

# Introduction

Over the past two years the World Wide Web has experienced a phenomenal surge in popularity. The contributing factors to the popularity of the Web are its inherent client/server architecture; its ability to make physical location and underlying platform incompatibilities transparent to users; and the simplicity with which information can be accessed through a browser such as Netscape Navigator or Microsoft's Internet Explorer. The browser offers a ready-made graphical user interface which is capable of interpreting HyperText Markup Language (HTML) documents. HTML is a tagging language used to create Web documents. For more information on HTML, visit <<http://www.ncsa.uiuc.edu/General/Internet/WWW/HTMLPrimer.html>>

The availability of HyperText Markup Language (HTML) authoring tools combined with the Web's tremendous reach has made it easy for almost anyone to publish for a global audience. Companies are recognizing the need to capitalize on the wide ranging business opportunities offered by the World Wide Web. Although the information published on the Web is multimedia format, the majority of Web pages today simply provide access to manually-maintained "static" documents. This limits business activity

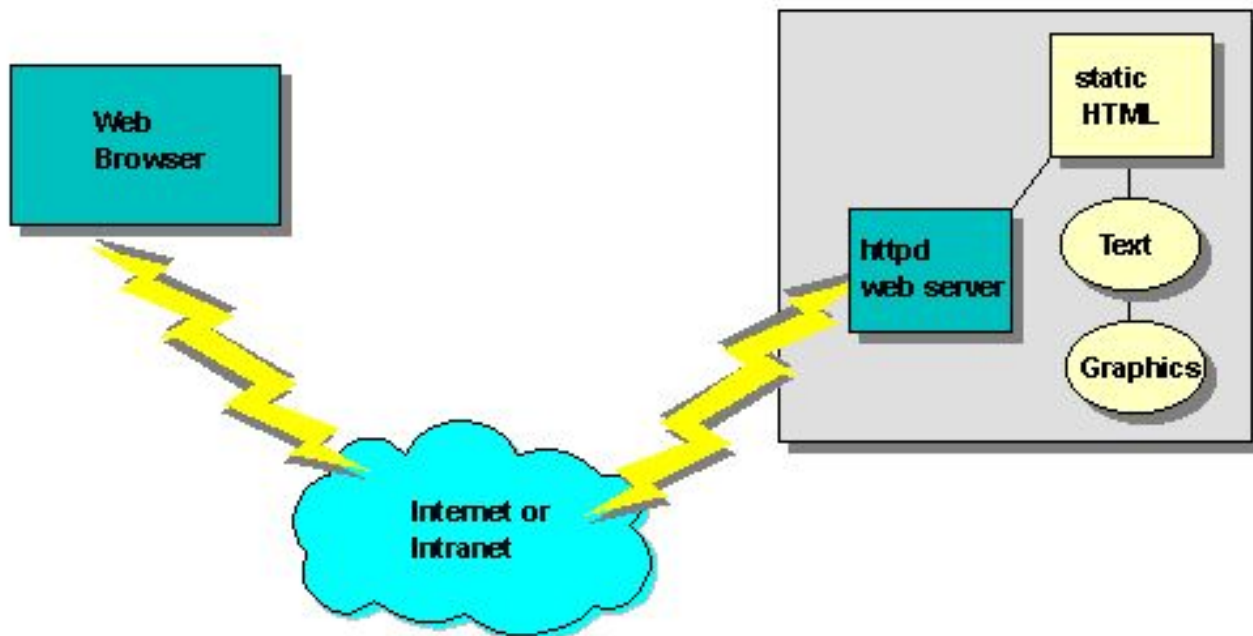


Figure 1. Static Web Page

For example, there are many possible business applications (such as retail and distribution) which could be deployed to the external (Internet) Web, as well as to a company's internal network (Intranet). Static documents would still to be periodically updated. Instead, businesses require the ability to implement "dynamic" on-line applications, integrated with legacy systems and databases where the business data is currently located.

The purpose of this paper is to explain the technology and tools available to access data via an HP 3000 web server and integrate existing or create new HP 3000 applications with a web server.

Initially your Web server (either internal or external) may be serving web pages containing information which does not change often. These are static documents (text files). Now, let's say you want to connect a database to the Web, to allow people in your organization (or the world) to query it. What is the mechanism to do this? The native mechanism is the Common Gateway Interface (CGI). CGI is a standard protocol for interfacing external applications with Web servers. A CGI program is executed, in real-time, so that it can output dynamic information. So, to connect to the database, you need to create a CGI program or script that the Web daemon will execute to transmit information to the database engine, and receive the results back and display them to the client. This is an example of a gateway and is the origin of CGI.

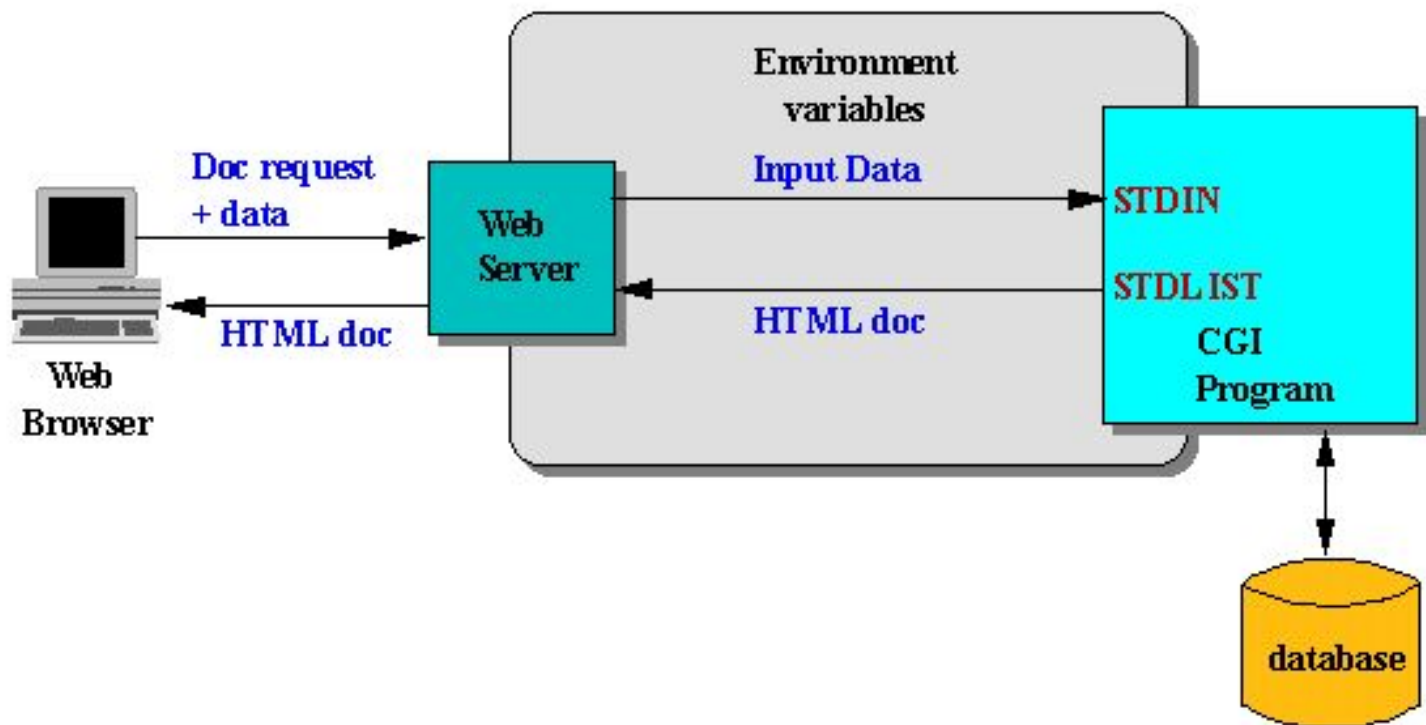


Figure 2. Common Gateway Interface

Any program, in any language can serve as a CGI program as long as it follows these rules:

- The program reads standard input (STDIN) and writes to standard output (STDOUT). (*See Section II for a discussion on CGI with various programming languages.*)
- The CGI program/script must reside in a CGI directory (usually cgi-bin) on the HP 3000 Web server. For example, < [http://jazz.external.hp.com/cgi-bin/omi\\_registration](http://jazz.external.hp.com/cgi-bin/omi_registration) > refers to a CGI program in /cgi-bin on the JAZZ HP3000 web server.

*Note: Some Web servers (Netscape, for example) allow CGI programs to be specified with a particular extension (eg: \*.cgi indicates a CGI program irrespective of where it resides). However, the Open Market WebServer for the HP 3000 requires all CGI programs to reside in the /cgi-bin directory.*

Most uses of CGI scripts are for processing input from Forms. Forms are HTML pages with GUI elements such as buttons, text fields, listboxes, etc. Every form is composed of two parts:

- The HTML code for the form which the user sees displayed on the browser.
- The CGI script running on the web server which processes the contents of the form.

The typical sequence of steps of a CGI script is:

1. Read the contents of the form from STDIN
2. Do processing based on the form data
3. Write the HTML response to STDOUT/STDLIST.

Web protocol has a stateless architecture. A request is made to the server from the client, the information is provided and the connection is dropped. The Web server itself has no way of "remembering" the request. This is standard Web behavior. It is not a problem as long as a complete transaction is contained within a single Web client to Web server connection. However, most applications are more complex and assume that the user is directly connected to the server and that the server is "aware" and "keeping track" of the user. This is a "state-oriented" environment. Therefore, the CGI Web program must contain code to keep track of the user, all requests associated with that user and what information was sent to the user to create the illusion of a "state" environment.

**Note:** It is beyond the scope of this paper to "teach" CGI programming. For more information or tutorials on CGI, visit the following sites:

< <http://hoohoo.ncsa.uiuc.edu/cgi> >

< <http://www.jmarshall.com/easy/cgi/> >

or Search for keyword "CGI" using any one of the popular search engines (Yahoo, Lycos etc.)

In addition, there are numerous books available on CGI programming in bookstore computer sections.

## Data access paradigms

Let's take a look at a "traditional" HP 3000 application. A traditional HP 3000 application is usually a host/terminal-based Cobol application, using VPLUS to handle the communication with the user and Image/SQL intrinsics to read/write data from and to the database.

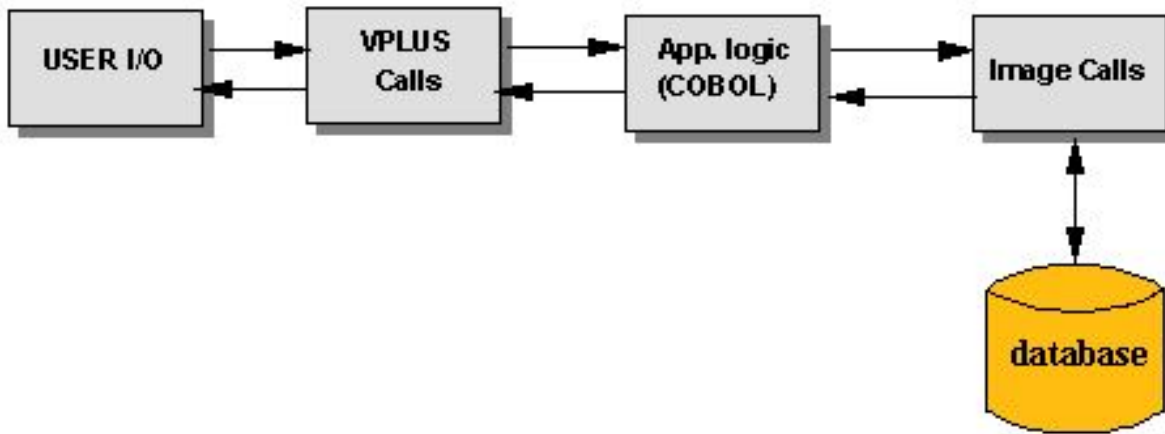


Figure 3: Traditional HP 3000 Application

Now, let's look at that application in a Web environment. Note that the main application logic and communication with the database has not changed. What has changed is the VPLUS calls. Also note that for small to medium sized applications the application logic can be in the CGI program. The diagram below demonstrates an architecture that will be effecient for medium to large sized applications which need to scale.

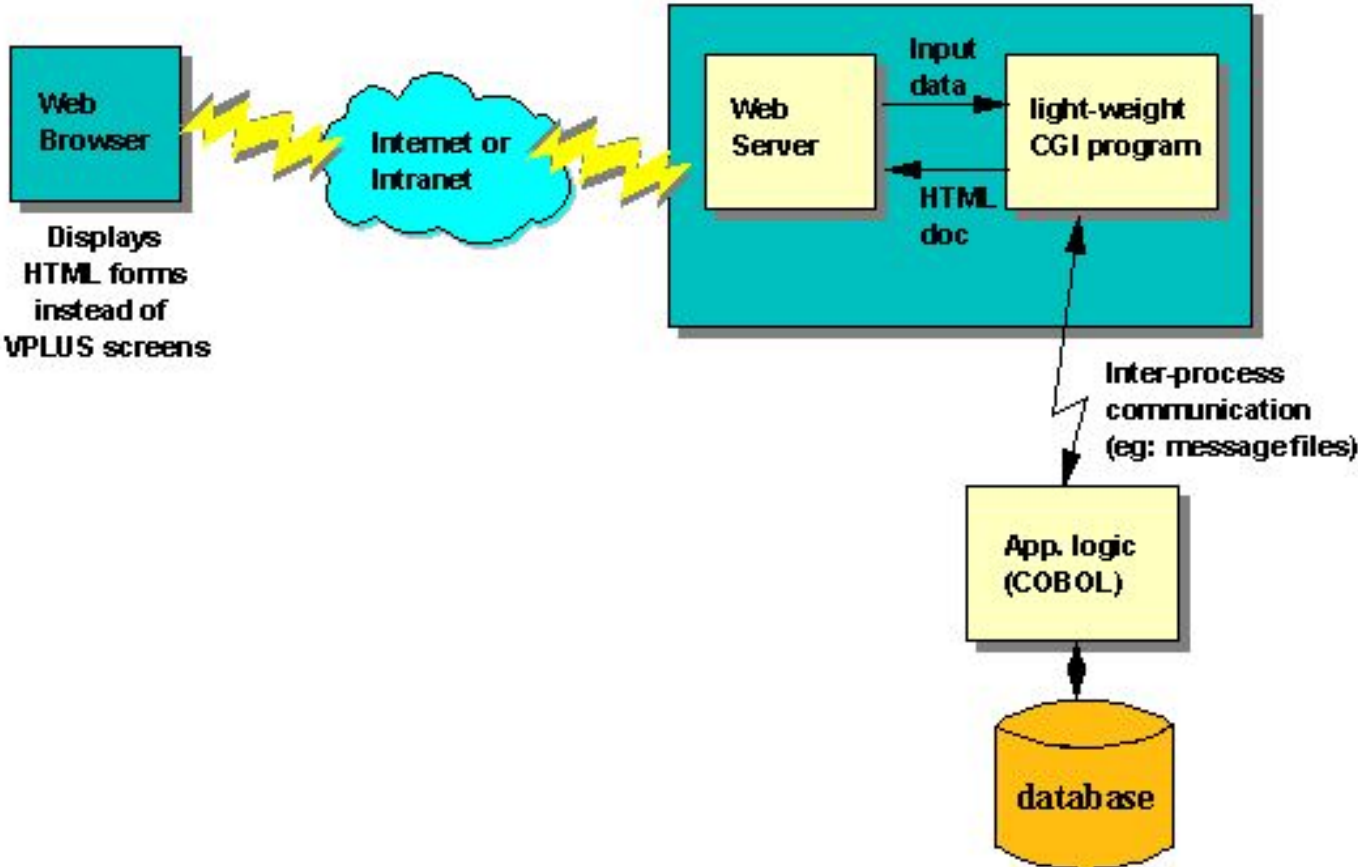


Figure 4. Webified VPLUS application

# CGI programming on the HP e3000

The rest of this paper will demonstrate CGI programming in some popular languages on the HP 3000. CGI was first developed on UNIX derivatives, and since MPE/iX supports POSIX, it is quite easy to write CGI programs on the HP 3000. The same example has been provided in C, PASCAL and PERL to demonstrate the differences in using these languages.

The application is a simple phone book which can store names and numbers. Once completed, this application can be used to add, delete and search for names and phone numbers. This application was intentionally kept small and simple for the sake of discussion. The emphasis here is to demonstrate what is required to write a CGI program and not the function of the program.

The general steps involved in the application are as follows:

1. Read input from STDIN (this input originated from the browser via the magic of CGI)
2. Parse the input string to separate out the individual fields
3. Perform the required operation (In this case ADD/DELETE/SEARCH)
4. Print the MIME header indicating the type of data that is to follow (this goes directly to the web browser via the magic of CGI).
5. Print the output that is to be displayed by the browser. The format of this output has to correspond to the MIME type specified in the MIME header in step 4 (this output also goes to the web browser by the magic of CGI)
6. Exit

To get a working CGI program two components are required:

1. HTML form which will receive input for the CGI program, and,
2. The CGI program itself.

The same HTML form will be the user interface to the phonebook applications in the different languages. Following is the HTML source of the [form](#) which will be used.

```
<HTML>
<HEAD> <TITLE> Phone Book </TITLE></HEAD>
<BODY>
<H1> Phone Book </H1>
<HR>
<FORM ACTION=http://hozone.cup.hp.com/cgi-bin/phonebook METHOD=POST>
<PRE>
Name : <INPUT TYPE="text" NAME=name SIZE=40 MAXLENGTH=40> <p>
Phone : <INPUT TYPE="text" NAME=number SIZE=10 MAXLENGTH=10> <p>
```

```
</PRE>  
<INPUT TYPE="submit" NAME=ACTION VALUE=ADD>  
<INPUT TYPE="submit" NAME=ACTION VALUE=DELETE>  
<INPUT TYPE="submit" NAME=ACTION VALUE=SEARCH>  
<INPUT TYPE="reset" VALUE=CLEAR>  
</FORM>  
<HR>  
</BODY>  
</HTML>
```

The application works as follows:

The ADD button allows you to add the name and number to the phonebook.

The DELETE button deletes the phonebook entry corresponding to the name entered in the name field.

The SEARCH allows you to search the phonebook using names.

The form used here is very simple, containing only two text input fields: one for the name and the other for the phone number. When the user types in the required information and clicks one of the ADD/DELETE/SEARCH buttons, the form data is submitted to the CGI program on the webserver. The URL of the CGI program appears to the right of the "ACTION=" argument in the FORM tag in the HTML form. In the form shown above this is <http://hozone.cup.hp.com/cgi-bin/phonebook>. The form data is submitted in an encoded form which has to be decoded by the CGI program.

The browser encodes the form information in a special way so that the CGI program at the web server can re-extract the individual fields in an easy manner. The encoding process is very simple, the browser just appends all the individual fields in the format "name=value", separated by "&" (ampersand) signs. Here "name" is the name of the field as specified in the HTML form and "value" is the data that the user typed into the field. Spaces are replaced with "+" (plus) signs; Special characters are replaced by the hex value prefixed with the % (percent) sign. Refer to the IETF RFC 1738 <<http://www.ics.uci.edu/pub/ietf/uri/20rfc1738.txt>> to learn more about these encodings. The process of re-extracting the individual fields of the form is commonly referred to as "parsing the CGI input".

To illustrate this encoding with an example, consider the following form:

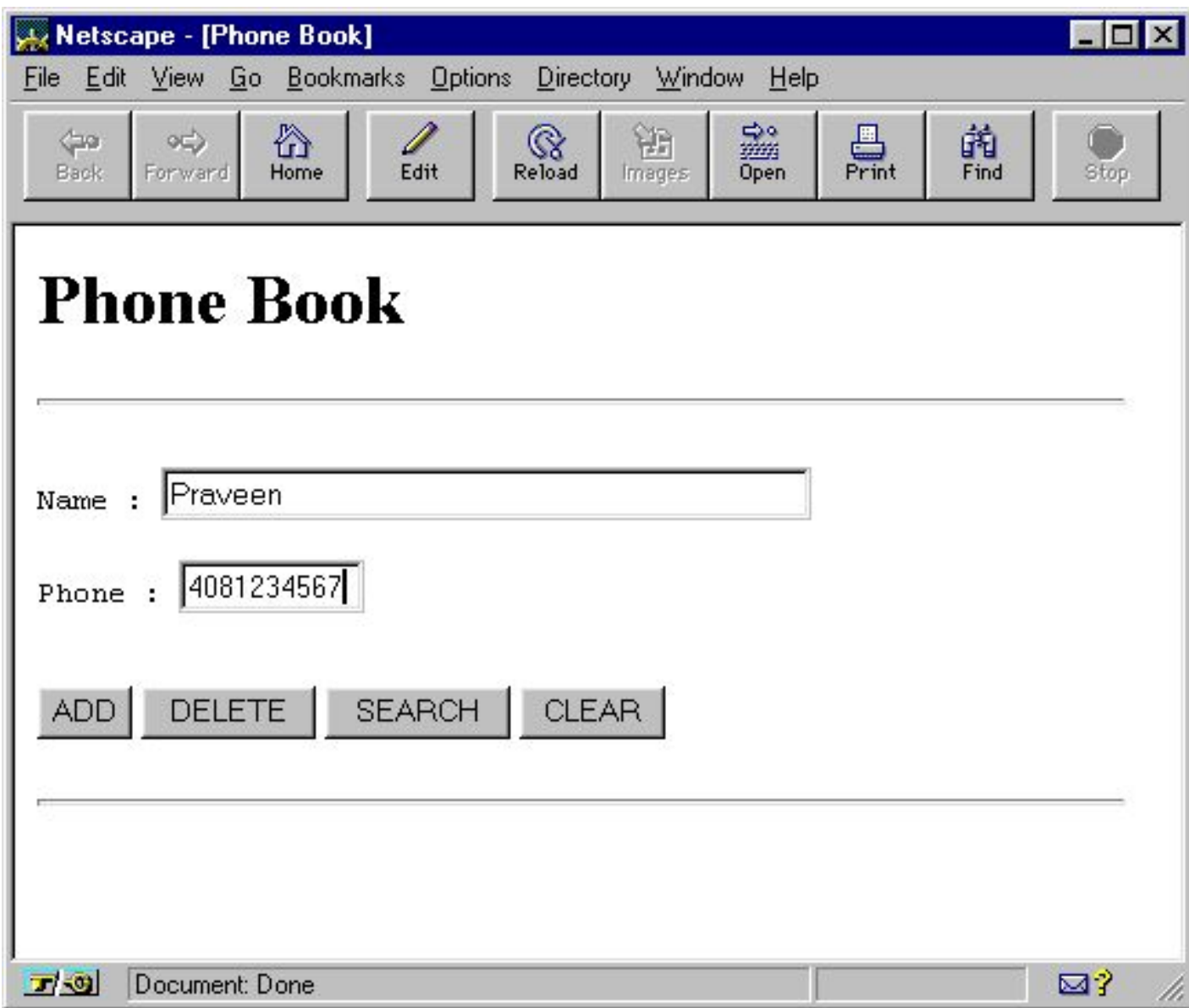


Figure 6. Phonebook HTML

When the user clicks on the "ADD" button, the following encoded string is sent to the CGI application:

```
name=Praveen&number=4081234567&ACTION=ADD
```

Since the format of CGI input is a standard, some good samaritans have put out "CGI parsing libraries" on the WWW which can be downloaded and used. These libraries are available in many popular languages. This paper has CGI parsing functions in three popular languages (C, Perl, and Pascal) and should be sufficient for writing CGI programs in these languages.

## Preparation

Before diving into the examples, set up a new group on your HP 3000 where all these programs can be created and compiled. For example, create a new group called CGI as follows,

## NEWGROUP CGI;CAP=BA,IA,PH

Your HP 3000 webserver should be set up before trying these examples. Make sure that the cgi-bin directory is enabled for your webserver (via directives in the httpd.config file). Refer to your webserver manual for further information. Once the webserver is setup and running, to test whether the your cgi-bin/ has been set up correctly, create the following three line program and save it as 'cgitest' in the cgi-bin/ directory.

```
echo "Content-type:text/plain"
echo
echo "WELCOME TO THE WORLD OF CGI"
```

Now change the permissions so that it is executable by your WWW user ("chmod 755 cgitest" from the POSIX shell).

Using your favorite web browser, load the URL <http://<webserver>/cgi-bin/cgitest> , where <webserver> is the name of your webserver. If you don't see the one liner "WELCOME TO THE WORLD OF CGI" and get an error message instead, that means that your cgi-bin/ directory hasn't been set up correctly for your web server. Refer to your webserver manual for correcting the problem.

***NOTE:** All these programs were tested on MPE/iX 5.5. It is strongly recommended that you do the same since 5.5 contains many fixes for CGI problems (especially for CGI programs NOT written in C or Perl). Moreover, the phonebook example **was not designed to work in a production environment with multiple users**. Why?, because the code was written to demonstrate CGI programming on the HP 3000 and not to demonstrate robust coding techniques. An example where there would be problems is the case when two users simultaneously try to add to & delete from the phone database. It should work fine for a single user though. The exercise of robustizing the code is left to the reader.*

Now without further ado, let's dive into the examples.

## Phonebook application in C

This example consists of two files. One is the [main phone application](#) and the other contains the [CGI input parsing routines](#).

The phonebook application starts of by printing the standard MIME header to STDLIST :

```
printf("Content-type: text/plain\n\n");
```

It uses the environment variable `CONTENT_LENGTH` to determine the size of the CGI input and reads that much from `STDIN`. This data is then parsed to extract the individual "name=value" components. This extracted data is stored in an array of structures containing two fields: *name* and *val*. As pointed to [above](#) the data from the phonebook form has three *name=value* components. The last *name=value* component will tell us what action the user requested by clicking one of the `ADD`, `DELETE`, or `SEARCH` buttons. The main program calls the appropriate phonebook routine, (`phonebook_add`, `phonebook_delete`, or `phonebook_search`) based on the *value* in this third *name=value* component.

This program could be extended to do more complicated things. For example the `phonebook_add()`, `phonebook_delete()` and `phonebook_search()` functions could be modified so that they operate on a phone (Image) database instead of a flat file. The output can also be made more attractive by adding more HTML embellishments in the output. In the given example the output is in plain text format. Note that if you decide to output fancy HTML pages instead of the simple text used in the example you will need to change the MIME header to be `Content-type:text/html`. The MIME header tells the browser how the text from the CGI program is to be interpreted and rendered.

## Compiling and activating the phonebook

To compile these programs, the C compiler is required. Save the programs as `PHONEC` and `CGILIBC` respectively. Use the following commands from the CI prompt to create the phonebook application:

```
CCXL PHONEC, YPHONE, LPHONE; INFO = '-Aa -D_POSIX_SOURCE -DMPE'
```

```
CCXL CGILIBC, YCGILIB, LCGILIB; INFO = '-Aa -D_POSIX_SOURCE -DMPE'
```

```
LINK FROM=YPHONE, YCGILIB; TO=./phonebook.c;RL=/lib/libc.a;cap=ph;posix;
```

Now install the application in the `cgi-bin` directory of your webserver using the following steps:

1. Invoke the shell by running "**SH.HPBIN.SYS -L**". Change to the directory containing the above source code ("**cd ../CGI**")
2. Copy the phonebook application that you have just created to the `cgi-bin` directory of your web server, for example, using the command: "**cp phonebook.c /WWW/OMI/cgi-bin**"
3. Change the owner of this copied file to `USER.WWW` or whoever your web user is, for example,

using the command : "**chown USER.WWW /WWW/OMI/cgi-bin/phonebook.c**". (If you don't have the permissions to do this then you will need to login as MANAGER.SYS and do it.)

4. Next change the permissions of phonebook app. so that it can be executed by the web server. For example, using the command : "**chmod 750 /WWW/OMI/cgi-bin/phonebook.c**"

Now assuming that your webserver is up and running, the phonebook application is ready for use. All that is needed now is an HTML form which can serve as the front-end of the phonebook. Use the [HTML form](#) provided at the beginning of this section. Modify the URL to the right hand side of the "ACTION=" argument in the FORM tag to `http://<webserver>/cgi-bin/phonebook.c` replacing <webserver> with the name of your webserver. Save this HTML document locally in your PC hard-drive or in your webserver's document root. Now once you load this form into your browser you should be able to start using the phonebook application. Remember that you need to add a phonebook entry before you can use the delete or search functions.

**NOTE:** Make sure that the web user (usually USER.WWW - check your httpd.config file) has read/write permissions for the cgi-bin directory since the phonebook application needs to create a flat file in that directory.

## Phonebook application in Pascal

The [Pascal implementation](#) of the phonebook application is very similar to the C version. Unfortunately, this program is not as modularized as the C version since the CGI parsing routines and the main application is all in one source file. But, you should still be able to cut and paste the CGI parsing routines into your own CGI programs.

The main difference between the Pascal version and the C version is that in Pascal you need to use the intrinsics **READ** and **PRINT** to read from STDIN and write to STDOUT respectively. The rest of the program is essentially very similar to the C version.

Note that *input and output* are not included in the program's parameter list. Including this causes problems in the communication between the webserver and the CGI application.

## Compiling and activating the phonebook

To install the program, save it as PHONEP in your CGI group. compile and link using the following commands at the CI prompt:

```
PASXL PHONEP ; YPHONE ; LPHONE  
LINK FROM=YPHONE ; TO= ./phonebook.pas ; RL=/lib/libc.a ; posix
```

If you don't link with `/lib/libc.a` then the `getenv()` function that will be linked in will not be able to read the CGI environment variables.

Copy `phonebook.pas` to the `cgi-bin` directory and set the appropriate permissions and ownership (refer to the [C example subsection](#)). Also make sure that the `cgi-bin` directory is writable by the web user since the phonebook application needs to create a flat file in that directory. Now create the [HTML form](#) that has been provided at the beginning of this section. Modify the URL in the `ACTION` argument of the `FORM` tag to point to your pascal phonebook application (`http://<webserver>/cgi-bin/phonebook.pas`) replacing `<webserver>` with the name of your webserver. Save this on your PC's hard disk or on the webserver. Once you load the form into your web browser you should be able to use the Pascal Phonebook application.

## Phonebook application in Perl

PERL is the most popular language for writing CGI programs. The reason for this is due to PERL's plethora of simple but powerful string manipulation and parsing capabilities. This capability saves having to write the CGI parsing routines which were required in the previous examples.

Perl (version 5) can be downloaded for free from [< http://www.bixby.org/mark/perlix.html >](http://www.bixby.org/mark/perlix.html)

The [Perl implementation](#) of the Phonebook application is similar to the C version except that it is much shorter. The main reason is due to built in functions in Perl which do away with the need for having separate CGI input parsing routines.

The important portion of the Perl code is the main program which does not come under any subroutine heading (the first 50 or so lines). This portion of code does the CGI parsing and calls appropriate subroutines depending on the `ACTION`. The subroutines print the output for the action. Note the first line of the program (`#!/usr/local/bin/perl`). This needs to point to the location of the perl interpreter on your system. If Perl resides elsewhere on your system then modify this line to point to the right place.

## Compiling and activating the phonebook

Perl programs are interpreted so there is no separate compilation step to prepare this application.

To install the program, save it as `phonebook.perl` and copy it to the `cgi-bin/` directory on your webserver. Make sure that the permissions have been set correctly so that it is executable by the web user (refer to the Phonebook Application in [C example subsection](#)). Also make sure that the `cgi-bin` directory is writeable by the web user since the phonebook application needs to create a flat file in that directory. Now just create the [HTML form](#) that has been provided at the beginning of this section. Modify the URL in the `ACTION` argument of the `FORM` tag to point to your perl program (`http://<webserver>/cgi-bin/phonebook.perl`) replacing `<webserver>` with the name of your webserver. Save this on your PC's hard disk or on the webserver. Once you load the form into your web browser you should be able to use the Perl Phonebook

application.

## **COBOL CGI program**

We have contracted with an HP 3000 Electronic Commerce Integrator to provide a CGI programming example with Cobol. When this sample application becomes available it will be placed on the Jazz server (<http://jazz.external.hp.com>) in the Sources and Binaries directory.

## **Guidelines for writing CGI programs**

CGI programs should be light-weight. i.e. they should not be large and complex. The larger they are the heavier the load is on your webserver. The reason is that for each CGI request (user clicking on a button to perform some action) from the browser a copy of the CGI application has to be loaded, executed and destroyed. This can easily degrade the performance of a system with just a nominal CGI request load IF the CGI applications are large. Another fact to keep in mind is that the probability of bugs in your CGI program is proportional to its size and complexity. Bugs in CGI programs can cause security problems, especially if these bugs can be utilized to do devious things. Again, make sure that your web user (usually USER.WWW) does not have any special capability, like PM. Also make sure that all the CGI programs are owned by this user. If you need your CGI programs to have a lot of functionality and perform complex tasks then it is better to isolate this functionality into a separate server program and make the CGI program a small and lightweight client that communicates with this server program via IPC mechanisms (like message files) to get the task done. This will be simple to do with the FastCGI architecture which will become available on the HP 3000 soon. With FastCGI, your CGI program will become a true server, always running (as opposed to being created, executed and destroyed with every request). Moreover, the design of FastCGI is such that you can migrate your CGI program to use FastCGI with very minor modifications.

## **Conclusion**

Overall, writing CGI programs for MPE is not very different from writing them for Unix or any other platform. CGI is an easy way to put a universal Graphical front end on your legacy applications. The COBOL example that will soon be included in this paper will demonstrate how to convert a COBOL application with a VPLUS front end to a CGI program with a browser front end. At HP World '97 in Chicago, a paper detailing writing CGI programs accessing Image data will be presented.

Watch out for more white papers like these that help you web-enable your HP 3000s and do e-commerce on the web.