APPENDIX B

The Future Of Perl & CGI Programming

OVERVIEW

Although Perl 5 has been available for quite a while now, not many CGI authors have taken advantage of Perl 5 specific features. There are several reasons for this. First of all, not all Internet Service Providers (ISPs) have upgraded to Perl 5. It is very frustrating to write a Perl /CGI program for the masses only to discover that a large number of people cannot use your program because they are still stuck using Perl 4. Second, there is not a great deal of information about Perl 5 released yet. The O'Reilly books on Perl discussed in Appendix A are the de facto standard reference guides for the language, yet as of this writing, they have not been updated to include information about Perl 5.

However, the future of Perl 5 looks bright. More and more ISPs are upgrading their Perl executables to version 5. And as time goes on, literature about Perl 5 will be released into the hands of willing and able program-

mers. In addition, Perl 5-specific libraries for handling CGI programming are out now. The most popular of these is Lincoln Stein's CGI.pm module. Because CGI.pm takes advantage of the new features in Perl 5, it is a much more powerful library than CGI-LIB.PL. This in itself will entice more and more CGI programmers over to Perl 5 and CGI.pm. While we will not go into the details of either Perl 5 or CGI.pm, a brief overview should impress upon you some of the advanced features that can be utilized when you decide to make the move to the next generation of Perl and CGI. CGI.PM is also included on the CD-ROM accompanying this book for your own use.

PERL 5

Perl 5 does not actually add a lot more features to Perl 4, but those that it does add are fairly significant. For example, the ability to do object-oriented programming in Perl 5 can change the way that entire Perl programs are written.

Object-Oriented Programming

Perl 5 now includes a number of features that support object-oriented programming (OOP). Basically, OOP allows you to encapsulate data and the functions that operate on that data. Encapsulating data in this way allows you to change the internal structure of an object without affecting the rest of the program. This can make programs more modular and efficient. In addition, rather than writing new objects each time you want to perform a new type of task, you can expand upon more generic objects by inheriting the capabilities of those objects.

Arbitrarily Nested Data Structures

In Perl 4, arrays only have one dimension. In other words, you can only store items as a single list in an array referenced by one index number. Perl 5 enables you to use nested data structures where arrays can reference other variables and other arrays. This allows you to form multi-

dimensional arrays that can model the real-world more effectively. For example, the contents of a chess board or pixels on a computer screen are more easily modeled in terms of X and Y coordinates (2 dimensions). A two-dimensional array that takes two index numbers (X and Y coordinates) can easily represent these types of objects.

The following would be an example of a chess board array mapping numeric index coordinates to the equivalent algebraic notation (letter plus number coordinates) using Perl 5:

```
$chess =[['a1','a2','a3','a4','a5','a6','a7','a8'],
         ['b1','b2','b3','b4','b5','b6','b7','b8'],
         ['c1','c2','c3','c4','c5','c6','c7','c8'],
         ['d1','d2','d3','d4','d5','d6','d7','d8'],
         ['e1','e2','e3','e4','e5','e6','e7','e8'],
         ['f1','f2','f3','f4','f5','f6','f7','f8'],
         ['g1','g2','g3','g4','g5','g6','g7','g8'],
         ['h1','h2','h3','h4','h5','h6','h7','h8']];

print "$chess->[0][0]\n";
print "$chess->[1][2]\n";
print "$chess->[7][7]\n";
```

The above code would produce the following output:

```
a1
b3
h8
```

Regular Expression Enhancements

Perl 5 contains enhancements to the regular expression engine in Perl. There are a couple new pattern match operators for use within a regular expression as well as new arguments for use with the `s` (substitution) and `m` (match) Perl commands.

Grouping Paired Arguments

Perl 5 now allows you to substitute "=>" for the comma "," in a list. This allows you to write your code in such a way that associative array references

are really obvious. For example, in Appendix A, we learned to create an associative array by making a list of pairs. An example of this appears below:

```
%program_authors =
   ("chat", "Gunther Birznieks",
    "shopping cart", "Selena Sol");
```

In Perl 5, you could also create the same array as:

```
%program_authors =
   ("chat" => "Gunther Birznieks",
    "shopping cart" => "Selena Sol");
```

Here it becomes much more obvious which elements are associated with which other elements in the associative array.

Dynamic Modules

Perl 5 allows modules to be loaded on an "as-needed" basis rather than always loading all the routines into memory with the require command like Perl 4 does. The require command in Perl basically loads and compiles the entire library at once which takes up extra time and resources. Using dynamically loaded modules that only load when needed can increase performance significantly if you use a lot of external modules/libraries and do not necessarily use all the routines within them.

Perl 4 To Perl 5 Migration Issues

Unfortunately, with the advanced features that Perl 5 offers, there are also some issues involved with making sure a program will run under Perl 5. Thankfully, the changes are really not that difficult to implement.

First, the at symbol (@) must be escaped inside a string in Perl 5. For example, if you have an assignment such as the following:

```
$myaddress = "gunther@foobar.com";
```

then this must be replaced with

```
$myaddress = "gunther\@foobar.com";
```

In addition, parenthesis must be used to place parameters for calls to subroutines. For example, in Perl 4, the following is acceptable:

```
open FILEHANDLE || &CgiDie("File Not Opened");
```

but in Perl 5, the "FILEHANDLE" argument must be encapsulated with parenthesis:

```
open (FILEHANDLE) || &CgiDie("File Not Opened");
```

In Perl 4, you could sometimes leave quotes off of strings. However, you should always use quotes (or at least some other character) to delimit strings in Perl 5 or you may risk the elements in the string being interpreted as function calls.

Finally, the package delimiter in Perl 5 has changed to double-colons (::) from the apostrophe (') that was used in Perl 4. However, this is not strictly enforced so, for now, it is not vital that you switch over to the new method, but keep in mind that this may not always be the case in the future.

Web Servers Pre-loading Perl (Windows NT)

Some Web Servers on Windows NT now have the capability of loading Perl 5 and running it permanently in memory. Normally, when a Perl CGI program is called from a Web server, Perl is loaded in memory, the script is executed, and Perl unloads in memory. The Web Servers that can keep Perl loaded in memory avoid the extra steps of always loading Perl and then unloading it. This can result in a significant performance increase for your server depending on how many Perl-based CGI programs run on it.

This is not necessarily a Perl 5 specific feature. Currently, only one implementation of Perl 5 on Windows NT supports this functionality. However, we are listing it as an enhancement if you are moving to the flavor of Perl 5 on Windows NT which supports the pre-loading of Perl.

CGI.PM

CGI.pm was written by Lincoln Stein as a Perl 5 module to perform CGI programming tasks. However, CGI.pm goes beyond the features of CGI-LIB.PL for Perl 4 discussed in Chapter Five since it takes advantage of the features in Perl 5.

Object-Oriented

When CGI.pm is used by a Perl 5 program, the "new" operator is used to construct a CGI object. When the "new" operator is called, Perl creates the CGI.pm object and automatically parses and reads in the form variable names that have been passed to the script. From then on, the object can be referred to in order to read the parameters or perform other CGI-related operations. For example, if `$cgi` was set up as the CGI object, we would use the following code to bring it to life:

```
use CGI;
$cgi = new CGI;
```

After these few lines, any reference to CGI routines can be made by using the `$cgi` object that has been created. For example, to get all the form variables that have been passed to your CGI script, simply use the following code:

```
@parameters = $cgi->param;
```

To get a specific form variable such as "email_address" use the following code:

```
$email = $cgi->param('email_address');
```

Values split automatically into arrays

Normally, you can access form variables inside the CGI object by using the param method. For example, to get the value of a "first_name" form

variable from the `$cgi` object, you would use the following code:

```
$firstname = $cgi->param("first_name");
```

But what about multi-valued form variables such as multiple selection list boxes? In CGI-LIB.PL, we have to explicitly call the `SplitParam` function on the returned value in order to split the form variable's values into an array. With CGI.pm, all you have to do is reference the `param` method with an array and the elements will be automatically parsed out into this array without doing any extra work! For example, if I had selected the items "English", "French", and "German" among the languages that I know from a multi-select list box, then the following would break down this form variable value into its constituent elements.

```
@langs = $cgi->param("which_languages_do_you_know");
```

`@langs` would now contain "English," "French," and "German" as elements.

Environment Variables

The environment variables that are associated with CGI are already parsed and defined in the CGI object that CGI.pm creates. For example, to reference the server name, simply use $cgi ->server_name(). Each environment variable is referenced as a method in the $cgi object used above.

Migration From CGI-LIB.PL

Since there are many programs written to use CGI-LIB.PL, CGI.pm includes the capability of being compatible with most of CGI-LIB.PL's syntax. This is accomplished by using the following commands:

```
use CGI qw(:cgi-lib);
&ReadParse;
```

By using these commands, the form variables are read into the "%in" associative array just as they would be in CGI-LIB.PL. However, CGI.pm goes one step further. In addition to being able to use "%in", you can still

use the functions of the CGI object itself to manipulate the form variables and these changes will be reflected automatically in the "%in" associative array.

Dynamic Form Creation

CGI.pm supports a variety of functions to automatically create various headers, footers, forms, input fields, and a lot of other commonly used HTML tags. You may recall from Chapter Ten that CGI-LIB.SOL was written to do a similar task. CGI.pm has more features and is integrated with the main library without needing to call another set of routines. For example, to create a text field with the `$cgi` object in CGI.pm, you would use the following code:

```
print $cgi->textfield(-name=>'first_name',
                      -value=>'Gunther');
```

Saving State

A powerful feature of CGI.pm is the integrated ability to maintain "state." Many chapters of this book rely on the scripts being able to recognize the different users who are currently using an application such as the BBS or Chat without having to ask for their user information over and over again. CGI.pm allows the current form variable values to be written to a file. This file can be reloaded at a later time in order to retrieve the previous state of the form.

Standard HTTP Headers

CGI.pm also provides functions for printing the standard HTTP CGI header ("Content-type: text/html\n\n") as well as many other headers. For example, CGI.pm can send a redirection signal to a user's Web browser to tell them to go to another site just like we discussed previously in Chapter Twenty-Four about the Advertising Tracker. In addition, CGI.PM can generate and retrieve Netscape cookie information.

Conclusion

Perl 5 and CGI.pm clearly have a lot to offer CGI programmers. It is just a matter of time before Internet Service Providers are pressured into upgrading their versions of Perl, and programmers start getting their hands on Perl 5 books. When these things happen, the advantages of using Perl 5 and CGI.pm will play a significant role in the future of CGI programming.