STEPHEN B. JENKINS

offline programmatic generation of Web pages

Stephen is the senior programmer/analyst at the Aerodynamics Laboratory of the Institute for Aerospace Research, National Research Council of Canada. For more information, see http://www.erudil.com.

Stephen.Jenkins@nrc-cnrc.gc.ca

AS PROGRAMMERS, WHEN WE NEED to provide Web-accessible information, two methods usually come to mind: a static one—creating Web pages in an editor or Web development tool, and a dynamic one—creating CGI programs to generate HTML. There is, however, a third, often overlooked, option: offline programmatic generation of Web pages (OPG). By OPG, I mean writing programs to generate HTML documents at the time and location of your choosing, as opposed to CGI programs, where the pages are generated at access time on the computer hosting the Web server.

When to Use OPG

While it may appear, at first glance, that OPG has little to offer over the other two methods, this is not the case. Its primary advantage is that complex HTML documents can be quickly and easily modified, without the need for CGI programs. This is an absolute necessity for people using the services of many of the largest ISPs, since those companies typically only provide a small number of "canned" CGI scripts (e.g., formmail) and do not allow user-written programs.

Even if you do have complete access to your Web server, OPG offers a significant benefit in performance: Web pages can be generated at times when CPU and IO loads are low. This can be especially significant for large Web pages that take a considerable time to generate, such as log file summaries. Rather than create the documents on demand, as CGI programs do, the pages can be generated once (e.g., each night) via a crontab entry. This is also useful for information that is rarely modified (staff email address, phone lists, etc.). The Web pages only need to be generated as often as the data changes.

The third place that OPG makes sense is for pages containing large/multiple tables of data. Even if the information is allegedly unchanging (we've all heard that before!), creating and modifying large tables by hand can be tedious and error prone. Also, as programmers, many of us would rather spend the time writing code to perform a task rather than do it manually.

One final issue is security. While CGI programs can be made as secure as any other software on the Net, inexperienced coders can inadvertently leave themselves open to malicious attacks. For the wary (and the

downright paranoid), OPG offers many of the benefits of CGI, while avoiding all of the potential risks.

A Simple Example

By way of example, I thought I'd show a simplified version of a program I wrote for my wife, Christine, who gives private music lessons. She needed a way to display her schedule and to show which lesson times were available to potential students visiting her Web site. Since her HTML skills are rudimentary and her ISP has difficulties with custom CGI programs, I decided to write some Perl code to generate the Web pages on our home computer. When her timetable changes, Christine modifies the data, double-clicks the program's icon, and then uses a graphical FTP program to upload the newly created Web pages to her service provider.

As so often happens with these kinds of small projects, I decided to add features after I started writing the program. Rather than just generate a public Web page showing the available time slots, I decided to have the program also generate a private page to show such information as the student's initials, other musical commitments, and time off. To keep things as simple and compatible as possible, I decided to put the schedule information in a "DATA" segment at the end of the program, and chose not to use external Perl modules or Cascading Style Sheets (CSS). Figures 1 and 2 show the public and private Web pages generated by the example program shown in Listing 1.

FIGURE 1 Public Schedule Showing Only Available Time Slots

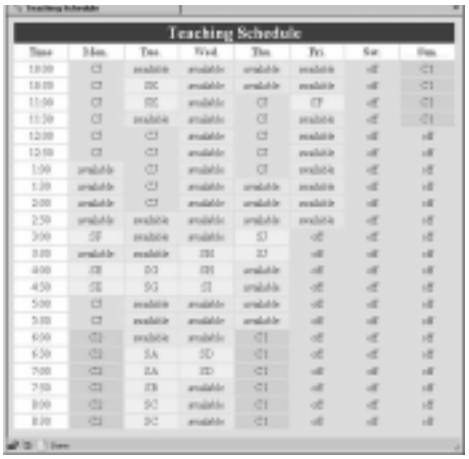

FIGURE 2 Private Schedule Showing All Information

```
#!/usr/bin/perl
use strict;
use warnings;

my $title  = 'Teaching Schedule';
my $colwidth = 'width=75';
my %colorfor = ( 'bg'    => '"#D8E8D8"',
          'hddark'  => '"#336666"',
          'hdlight' => '"#FFFFFF"',
          'choir'   => '"#FFCCCC"',
          'student' => '"#CCFFCC"',
          'avail'   => '"#FFE7CC"',
         );

my $html1 =<<EOF;
<html><head><title>$title</title></head>
<body bgcolor=$colorfor{'bg'} text=$colorfor{'hddark'}>
<table border=0>
<tr><th colspan=8 bgcolor=$colorfor{'hddark'}>
<font color=$colorfor{'hdlight'} size="+2">$title</font></th></tr>
<tr>
EOF

foreach ( qw( Time Mon. Tue. Wed. Thu. Fri. Sat. Sun. ) ) {
  $html1 .= "<th $colwidth bgcolor=$colorfor{'hdlight'}>$_</th>";
}
$html1 .= "</tr>\n";

my $pri = "";
my $pub = "";

while( <DATA> ) {
  next unless /^\d/;

  my $time = substr($_, 0, 6, "");
  $pri .= "<tr><td bgcolor=$colorfor{'hdlight'} align=\"center\">$time</td>";
  $pub .= "<tr><td bgcolor=$colorfor{'hdlight'} align=\"center\">$time</td>";

  my @days = /.{1,4}/g;
  @days = splice @days, 0, 7;
  foreach ( @days ) {
    my $bgc = $colorfor{'avail'};
    if( /\S/ ) {
      if( /CJ|off/ ) { $bgc = $colorfor{'bg'};  }
      elsif( /C\d/ ) { $bgc = $colorfor{'choir'};   }
      else      { $bgc = $colorfor{'student'}; }
      $pri .= "<td bgcolor=$bgc align=\"center\">$_</td>";
      $pub .= "<td></td>";
    } else {
      $pri .= "<td bgcolor=$bgc align=\"center\">available</td>";
      $pub .= "<td bgcolor=$bgc align=\"center\">available</td>";;
    }
  }
  $pri .= "</tr>\n";
  $pub .= "</tr>\n";
}

my $html2 = "</table></body></html>\n";

open  PRI, ">private.html" or die "Oops: $!";
print PRI  "$html1$pri$html2\n";
close PRI;
open  PUB, ">public.html" or die "Oops: $!";
```

```
print PUB  "$html1$pub$html2\n";
close PUB;

__DATA__
Time   Mon  Tue  Wed  Thu  Fri  Sat  Sun
10:00  CJ                      off  C1
10:30  CJ   SK                 off  C1
11:00  CJ   SK        CJ   SF   off  C1
11:30  CJ             CJ        off  C1
12:00  CJ   CJ        CJ        off  off
12:30  CJ   CJ        CJ        off  off
1:00        CJ        CJ        off  off
1:30        CJ                  off  off
2:00        CJ                  off  off
2:30                            off  off
3:00   SF             SJ   off  off  off
3:30             SH   SJ   off  off  off
4:00   SE   SG   SH        off  off  off
4:30        SE   SG   SI   off  off  off
5:00        CJ             off  off  off
5:30        CJ             off  off  off
6:00   C2             C1   off  off  off
6:30   C2   SA   SD   C1   off  off  off
7:00   C2   SA   SD   C1   off  off  off
7:30   C2   SB        C1   off  off  off
8:00   C2   SC        C1   off  off  off
8:30   C2   SC        C1   off  off  off
```

LISTING 1 Schedule Web Page Generator

The first three lines start the program by invoking Perl with the `strict` and `warnings` pragmas. The next nine lines set up some HTML parameters for later use. After that, a "here document" is used to define the HTML head and title elements, as well as setting up the beginning of the main table that will hold the schedule. Next, a `foreach` statement is used to create the table header entries. Until this point, the HTML code has been common to both the private and public pages, but now we need to define two scalars to hold the HTML elements that are unique to each.

At this point, I'll jump down to the end of the program to describe the DATA segment. It consists of a header line followed by multiple lines of data in a simple table, one time slot per row and one day per column. Available slots are expected to be blank, but occupied slots are expected to contain either the string "off", a "C" followed by a number for a choir, or a set of initials, set here to "SA" to "SJ" to denote 10 students or to "CJ" to denote Christine.

A `while` loop reads the rows of the DATA block. Lines that do not begin with a numerical digit are ignored (to allow for blank, formatting, or comment lines). The first six characters are removed and are used as the time string for this row of the HTML table. Next, a regex breaks the rest of the line up into an array of four-character strings. The `@days` array is truncated to seven elements, just in case some extra characters were placed in the data by accident. A `foreach` loop examines each day's entry, and the table cell background color is set to the "available" color. If the entry contains non-whitespace characters, it is compared against two regexes to determine the appropriate background color. For the private page, the schedule's data is written into the table cell, but the public page's cell is left empty to show an unavailable time slot. If the day's entry was only whitespace, the table cells of both private and public pages are set to "available". The table row tags are then closed, and the while loop repeats until all of the

time periods have been processed. In the final few lines of the program, the HTML closing tags are added, and the files are written to disk.

Other Examples

I've used OPG several other times in the past year or so; once was for a local minor hockey league. They wanted to put player statistics (goals, assists, goals per game, points per game, etc.) on their Web site. I wrote two small Perl programs: one for the goalie stats and one for the other players. The data comes from tab-separated text files maintained by one of the league officials. He updates the data files on his home PC and then runs the programs to calculate the stats, sort the player rankings, and generate the Web pages. He then uploads the HTML documents to the league Web server.

As part of my job at the Aerodynamics Laboratory, I've created an event-logging system that receives and records software events from a number of independent computers and stores them in a log file similar to the type used by the Apache Web server [Jenkins]. In order to provide an executive summary of the events that occurred during the previous day, week, month, etc., I wrote a Perl program that reads and analyzes the log files, then generates two Web pages each day: a daily summary and an index of all of the available daily summaries. Since this program can take several minutes to run, and heavily exercises the hard drives, I didn't want it to run on-demand as a CGI would. Instead, I set up a crontab entry to run the process daily (shortly after midnight) and place the output pages in a Web-accessible location.

For a final example, I'd like to talk about three somewhat more complex programs that I wrote for the Fifth International Colloquium on Bluff Body Aerodynamics & Applications (BBAA V). As anyone who's ever been on the organizing committee for a conference will tell you, many of the meeting details (the list of papers, the paper titles, the authors, and the program schedule, just to name a few) are far from static. I wrote Perl programs to take information from the master "database" (an Excel spreadsheet that I read using `Spreadsheet::ParseExcel::Simple`) and generate a list of presentations by topic, an author's index, and the daily program schedules (see Figure 3 for a facsimile).

FIGURE 3 A Portion of the Conference Presentation Program

As new information arrives, the conference administrative assistant updates the spreadsheet and forwards me a copy via email. Within minutes, I can run my programs, update the Web site, and reply that the changes have been made public.

Concluding Remarks

One of the beauties of such a simple concept is that it is completely OS-, Web server-, and Web browser-independent. While OPG is also independent of implementation language, it definitely works best with a language such as Perl, which was designed to manipulate text. Perl also has many other benefits such as low/no cost (open source), portability (available for most modern OSes), easy access to databases through DBI, and a large, freely available archive of modules in CPAN.

For me, the best things about offline programmatic generation are summed up in the first two of the three great virtues of a programmer: laziness and impatience. It enables me to be lazy because with only a few hours' work, I can write a program that enables nonprogrammers to maintain their complex Web pages, putting the responsibility on them. It enables those users to be impatient because rather than wait for someone else to update a Web site, they can do it themselves, immediately. If necessary for the most naïve users, I could even automate the FTP process using `Net::FTP`. This means that to modify their Web site, they would only have to update their text files, Excel spreadsheets, or database entries, and double-click a desktop icon on their PCs.

REFERENCE

Jenkins, S.B., "A Web-Based Environment to Support Aerodynamic Testing," *IEEE Aerospace and Electronic Systems Magazine* 19:1 (January 2004), p. 3.