

Beginning Perl Programming (X52.9543.001)**Handout 6: June 29, 2000*****logHit.pl***

Here's a routine straight out of the Web/perl package used by a large company. It takes information about the client making the Web request and updates hit log and DB Hash files.

```
# Logs relevent data for a "hit" to a log file.
# On problems, we call dieCgi, unless the -noDie option is set to nonzero.
# Returns undef on success, or an error string on failure
# (if the -noDie option is set to nonzero)
#
# Options:
# -noDie      If set to non-zero, this routine return the text of any
#             error message.
#
# -noMetrics  Only write the hit.log file, not the metrics DB_HASH files.
#
# Example:
# my ($errorText) = LogHit ("-noDie" => 1);
# if ($errorText) { die $errorText; }

sub LogHit {
    my (%options) = @_ ;

    my ($remHost) = $ENV{'REMOTE_HOST'};
    my ($remAddr) = $ENV{'REMOTE_ADDR'};
    my ($remUser) = $ENV{'REMOTE_USER'};

    my ($fname) = "../log/hit.log";
    my ($hitHandle) = new FileHandle;

    if (! open ($hitHandle, ">> " . $fname)) {
        my ($complaint) = "cannot open $fname: $!";
        if ($options{'-noDie'}) {
            return $complaint;
        } else {
            &dieCgi ($complaint);
        }
    }

    # Grab a writer-block on the file

    if (! &Sync::writer ($hitHandle)) {
        my ($complaint) = "error synchronizing on $fname: $!";
        if ($options{'-noDie'}) {
            return $complaint;
        } else {
            &dieCgi ($complaint);
        }
    }

    # We used to fetch the time via:
    #     = localtime ($^T);
    # HOWEVER, that got the time at the startup of the script, which is
    # bad idea in the case of a long-lived Interface Process.
```

```

my ($sec, $min, $hour, $mday, $mon, $year, $yday, $yday)
    = localtime (time);

print $hitHandle "URL: ", &MyURL, "\n";

if ($remUser) {
    printf $hitHandle
        "Date: %02d/%02d/%04d @ %02d:%02d:%02d %15s %s <%s>\n",
        $mon + 1, $mday, $year + 1900, $hour, $min, $sec,
        $remAddr, $remHost, $remUser;
} else {
    printf $hitHandle
        "Date: %02d/%02d/%04d @ %02d:%02d:%02d %15s %s\n",
        $mon + 1, $mday, $year + 1900, $hour, $min, $sec,
        $remAddr, $remHost;
}

unless ($options{'-noMetrics'}) {

    # Log it to the usage database by incrementing the count of
    # hits for this host.
    # A NOTE on locking: We keep our lock on the hit log file
    # established above, and use it to coordinate access
    # to the db_usage database.

    my ($dbdir) = "../db_usage/";
    my ($access) = O_RDWR | O_CREAT;
    my ($mode) = 0666;

    my (%dateCountTable);
    my ($todayDate) = sprintf ("%04d-%02d-%02d",
        $year + 1900, $mon+1, $mday);
    my ($dateCountDbHandle) = tie (%dateCountTable, "DB_File",
        $dbdir . "usage_date_count.hsh",
        $access, $mode, $DB_HASH);
    $dateCountTable{$todayDate}++;
    untie %dateCountTable;

    my (%dayCountTable);
    my ($dayCountDbHandle) = tie (%dayCountTable, "DB_File",
        $dbdir . "usage_day_count.hsh",
        $access, $mode, $DB_HASH);
    $dayCountTable{$yday}++;
    untie %dayCountTable;

    my (%hostCountTable);
    my ($hostCountDbHandle) = tie (%hostCountTable, "DB_File",
        $dbdir . "usage_host_count.hsh",
        $access, $mode, $DB_HASH);
    $hostCountTable{$remHost}++;
    untie %hostCountTable;

    my (%hourCountTable);
    my ($hourCountDbHandle) = tie (%hourCountTable, "DB_File",
        $dbdir . "usage_hour_count.hsh",
        $access, $mode, $DB_HASH);
    my ($formatHour) = sprintf ("%02d", $hour);
    $hourCountTable{$formatHour}++;
    untie %hourCountTable;
}

```

```

    if (!$remAddr) { $remAddr = "Unknown"; }
    my (%ipCountTable);
    my ($ipCountDbHandle) = tie (%ipCountTable, "DB_File",
        $dbdir . "usage_ip_count.hsh",
        $access, $mode, $DB_HASH);
    $ipCountTable{$remAddr}++;
    untie %ipCountTable;

    if (!$remUser) { $remUser = "Unknown"; }
    my (%userCountTable);
    my ($userCountDbHandle) = tie (%userCountTable, "DB_File",
        $dbdir . "usage_user_count.hsh",
        $access, $mode, $DB_HASH);
    $userCountTable{$remUser}++;
    untie %userCountTable;
}

# Free our writer's block

if (! &Sync::free ($hitHandle)) {
    my ($complaint) = "error freeing lock on $fname: $!";
    if ($options{'-noDie'}) {
        return $complaint;
    } else {
        &dieCgi ($complaint);
    }
}

if (! close ($hitHandle)) {
    my ($complaint) = "cannot close $fname: $!";
    if ($options{'-noDie'}) {
        return $complaint;
    } else {
        &dieCgi ($complaint);
    }
}

return undef;
}

```

map_gen.pl

```
#!/App/Perl/bin/perl -w
#
# map_gen.pl
#
# Perl script to generate a map grid database file.
#
# (c) Copyright 2000 by Clint Goss, All Rights Reserved.

use strict 'vars';

# Pick up our options

use Getopt::Std;
use vars qw($opt_d $opt_D $opt_f $opt_h);
if (! getopts("f:dDh")) { &usage(); exit; }

sub usage {
    print <<END_OF_HELP;
usage: $0 [options]

Generate a map grid database file.

Options:
  -f FILE    Specify the output file. Default is map_grid.db
  -d         Turn on moderate debugging
  -D         Turn on very verbose debugging
  -h         Print this help message
END_OF_HELP
}

if ($opt_h) { &usage(); exit; }

# Parameters to this program

my ($debug) = $opt_D ? 2 : $opt_d ? 1 : 0;
my ($mapGridFilename) = $opt_f ? $opt_f : "map_grid.db";

# File handling

open (MAP_GRID, ">$mapGridFilename") || die "Can't open $mapGridFilename: $!";

# Standard sizes in the file.

my ($headerByteCount) = 10;
my ($bytesPerFileName) = 20;

# Populate the file names

my (@rowNames) = qw(m n o p q r);
my (@colNames) = qw(0 1 2 3 4 5 6 7 8 9 a);
my ($rowCount) = scalar @rowNames;
my ($colCount) = scalar @colNames;

# Write the row count and the column count

seek (MAP_GRID, 0, 0);
print MAP_GRID pack("cc", $rowCount, $colCount);
```

```

# Write each of the file names into the database file

my ($row, $col);
foreach $row (0 .. $rowCount-1) {
    foreach $col (0 .. $colCount-1) {

        seek (MAP_GRID,
              $headerByteCount +
              (($row*$colCount) + $col) * $bytesPerFileName,
              0);

        print MAP_GRID pack ("A20",
                              $rowNames[$row] . $colNames[$col] . ".jpg");
    }
}

close MAP_GRID;
exit;

```

map_info1.pl

```

#!/App/Perl/bin/perl -w
#
# map_info1.pl
#
# Perl script to output the information for a map grid location.
#
# (c) Copyright 2000 by Clint Goss, All Rights Reserved.

use strict 'vars';

# Pick up our options

use Getopt::Std;
use vars qw($opt_d $opt_D $opt_f $opt_h $opt_p $opt_v);
if (! getopts("dDf:hp:v")) { &usage(); exit; }

sub usage {
    print <<END_OF_HELP;
usage: $0 [options]

Produce the information for a map grid location.

Options:
-d          Turn on moderate debugging
-D          Turn on very verbose debugging
-f FILE    Specify the map grid database file. Default is map_grid.db
-h          Print this help message
-p RxC     Position (required!).
            The R and C are Row and column positions (zero-based).
            For example, -p 0x17 indicates the first row, 18th column.
-v          Turn on verbose reporting mode
END_OF_HELP
}

if ($opt_h) { &usage(); exit; }

```

```

# Parameters to this program

my ($debug) = $opt_D ? 2 : ($opt_d ? 1 : 0);
my ($mapGridFilename) = $opt_f ? $opt_f : "map_grid.db";
my ($verbose) = $opt_v;

# Parse the required -p position parameter

my ($position) = $opt_p;
if (! $position) { &usage(); exit; }
my ($rowPosition, $colPosition) = split ("x", $position);
if (! $rowPosition) { &usage(); exit; }
if (! $colPosition) { &usage(); exit; }

# Open the input file.

open (MAP_GRID, "<$mapGridFilename") || die "Can't open $mapGridFilename: $!";

# Pick up the number of rows.

my ($header);
read (MAP_GRID, $header, 2) || die "Can't read $mapGridFilename: $!";
my ($rowCount, $colCount) = unpack ("cc", $header);

# Standard sizes in the file.

my ($headerByteCount) = 10;
my ($bytesPerFileName) = 20;

# Pick up the four file names.

my ($file_A_0) = &mapFileName ($rowPosition, $colPosition,
                               $rowCount, $colCount);

my ($file_A_1) = &mapFileName ($rowPosition, $colPosition + 1,
                               $rowCount, $colCount);

my ($file_B_0) = &mapFileName ($rowPosition + 1, $colPosition,
                               $rowCount, $colCount);

my ($file_B_1) = &mapFileName ($rowPosition + 1, $colPosition + 1,
                               $rowCount, $colCount);

# Produce the output for this program

printf "Map files: %s %s\n", $file_A_0, $file_A_1;
printf "Map files: %s %s\n", $file_B_0, $file_B_1;

close MAP_GRID;
exit;

```

```

# Read a single file name at a given row and column position from MAP_GRID.

sub mapFileName {
    my ($rowPos, $colPos, $rowCount, $colCount) = @_;

    # Move the file pointer to the start of the file name.

    seek (MAP_GRID,
          $headerByteCount +
          (($rowPos*$colCount) + $colPos) * $bytesPerFileName,
          0);

    # Read the file name and strip any blanks.

    my ($filename);
    read (MAP_GRID, $filename, $bytesPerFileName)
        || die "Can't read $mapGridFilename: $!";

    $filename =~ s/^\s+//;
    $filename =~ s/\s+$//;
    return $filename;
}

```

Output of map_info1.pl under MKS Toolkit 6.1 under Win98 using ActivePerl

```
➤ map_info1.pl -p 2x2
```

```
Map files: o2.jpg o3.jpg
Map files: p2.jpg p3.jpg
```

Reading for Next Class

Llama book: Page 192, Perl References
Reference Guide, Object Oriented Programming (very short!)
Optional: Camel book, Chapter 5

Homework Assignment

1. Download the code of map_gen.pl and run it on your system, to produce a map_grid.db file.¹
2. Generalize the program map_info1.pl in the following ways:
 - (a) If the user has requested a grid location which is out of the bounds given by the map_grid.db file, output the file name off-map. For example, if the right side of the map has been reached, you might output:

```
Map files: m9.jpg off-map
```
 - (b) Output a line which gives the next grid reference to the right of the grid reference given by the -p option. The line should be formatted, for example:

```
Right: 2x4
```

However, only output this grid reference if all the maps in the grid reference to the right are present. If not all the maps are present in the map_grid.db file, then output:

```
Right: off-map
```

- (c) Emit similar lines for the grid above, below, and to the left of the grid reference given by the `-p` option. These lines should be tagged with “Up”, “Left”, and “Down” in place of “Right”.
- (d) Emit the lines for “Up-Half”, “Down-half”, “Left-half” and “Right-half” which move to grid references moved over by half the width of the map.

So, if this might be reasonable output for an option `-p 1x1`:

```
Map files: n1.jpg n2.jpg
Map files: o1.jpg o2.jpg
Right: 1x3
Right-half: 1x2
Left: off-map
Left-half: 1x0
Up: off-map
Up-half: 0x1
Down: 3x1
Down-half: 2x1
```

- (e) (*Extra Credit*) The hard-coded map returned by `map_info1.pl` is 2x2 in size. Accept a new option, `-s RxC`, which specifies the size of the grid of maps which the user wants to see. The default should be 2x2. For example, `-s 3x4` is a request for a map containing 3 rows and 4 columns of JPG files. Note that the output of `Left`, `Right`, `Up`, and `Down` will change as the `-s RxC` changes, as well as the `Left-half`, etc.

Hand in a printout of your modified program along *with sample output* at the beginning of next session.

3. Prepare a one-sentence answer to the question: “Is programming a science, a craft, or an art?”¹

4. Download the handout for Session 7. This should be available at <http://www.goss.com/perl> by Wednesday, July 5 at 9AM. Bring a printout to Session 7.¹

Notes: ¹Not to be handed in.

This handout is Copyright © 2000 Clint Goss, All Rights Reserved.