SERVING FORTH

Presented by Bob Nash at the November 20th, 2004 Forth Day

Abstract

Although Forth is touted for its code transportability, GUI transportability remains a problem. Thus, Forth programmers must adapt existing GUI tools to their needs, increasing development time and departing from their strengths as Forth programmers.

This paper shows how standard HTML tools and a Forth-based scripting language helped to solve this problem.

Haiku

GUI path, watch your step!
Simple Words help many see the light;
Served right, Forth guides all

Introduction

While re-factoring an existing application, I decided to strip out much of the existing Windows GUI code and provide a user interface with an HTTP Server and a Forth-based scripting language.

The project started out as a straightforward conversion to:

- Provide remote user access,
- Reduce the size of executables by eliminating Windows GUI content, and
- Change program interfaces, as much as possible, to use text files for input, output and configuration (similar to Unix/Linux).

As mentioned in a previous paper, *DISGUISING FORTH*, this conversion was surprisingly successful, earning praise from both users and management not only for increased accessibility but also for expanded functionality.

Much of this increased accessibility and functionality were a natural consequence of using a browser-based user interface. The advantages inherently provided by an HTTP server and browser combination are:

- Remote user access via an existing corporate network
- Browser-based printing and display facilities that are automatically fitted to user PC configurations
- Easy display formatting and user input design using commonly available, but sophisticated, HTML tools
- The user interface can be more easily transported in the event that the core application is re-hosted on a different operating system

SERVING FORTH

Presented by Bob Nash at the November 20th, 2004 Forth Day

Overview

Although I believe that the principles outlined here are generally applicable to a number of computing environments, I describe only one specific combination.

The combination consists of a core application with a browser-based GUI. Both the application and the HTTP server for the GUI are written in SwiftForth™. The HTTP server was written by Mike Ghan of Logix Controls and is furnished as a turnkeyed Windows program with some code examples.

To the disappointment of some (e.g., Elizabeth Rather), I will not dwell on the use of SwiftForth™ for developing Windows™ applications or the core application. Suffice it to say that I am well pleased with the results I have been able to achieve with SwiftForth™.

However, the advantages outlined below should apply equally to other Forth-based Windows™ programming environments with similar facilities, and other programming environments such as Gforth running under Linux with Bernd Paysan's WEB server extensions.

Primarily I describe the Forth scripting capabilities of Mike Ghan's HTTP server to show the kinds of things that are possible with a CGI supercharged with Forth.

Mike Ghan's HTTP Server

At first, the server didn't seem to be much different from other Public Domain programs I had downloaded from the Internet. However, I chose Mike's server because I was interested in using Forth scripting. I also used Mike's server because it was:

1. Available Public Domain (thanks Mike!)
2. Written in SwiftForth™ (but only as a turnkey application)
3. Useful without consuming a large number of computer resources (the executable is only 925K)
4. Equipped with a number of useful configuration and status indicators
5. Proven to be reliable and suitable for process control applications, having been used internally by Mike for several years prior to release
6. Furnished with a CGI scripting interface that used Forth-based display and interface scripts mixed with HTTP statements
7. Provided with coding examples that illustrated important capabilities such as CGI command extensions (in Forth), extension of the server capabilities at startup and file I/O.

Following sections focus on items 6 and 7 above.

A Sample Forth Script

As I understand it, the development of the scripting interface provided with the Mike's server was influenced by Bernd Paysan's article, *A Web-Server in Forth* (see www.jwdt.com/~paysan).

To give you a flavor of how a Forth-based CGI script looks, see Exhibit 1.

As you can see, the script is a mixture of HTML and Forth Words. The <% … %> pairs demark the Forth CGI statements. In this example, part of the text for the error display is handled by standard HTML statements and part is handled by information extracted from the server data and output as HTML.

When I first saw statements like this, I felt that I had found something I could work with productively. It allowed me to leverage my experience with HTML and Forth to display information to a remote user. But wait! There's more!

Startup Extensions

As promising as the sample given in Exhibit 1 seemed when I first saw it, it didn't seem to be anything earthshaking. For an earthshaking example, consider Exhibit 2.

What is so exciting about this script? Well, ***it is executed at startup*** by the server. When I first saw this, I couldn't understand how a Forth definition could be executed by a turnkey application. The turnkey can execute the Forth definition, SHOW-VERSION, because the HTTP server has a dictionary. This dictionary allows the Forth scripting language to be globally extended at startup.

How much dictionary memory is available for extensions at startup? All the memory is available just as in a SwiftForth™ console session.

Pretty cool, huh? But wait! There's more!

Client Extensions

Now consider Exhibit 3, a client script from one of my applications. Note that the script allows the client's dictionary to be extended with Forth definitions such as:

```
: >CLOG  ( a n .. )  <CRLF> COUNT RAW>CLOG  RAW>CLOG ;
```

Also note that the other normal dictionary operations can also be performed, such as conditionally defining a buffer:

```
\ ----- buffer for data file name
[DEFINED] FBUF NOT [IF]
   64 CONSTANT |FBUF|
   CREATE FBUF |FBUF| ALLOT
[THEN]
```

Of course, client definitions only persist as long as the client application is active (e.g., when a client "task" is created at connection).

Because the server must handle multiple client connections, client script dictionaries are limited. In the system as delivered, the default client dictionary is 50K but, ***because the dictionary size is controlled by a VALUE, you can alter the client dictionary size in the startup script***. For example: 75000 TO /SCRIPT-DICT . Thus, the resource usage of client scripts can be tuned, based on the server resources and the likely number of concurrently connected clients.

Other Features

For the sake of brevity (not my strong point), here are some other features provided or supported by Mike's server:

1. File reading and writing from a client script
2. Cookies (read and write)
3. Access to the connection stream (e.g., for IP address, protocols, data values)
4. Forms for user input, including validation scripts
5. Client connection data such as socket numbers, open connections, client connect/terminate times, socket data)
6. Server configuration (e.g., set and view TCP Port, IP address, connection timeout)
7. Style sheets
8. Small footprint – my application with the server and all scripts is less than 1.1 Megabytes, including 925K for the server executable

Summary

Using an HTTP server and a Forth scripting language to provide a user interface can provide:

1. A user interface that is much more transportable (or at least more translatable) than with native GUI APIs such as Windows
2. User functionality that is inherent in the use of a browser, such as remote access, customized printing and flexible displays
3. A convenient development environment, including Forth and HTML tools

The only drawback to using a server-based GUI is that there is some learning curve, especially if you are not familiar with HTML development tools. Also, formatting HTML text did consume more of my time than I expected, but those more experienced with WEB development should have fewer problems.

EXHIBIT 1
SAMPLE FORTH CGI SCRIPT
(BAD-URL.FS)

```
<HTML>
<HEAD>
<TITLE>Page Not Found - Unknown URL (404)</TITLE>
</HEAD>
<BODY>
<H2>There was a problem with your request, sorry ...</H2>
<% ( We're in FORTH! )
   PARA
   S" The page you requested " .<TYPE>
   .( <CODE>) SCRIPT-CLIENT USING WEB-CLIENT-RES GET-RAW-READ
   BL PARSE-WORD 2DROP BL PARSE-WORD 2NIP DKRED-COLOR COLOR .<TYPE> .(
</CODE>)
   S"  does not exist." .<TYPE>
   END-PARA
   PARA .( Please correct your request and try again.)
   END-PARA
   HTML.SERVER
%>
</BODY>
</HTML>
```

EXHIBIT 2
EARTHSHAKING SCRIPT
(CUSTOM.FS)

```
{ ============================================================
Custom Forth Routines for WebServer

  Changes:
  Created 12/10/2002 by Mike Ghan
============================================================ }

HTML DEFINITIONS


: SHOW-VERSION  ( -- )
  BREAK RED BOLD <ATTR> .VERSION   </ATTR> BREAK ;


/FORTH
```

EXHIBIT 3
CLIENT SCRIPT
(ACLM-RcvForm.FS)

```
HTML

\ ----------------------------[ Client Log File ]---------------------------

: RAW>CLOG  ( a n -- )   C" CLIENT.log" ~>>FILE TYPE CONSOLE ;  \ no crlf
: >CLOG  ( a n .. )   <CRLF> COUNT RAW>CLOG  RAW>CLOG ;


\ ----------------------------[ Switch Data File ]---------------------------

\ ----- buffer for data file name
[DEFINED] FBUF NOT [IF]
   64 CONSTANT |FBUF|
   CREATE FBUF |FBUF| ALLOT
[THEN]

\ ----- create dated filename for switch data file
\ (e.g., 040514083417.dat for May 14th, 2004 at 08:34:17)

S" C:\ACLM\DEACTIVATE\" FBUF PLACE  DatedFName FBUF APPEND  S" .dat" FBUF
APPEND

: $>DFILE  ( a n -- )   FBUF ~>>FILE TYPE CONSOLE ;  \ add $ to deactivate file

[DEFINED] DBUF NOT [IF]
   4096 CONSTANT |DBUF|   \ data buffer for deactivate images
   CREATE DBUF |DBUF| /ALLOT
[THEN]

\ Note: Do not append a CRLF to each switch.  This is not only unnecessary,
\       but will also crash MASTA!  Terminating CRLF is optional
: !DBUF  ( a cnt -- )   \ store a $ in the switch data buffer
   DBUF DUP C@ IF APPEND ELSE PLACE THEN
   S"  01 000 000 01" DBUF APPEND ;
```

EXHIBIT 3 (Cont.)

```
\ ------------------------------[ Client Cookie ]----------------------------

\ ----- set name value in cookie
S" Name" SET-CLIENT-COOKIE-NAME
S" First" GET-CLIENT-CONTENT-VALUE >SPAD SPACE>SPAD
S" Last"  GET-CLIENT-CONTENT-VALUE SPAD+
SPAD COUNT DUP 1 > [IF] SET-CLIENT-COOKIE-VALUE [ELSE] 2DROP [THEN]

\ S" password" SET-CLIENT-COOKIE-NAME
\ S" password" GET-CLIENT-CONTENT-VALUE SET-CLIENT-COOKIE-VALUE

0 ( time offset in seconds )  10 ( days ) SET-CLIENT-COOKIE-EXPIRE


\ ----------------------[ Write user data to a log file ]----------------------

\ ----- write Client name to log
SPAD COUNT DUP 1 > NOT [IF]
   2DROP S" No user name entered"  >CLOG
   S" Cookie name is: " >CLOG
   S" Name" GET-CLIENT-COOKIE-VALUE RAW>CLOG
[ELSE]
   S" Name: " >CLOG  RAW>CLOG
[THEN]

\ ----- write Client IP to log
GET-CLIENT-IP DUP [IF] >CLOG [ELSE] 2DROP [THEN]

\ ----- write Client raw switch data to log
S" switches" GET-CLIENT-CONTENT-VALUE DUP 0= [IF]
   2DROP S" No data entered"
[THEN] >CLOG

\ ----- echo data filename to capture date/time
FBUF COUNT >CLOG   S" ------------------" >CLOG
```

EXHIBIT 3 (Cont.)

```
 ( Next Call the "Success" Page/Script -- see HTML-Script.f )
REPLY-FORM-SUCCESS? [IF] ( True = Success URL Specified? )
  DBUF COUNT DUP [IF]
    $>DFILE  <CRLF> COUNT $>DFILE
    CR .( Switch data written to: )  FBUF COUNT TYPE
  [ELSE]
    2DROP  CR .( No switch data to write!)
  [THEN]
\  CR CR .( ACLM-RecvForm.SHTML -- Form Processing is Complete)
[ELSE] ( else )

\  PLAY-WARNING \ Play Windows warning sound on Server
  <HTML>
  <HEAD>
  <TITLE>Missing Success Page</TITLE>
  <META http-equiv="Pragma" content="no-cache">
  </HEAD>
  <BODY>
  <P>
  <HR>
  <STRONG>Missing Success Page! (Data Dump Below)</STRONG>


<% ( ############################ Start Forth ################################## )

  BREAK .( URL: )            GET-HTML-URL TYPE
  BREAK .( HTTP Arguments: )   GET-HTML-ARGS TYPE
  BREAK
  .( <PRE TITLE="Raw HTTP Request">)  \ Add Tooltip!
  .( The Raw HTTP Request was:)
  CR SCRIPT-CLIENT USING WEB-CLIENT-RES GET-HEADERS TYPE
  CR .( The Raw HTTP Content was:)
  SCRIPT-CLIENT USING WEB-CLIENT-RES GET-CONTENT TYPE
  .( </PRE>)

( ############################### End Forth ################################## ) %>

  <HR>
  </P>
  </BODY>
  </HTML>
                                          [THEN]
```

EXHIBIT 4
ACTUAL STARTUP SCRIPT
(Custom.FS)

```
{ ==================[ CUSTOM FORTH ROUTINES FOR WEB SERVER
]=============

   Change History:

   021210 Mike Ghan      Original version created
   040511 rjn            Added DISPLAY-FILE and support code
   040617 rjn            Added GET-FILE?

================================================================================
== }

VARIABLE TRANSACTION#  \ Example of a Global Variable

HTML DEFINITIONS

\ NOTE: All must be relative to a Client task resource (e.g., PAD, R-ALLOC)

: SHOW-VERSION  ( -- )
   BREAK RED BOLD <ATTR> .VERSION   </ATTR> BREAK ;

: DATEDFNAME ( -- a n )  TIME&DATE  2000 - 0 5 0 DO 100 1 M*/ ROT M+ LOOP
   <# # # # # # # # # # # # # #> ;

: NUMERIC?  ( char -- ? )   [CHAR] 0  [CHAR] 9  1+ WITHIN ;


{ ------------------------[ OUTPUT FILE AS HTML ]------------------------- }

: (.X)   ( n #digits -- a n )   0 SWAP <# 0 DO # LOOP #> ;

: DateTime$  ( -- a n )   \ return date/time $
   0 LOCALS| T$ |  64 R-ALLOC TO T$
   (@date) >R  2 (.X)  ( month) T$ PLACE  S" /" T$ APPEND
            2 (.X)  ( day)   T$ APPEND S" /" T$ APPEND
   R> 2000 -   2 (.X)  ( yr)    T$ APPEND S"  " T$ APPEND
   @HOUR  (.) T$ APPEND          S" :" T$ APPEND
   @MINS  2 (.X) T$ APPEND   S" :" T$ APPEND
   @SECS  2 (.X) T$ APPEND
   T$ COUNT   PAD PLACE   PAD COUNT ;
```

EXHIBIT 4 (Cont.)

```
: EXEPATH  ( -- zadr)
   0 GetModuleHandle HERE 255 GetModuleFileName DROP
   HERE ZCOUNT -NAME PAD ZPLACE S" \" PAD ZAPPEND  PAD ;

: .ERR$   ( a n -- )     2 REL-SIZE  RED  BOLD .<TYPE> ;
: .BLUE   ( a n size -- )   REL-SIZE  BLUE BOLD .<TYPE> ;
: .BLACK  ( a n size -- )   REL-SIZE  ( BLACK)  .<TYPE> ;

: ONELINE ( fid -- a n flag )
   PAD 255 ROT READ-LINE 0<> OR 0<> PAD -ROT ;

: DISPLAY-FILE  ( a n size -- )  \ given filename, display the file as HTML
   0 LOCALS| T$ size |  128 R-ALLOC TO T$  T$ PLACE
   T$ COUNT R/O OPEN-FILE ?DUP IF ( bad open)
     NIP ( fid) (THROW)  BREAK .ERR$   BREAK T$ COUNT .ERR$
   ELSE
     BEGIN
       DUP ONELINE WHILE ( fid a n)
       BREAK  size .BLUE
     REPEAT  2DROP CLOSE-FILE DROP
     S" <HR>" TYPE
     BREAK BREAK     DateTime$  1 .BLACK
     BREAK          T$ COUNT   1 .BLACK
   THEN ;


{ -------------------------[ READ FILE TO MEMORY ]-------------------------

0 VALUE |FILE|   \ holds last file size: valid only immed. after GET-FILE?

: GET-FILE? ( adr len -- adr flag)   \ flag is true for a good read
   2DUP R/O OPEN-FILE IF
     DROP  4096 ALLOCATE DROP >R
     S" File " R@ ZPLACE R@ ZAPPEND
     S"  not found." R@ ZAPPEND  R> FALSE EXIT
   THEN ( handle) 0 0 LOCALS| buf n fid | 2DROP
   fid FILE-SIZE 2DROP DUP TO n  TO |FILE|
   n 4 + ALLOCATE DROP TO buf  0 buf n + C!
   buf n fid READ-FILE 2DROP
   fid CLOSE-FILE DROP  buf TRUE ;
  ------------------------------------------------------------------------- }

/FORTH
```

EXHIBIT 4 (Cont.)

```
{ -------------------------[ OPTIONAL HTTP PORT ]------------------------- }

\ 81 WEB-MASTER TCP-PORT ! \ Set HTTP Port

FALSE [IF] \ Test Message Box - Be Careful, No Window Exists Yet
S" Port altered to " PAD ZPLACE
WEB-MASTER TCP-PORT @ (.) PAD ZAPPEND
0 ( No HWND ) PAD  Z" Testing"
MB_SYSTEMMODAL MB_OK OR  MessageBox DROP
[THEN] \ End Test
```

EXHIBIT 5
USAGE NOTES

The following are notes I left for myself when I first brought up Mike Ghan's server. They may be useful if you decide to check it out.

Setup

1. Unzip the distribution and place the executable, WEBSERVER.exe, HTML-Script.f and Custom.f (startup extensions) in the root directory (e.g., C:\WEBSERVER). To include the extensions, also create a batch file, startup.bat, containing the statement: START WEBSERVER Custom.f .
2. HTML and script (*.FS) files are contained in the WEB subdirectory (e.g., C:\WEBSERVER\WEB).
3. Users accessing the server (e.g., via HTTP://10.31.199.16 or localhost) will see INDEX.html contained in the WEB subdirectory.

Cookies

1. The cookie info is stored in a file named something like:
2. C:\Documents and Settings\bnash\cookies\bnash@10.31.199.16[1].txt
3. The form text is stored in the web server root directory as COMMENT.txt

Miscellaneous. Info

(some correspondence with Mike)

Question1: Because your server is a turnkeyed app, approximately how much dictionary is available for extensions at startup? I assume there is also a limit on the Forth definitions contained in the client CGI scripts.

Answer1: At startup, all the memory is available just as in a SwiftForth console session. A client script is limited, the default is 50K, but because it is a VALUE, you can alter it at startup: 75000 TO /SCRIPT-DICT

Miscellaneous. Info (Cont.)

Question2: What is the latest version of your HTTP Server?

Answer2: Latest version is 05/10/2004 which fixed a few problems when INCLUDEing forth source when the server was launched.

Question3: Can you provide a simple explanation of how the HTTP requests from a client are formatted in the socket data?

Answer3: HTTP requests are fairly straightforward - keep reading until a pair of crlfs are received, the connection times out, or until your buffer is full (important). I've implemented the server in SWOOP (object oriented SwiftForth extensions) with each client allocating buffer space on the fly. I gleaned a few ideas from Bernd Paysan's web server: http://wwwejwdt.com/~paysan/httpd-en.html .