, a line is written to the filehandle consisting of a single period, then the file is closed. (This is a requirement of NNTP. It could, of course, be hidden by a proc.)

```
set fp [open_post_news -subject $subject \
    -newsgroups neosoft.announce \
    -distribution neosoft]
puts $fp "This is the body of the message."
puts $fp "."
close $fp
```

Figure 5 – Posting news from NeoWebScript

Note that for this to work you must have a news server supporting Network News Transfer Protocol (NNTP)[13] in the same domain as your webserver. For example, within *neosoft.com*, **open_post_news** will contact the news server *news.neosoft.com*.

5.4 Graphical Access Counters

A "hit counter" for the current page is fetched, incremented, and returned by calling **incr_page_counter**. **Incr_page_counter** automatically manages the counter using a db-file, where the index is created based on the current URL. An example showing the number of times a page has been visited, where GIF files represent each digit of the count is shown in Figure 6.

Figure 6 - Number of page visits shown by on-the-fly constructing HTML image references to images that exist as numbered GIF files, one per digit.

The code that created this display is shown in Figure 7. **Split** is used with an empty string for the characters to split on, causing the number of visits to be exploded into a list, where each digit in that number is one element of the list. We walk the list using **foreach**, emitting image references to a GIF image file for each digit to be displayed.

```
<center>
<h1>Welcome!</h1>
This page bas been visited
<nws>
foreach num [split [incr_page_counter] {}] {
html "<img src/images/counters/set22/$num.gif>"
}
</nws>
 times since July 1, 1998
</center>
```

Figure 7 - NeoWebScript code fragment for the graphical access counter example.

5.5 Rotating banner ads

Using Tcl, rotating banner ads can be easily implemented by calling **random_pick_html**. Figure 8 is a real example from the NeoSoft, Inc. homepage, located at http://www.neosoft.com/. The example has been simplified by having the heights and widths of the images, and alternate image strings removed to improve readability.

```
random_pick_html {
{<a href/neosoft/>
        <img srcimages/sroom2.gif></a>}
{<a href/neosoft/neorules.html>
        <img srcimages/resources.gif></a>}
{<a href/neopolis/>
        <img srcimages/neopolis.gif></a>}
{<a href/neowebscript/>
        <img srcimages/logos/nws7.gif></a>}
}
```

Figure 8 - NeoWebScript code demonstrates the random selection of one of a number of specified HTML components every time the page is retrieved.

5.6 Issuing and retrieving cookies

```
neo_make_cookie –user ellyn –days 30 \
        –path /myApp
```

This example creates a cookie named *user* containing the text "*ellyn*". This cookie will be sent by the user's browser as part of all HTTP requests sent to this same server, for a period of 30 days, whenever the URLs requested are underneath **`/myApp`** on this server, and the browser is cookie-enabled.

```
load_cookies cookies
html $cookies(email)
```

`load_cookies` retrieves all of the cookies, if any, uploaded by the browser in the HTTP request header, breaking out each cookie into a separate array element of the array name specified in the call.

NeoWebScript can generate and/or modify graphic images at runtime. In Figure 9 we see an "analog" clock, generated live, showing the current time of day.

Figure 9 - GIF file of an analog clock, produced on the fly using NeoWebScript

We create live GIF files by creating Tcl scripts that are end in a **`.gd`** extension. When the file is referenced from an **`<img src…>`** tag in a webpage, our code is executed and is expected to produce a GIF file when complete by executing *gd writeGIF*. (Note that the image filehandle is passed in as the hard-coded global variable *imageFile*.)

5.7 Creating Graphic Images from NeoWebScript

```
proc draw_analog_time {im color} {
    scan [clock format [clock seconds] -format "%I %M"] "%d %d" hour minute
    set hourStart [expr 90 - ($hour * 30 + $minute / 2)]
    set minuteStart [expr 90 - ($minute * 6)]

    set minuteX [expr int(40 + cos($minuteStart * 3.14159 / 180) * 30)]
    set minuteY [expr int(40 - sin($minuteStart * 3.14159 / 180) * 30)]
    gd line $im $color 40 40 $minuteX $minuteY; # draw three times
    gd line $im $color 40 43 $minuteX $minuteY; # to make it pointy
    gd line $im $color 43 40 $minuteX $minuteY

    set hourX [expr int(40 + cos($hourStart * 3.14159 / 180) * 20)]
    set hourY [expr int(40 - sin($hourStart * 3.14159 / 180) * 20)]
    gd line $im $color 40 40 $hourX $hourY
    gd line $im $color 40 43 $hourX $hourY
    gd line $im $color 43 40 $hourX $hourY

    gd text $im $color large 25 80 "[format %d:%02d $hour $minute]"
}

set im_out [gd create 82 110]; # create the image
set white [gd color new $im_out 255 255 255]; # background
set black [gd color new $im_out 0 0 0]

gd arc $im_out $black 40 40 70 70 0 360; # draw clock face
gd text $im_out $black small 8 95 "Server Time"
draw_analog_time $im_out $black
gd writeGIF $im_out $imageFile
```

Figure 10 - NeoWebScript code for making the analog clock GIF file

The code that produced the analog clock is shown in Figure 10. In this code, the dimensions of the GIF file are specified using *gd create*. Colors are allocated, then we draw a circle to represent the clock face. We use *gd text* to write the text "Server Time" to the image, then call the proc **draw_analog_time** to create the clock hands and write the numeric time into the image.

Draw_analog_time fetches the current time in hours and minutes into variables called *hour* and *minute*. A bit of trigonometry serves to calculate the endpoints of the clock hands. By drawing the hands three times,

each with slightly different locations for the "center" of the clock face, the clock come out "thicker" in the center of the clock face, drawing to a point at the ends.

A more sophisticated instance of graphics generation from NeoWebScript can be found in *NeoWebStats* (see Figure 11), an on-demand graphic image generator that creates pie charts to show what areas of a website are getting what proportion of hits.

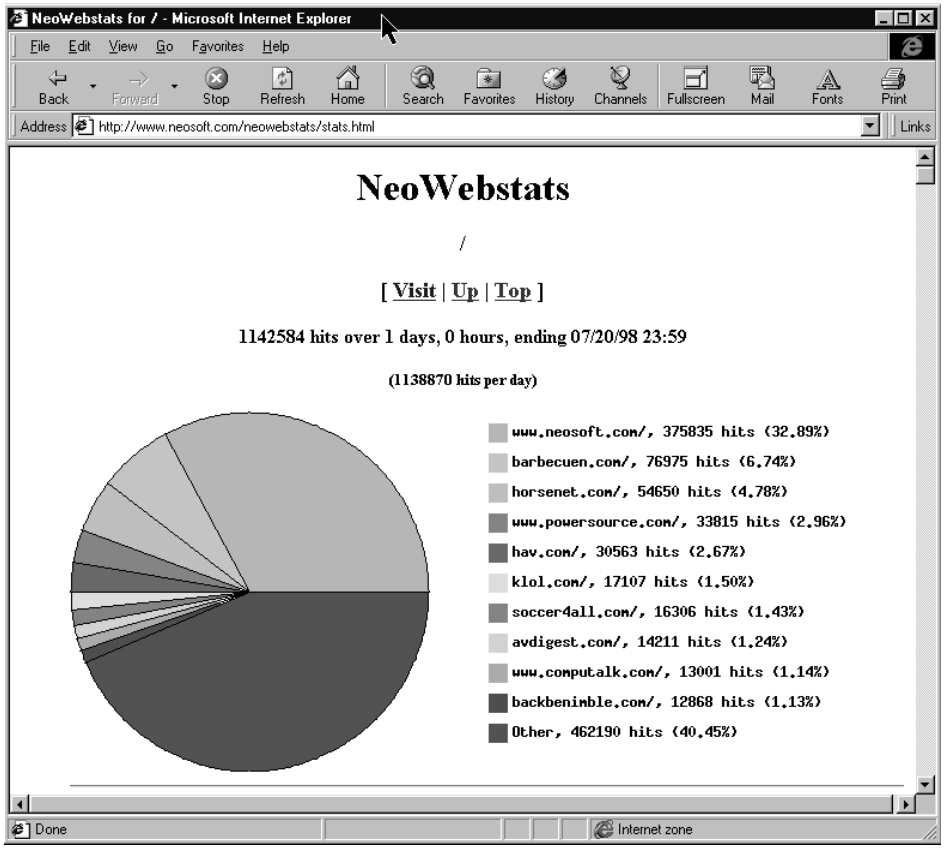

Figure 11 - On-the-fly graphical image generation is used to show the relative proportions of hits among a number of virtual sites. Visitors have the ability to "drill down" into a site and see the proportions of hits among subdirectories within each site, up to a settable maximum depth.

5.8 Accessing Web Environment Variables

NeoWebScript makes approximately thirty Apache webserver environment variables available to the Tcl interpreter when it's interpreting a NeoWebScript page. Among these are the referrer URL (HTTP_REFERER), the browser type and version (HTTP_USER_AGENT), plus a small number new ones, including the last modify date of the document (NEO_LAST_MODIFIED), and the user ID of the owner of the document being served (NEO_DOCUMENT_UID). These environment variables can be accessed by NeoWebScript code through the global *webenv* array. An example webpage that displays some data contained in, and derived from, the web environment array is shown in Figure 12.

The HTML code that created the display is shown in Figure 13. **Remote_hostname** returns the name of the host requesting the page, if it can be determined. (Otherwise the IP address is returned.) **Estimate_hits_per_hour** returns an estimate of the number of hits being served per hour. It does this by examining the server's **access_log** file. It works by seeking approximately 1,000 hits backwards into the access log, comparing the time that entry was made to the current time, and extrapolating the estimated number of hits per hour that the webserver is serving. We then format the current time and the date when the webpage was last modified using the standard Tcl **clock** command. Finally we emit an HTML[14] **mailto:** link to the web administrator, as read from the **SERVER_ADMIN** variable.

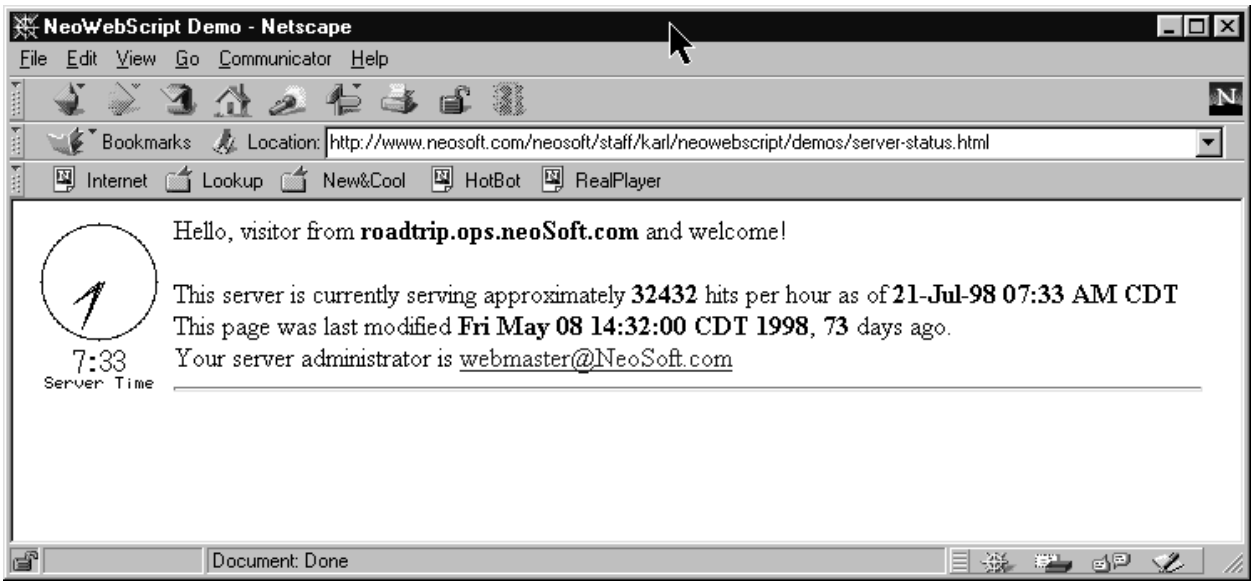

Figure 12 – Webpage showing data derived from webserver environment variables.

```
<img srcgdclock.gd alignleft>
Hello, visitor from <b>
<nws>html [remote_hostname]</nws></b>
and welcome!

<p>This server is currently serving
approximately <b>
<nws>html [estimate_hits_per_hour]</nws>
</b>hits per hour as of <b>
<nws>html [clock format [clock seconds] \
        -format "%d-%b-%y %I:%M %p %Z"]</nws>
</b><br>This page was last modified <b>
<nws>html \
"[clock format $webenv(NEO_LAST_MODIFIED)]"
</nws></b>, <b>
<nws>html "[expr ([clock seconds] - \
  $webenv(NEO_LAST_MODIFIED)) / 86400]"</nws>
</b> days ago.<br>Your server administrator is
<nws>
html "<a hrefmailto:$webenv(SERVER_ADMIN)> \
    $webenv(SERVER_ADMIN)</a>"
</nws>
```

Figure 13 – HTML with embedded NeoWebScript that created the webpage shown in Figure 12

5.9 Subst-style NeoWebScript Pages

An alternative to invoking NeoWebScript code within **<nws>** and **</nws>** tags is server-subst-style NeoWebScript. With subst-style code, the entire HTML file being served is run through Tcl's **subst** command, causing dollar sign variable substitution and square-bracketed code to be evaluated, with the results substituted in place.

The webserver's ability to evaluate subst-style webpages is configured as a handler via Apache's **`srm.conf`** file. The **AddHandler** directive allows you to map certain file extensions to "handlers", which causes Apache to take actions based on file extension. For example, to cause files ending in **.shtml** to be Tcl-substituted and emitted as a **text/html** MIME type, the following lines must be enabled in the **srm.conf** file:

```
AddType text/html .shtml
AddHandler server-subst .shtml
```

A subst-style implementation of the code that produced the webpage in Figure 12 is shown in Figure 14.

```
<img srcgdclock.gd alignleft>
Hello, visitor from <b>[remote_hostname]</b> and welcome!

<p>This server is currently serving approximately <b>[estimate_hits_per_hour]</b> hits per hour as of <b>
[clock format [clock seconds] -format "%d-%b-%y %I:%M %p %Z"]

</b><br>This page was last modified <b>[clock format $webenv(NEO_LAST_MODIFIED)]</b>, <b>
[expr ([clock seconds] - $webenv(NEO_LAST_MODIFIED)) / 86400]</b> days ago.
<br>
Your server administrator is  <a hrefmailto:$webenv(SERVER_ADMIN)>$webenv(SERVER_ADMIN)</a>
```

Figure 14 - A NeoWebScript subst-style webpage that produces the same results as the HTML shown in Figure 13. Note the lack of <nws> tags – the entire page is interpreted using Tcl's **subst** command, causing in-place variable substitution and evaluation and substitution of square-bracketed code.

6. Current Status

There are currently about 1,200 sites running NeoWebScript, according to the Netcraft survey. [15]

6.1 Availability

The current version of NeoWebScript is available for download from http://www.neosoft.com/neowebscript.

6.2 Version 3.0

A new version of NeoWebScript, version 3.0, is currently in beta. This release is the first one to integrate Tcl 8.0 (Version 2.3 uses Tcl 7.6), yielding significant performance improvements in most cases. Also it builds against Apache 1.3.1, and uses *dbopen 2.4.14*. Version 2 of dbopen has new locking, logging and transaction-oriented capabilities, and the release includes new code to support those capabilities through their native version 2 interfaces.

The 1.3.1 version of Apache is much easier to build, using a standard GNU (ftp://prep.ai.mit.edu/pub/gnu) *configure* script to automatically configure the package for the capabilities of the particular UNIX system for which it's being compiled. Apache 1.3.1 also includes a compiler "driver" that greatly simplifies building third-party modules. Also, it allows those modules to be kept separately from the Apache core, and NeoWebScript is currently being built and tested using this technology.

Apache 1.3 includes support for Windows 95 and NT for the first time, and work is underway to make NeoWebScript run in those environments as well.

6.3 Making It Easier to Build

Although we have steadily decreased the amount of effort required to build NeoWebScript from source code, we continue to add capabilities and interface with additional packages. A successful build of NeoWebScript requires successful builds of Apache, Tcl, TclX, and NeoTcl (A NeoSoft-maintained Tcl extension set.) Now that we have added interfaces for Postgres, Oratcl, Scotty, and other packages, we felt that, overall, we were losing ground in this area – NeoWebScript can be fiendishly difficult to build, configure, and install. A binary release for Windows, and native install packages for popular Unix architectures, will make NeoWebScript much easier to get running, continuing our work to improve the build process. These improvements will bring NeoWebScript to an entirely new audience. As always, we will build on the excellent work done by others in the Tcl community wherever possible.

6.4 Cgi.tcl

Although I have stated a number of concerns about the CGI method of producing webpages from programs, I'd like to point out that Don Libes' *cgi.tcl*[16] package provides powerful tools for generating sophisticated HTML constructs from within Tcl. NeoWebScript and cgi.tcl not only play together – developers using cgi.tcl's capabilities from within NeoWebScript have powerful tools for simplifying and enhancing their Tcl-generated HTML content.

6.5 Year 2000 Issues

We do not anticipate any "Year 2000" problems with NeoWebScript itself, as both Tcl 7.6 and Tcl 8.0's date functions are y2k-safe. The Apache group claims to have worked through all of their year 2000 issues. We have budgeted time for testing year 2000 issues in our current release cycle.

6.6 Future Development

Future development interests include creating tools to simplify providing a uniform look across a set of pages. Website integration tools are also an area of interest – NeoWebScript is a natural platform for integrating site-oriented search engines, validating links, etc.

7. Conclusions

NeoWebScript was designed to support NeoSoft, Inc. in providing server-side scripting capabilities to an untrusted user base while providing us with a margin of safety while doing so. As such, NeoWebScript is of particular use to ISPs, web-hosting providers, FreeNets, etc. After two and a half years of use, there have been no known security breaches related to exploits involving NeoWebScript.

Web content developers, including those without significant prior programming experience, have enthusiastically received NeoWebScript.

Sizable applications have been written in NeoWebScript, including two shopping carts, an editable event calendar and to-do list, in-out board, a web-based chat system, and a Yahoo-like hierarchical organizer (http://www.ghofn.org), among many others, all using our standard safe-interpreter interfaces. In fact, NeoWebScript has all but eliminated the need for other server-side tools, by allowing us to quickly and easily develop NeoWebScript equivalents for virtually every other commercial web-related application we have seen.

"Supervisor mode" provides a way to bypass the limits enforced by the architecture of the safe-interpreter/Apache interface in environments populated with a trusted user base, providing trusted developers with significantly more power than they would otherwise have had. Supervisor mode offers many as-yet-unexplored possibilities, and does not yet provide as tight of an integration of capabilities as is seen on the safe side.

A sizable NeoWebScript code base now exists, and we must balance our desire to innovate with the need to maintain compatibility with the work that has already been done.

NeoSoft is committed to continuing to develop and maintain NeoWebScript. We have been able to leverage this work in our own internal intranet applications, use it to lure, win, and keep web developers and the customers they bring with them, and to successfully win and deliver on a high-profile extranet site for a Fortune 100 company.

Hooking Tcl up with Apache is a simple idea that was straightforward to implement. So far, it has been one of the differentiators that have given us an edge in a highly competitive business.

---

[1] Apache Server Project, http://www.apache.org/

[2] Netcraft Webserver Survey, http://www.netcraft.co.uk/survey/

[3] NCSA httpd webserver, ftp://ftp.ncsa.uiuc.edu/Web/httpd/Unix/ncsa_httpd/

[4] John Ousterhout, http://www.scriptics.com/people/john.ousterhout/

[5] TclX 8.0, Mark Diekhans and Karl Lehenbauer, http://www.neosoft.com/tclx

[6] Tcl-DP: a distributed programming extension to Tcl, http://simon.cs.cornell.edu/Info/Projects/multimedia/Tcl-DP/

[7] The Berkeley dbopen Database, Keith Bostic, http://www.sleepycat.com/db.download.html

[8] Apache-SSL secure webserver, http://www.apache-ssl.org/

[9] Tom Poindexter, Oracle/Tcl Interface, *oratcl*, http://www.nyx.net/~tpoindex/tcl.html

[10] POSTGRES SQL database, http://www.postgresql.org/

[11] Thomas Boutell's gd graphics library, http://www.boutelle.com/gd/

[12] HTTP Protocol Specification, RFC1945 - Hypertext Transfer Protocol – HTTP/1.0, http://www.cis.ohio-state.edu/htbin/rfc/rfc1945.html

HTTP Protocol Specification, RFC2068 - Hypertext Transfer Protocol – HTTP/1.1, http://www.cis.ohio-state.edu/htbin/rfc/rfc2068.html

[13] NNTP Network News Transfer Protocol Specification, RFC-977, http://www.cis.ohio-state.edu/htbin/rfc/rfc977.html

[14] HTML Specification, http://www.w3.org/pub/WWW/MarkUp/Wilbur/

[15] Netcraft Webserver Survey (Ibid.)

[16] cgi.tcl, Don Libes, http://expect.nist.gov/cgi.tcl/