```
#!/usr/bin/perl
# ============================================================================
# FILE: DnB_Webserver.pm                                          9-14-2020
#
# SERVICES:  DnB WEBSERVER and RELATED FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides a basic webserver interface to the D&B model
#    railroad. There is a lot going on in this module since it uses perl,
#    webserver, CSS, javaScript, and HTML constructs to realize the necessary
#    functions. The data decoration CSS might be a bit much.
#
#    The webserver is started during the DnB.pl initialization phase. A
#    message is output on the console detailing the IP:Port value that is
#    used to connect an external web browser. This IP:port value is manually
#    entered into the browser's address bar. Upon successful connection, the
#    the D&B Model Railroad home page is displayed.
#
#    The webserver code monitors the IP:Port for browser requests. Validated
#    requests result in a corresponding data page to be created and sent to
#    the browser for display to the user. All dynamically created HTML pages
#    are stored and served from /dev/shm. Static files are served from the
#    DnB.pl confirgure $WebRootDir directory.
#
#    The webserver runs as a forked child process. As such, it does not have
#    access to current operational data. The main loop in DnB.pl therefore
#    writes the needed data to the /dev/shm directory about once per second.
#    This data is used to build the data pages that are sent to the browser.
#
# PERL VERSION: 5.24.1
#
# ============================================================================
use strict;
# ----------------------------------------------------------------------------
# Package Declaration
# ----------------------------------------------------------------------------
package DnB_Webserver;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   Webserver
);

# ----------------------------------------------------------------------------
# External module definitions.
use HTTP::Daemon;
use HTTP::Status;
use Sys::HostAddr;
use POSIX qw(strftime);
use DnB_Message;

# ============================================================================
# FUNCTION:  Webserver
#
# DESCRIPTION:
#    This routine is called during main program startup to launch the webserver
#    as a backgroung process. Directing an external web browser to the Rpi IP
#    (or hostname) and $ListenPort displays the home web page. Links on the
#    home page provide access to the other data pages; e.g. turnout positions.
```

```
61  #
62  #    Depending on the browser version, the raw IP:Port might be needed for the
63  #    initial connection. 'sudo ifconfig' will display the network interfaces
64  #    on the Rpi. Uncomment one of the methods below to get the IP address.
65  #
66  # CALLING SYNTAX:   (using Super::Fork)
67  #    $pid = fork {sub => \&Webserver, args => [ $WebRoot, $ListenPort,
68  #                                               $WebDataDir ] };
69  #
70  # ARGUMENTS:
71  #    $WebRoot            Webserver document root directory.
72  #    $ListenPort         Port to listen to for connections.
73  #    $WebDataDir         Directory for dynamic data content.
74  #
75  # RETURNED VALUES:
76  #    non-zero pid = Success,  undef($pid) = Error.
77  #
78  # ACCESSED GLOBAL VARIABLES:
79  #    $main::ChildName
80  # ============================================================================
81  sub Webserver {
82     my($WebRoot, $ListenPort, $WebDataDir) = @_;
83     my($result, $host, $url, $ipp, $daemon, $connect, $getRequest, $method);
84
85     # Remove any previous dynamic HTML files. The other data files are handled
86     # by the main loop code.
87     unlink glob("$WebDataDir/file_*.html");
88
89     # Define some working variables and display webserver connection point.
90     $host = `/bin/hostname`;
91     if (($? >> 8) != 0) {
92        &DisplayError("Failed to get hostname. Webserver not started.");
93        exit(1);
94     }
95     chomp($host);
96     $url = join(':', $host, $ListenPort);
97
98     # Determine raw ip:port for alternate browser connection point.
99     my($sha) = Sys::HostAddr ->new();
100    # $ipp = join(':', $sha->main_ip(), $ListenPort);
101    # $ipp = join(':', $sha->first_ip(), $ListenPort);
102    $ipp = join(':', $sha->first_ip('wlan0'), $ListenPort);
103    undef($sha);
104
105    $main::ChildName = "Webserver Process $ipp";
106
107    # Establish connection point and start the webserver.
108    unless($daemon = HTTP::Daemon->new(LocalPort => $ListenPort, ReuseAddr => 1,
109                     Family => AF_INET, Type => SOCK_STREAM,  Listen => 5)) {
110       &DisplayError("Webserver failed to start: $!");
111    }
112    else {
113       &DisplayMessage("Webserver started.  Client connection url: $url or $ipp");
114
115       # Process client connections.
116       while ($connect = $daemon->accept) {
117          while ($getRequest = $connect->get_request) {
118             $method = $getRequest->method;
119             if ($method eq 'GET') {
120                &NewConnection($WebRoot, $getRequest, $connect, $WebDataDir);
```

```
121                $connect->close;
122                last;
123             }
124             else {
125                $connect->send_error(RC_BAD_REQUEST, 'Unsupported method: $method');
126                &DisplayError("Webserver, unsupported method: $method");
127             }
128          }
129       }
130    }
131    &DisplayMessage("Webserver terminated.");
132    exit(0);
133 }
134
135 # ============================================================================
136 # FUNCTION:  NewConnection
137 #
138 # DESCRIPTION:
139 #    This routine is called to process new webserver connections. HTTP::Daemon
140 #    class methods are used to obtain request parameters ($Request) and send the
141 #    response to the $Connect attached browser. Supported requests are processed
142 #    by the RequestHandler() code.
143 #
144 #    All subsequent subroutines obtain their working parameters from the $Request
145 #    hash.
146 #
147 # CALLING SYNTAX:
148 #    $result = &NewConnection($WebRoot, $Request, $Connect, $WebDataDir);
149 #
150 # ARGUMENTS:
151 #    $WebRoot             Webserver document root directory.
152 #    $GetRequest          Request data structure.
153 #    $Connect             Connection socket structure.
154 #    $WebDataDir          Directory for dynamic data content.
155 #
156 # RETURNED VALUES:
157 #    0 = Success,  1 = Error.
158 #
159 # ACCESSED GLOBAL VARIABLES:
160 #    None.
161 # ============================================================================
162 sub NewConnection {
163    my($WebRootDir, $GetRequest, $Connect, $WebDataDir) = @_;
164    my(@array);
165
166    my(%dispatch) = ('top' => \&TopPageData, 'grade' => \&GradePageData,
167                     'block' => \&BlockPageData, 'sensor' => \&SensorPageData,
168                     'signal' => \&SignalPageData, 'turnout' => \&TurnoutPageData,
169                     'live' => \&LivePageData);
170
171    &DisplayDebug(1, "NewConnection, ========================================");
172
173    my(%request) = ('OBJECT' => $GetRequest->uri->path, 'ROOT' => $WebRootDir,
174                    'SHARE' => $WebDataDir, 'BUILDER' => '', 'PAGE' => '',
175                    'TYPE' => '');
176
177    # Validate the request and call the request handler. If no page is specified,
178    # the 'Top' page is served. Only a limited set of OBJECT requests are honored.
179    &DisplayDebug(1, "NewConnection, object: '$request{OBJECT}'");
180    if ($request{OBJECT} =~ m#^/(.*)#) {
```

```
181         $request{PAGE} = $1;
182         $request{PAGE} = 'top' if ($request{PAGE} eq '');
183         if (exists($dispatch{ $request{PAGE} })) {
184            $request{BUILDER} = $dispatch{ $request{PAGE} };
185            $request{TYPE} = 'text/html; charset=utf-8';
186            &RequestHandler($GetRequest, $Connect, \%request);
187         }
188         elsif ($request{PAGE} =~ m/\.ico$/i) {
189            $request{PAGE} = join('/', $WebRootDir, $request{PAGE});
190            $request{TYPE} = 'image/x-icon';
191            &RequestHandler($GetRequest, $Connect, \%request);
192         }
193         elsif ($request{PAGE} =~ m/\.(css)$/i or
194                $request{PAGE} =~ m/\.(webmanifest)$/i) {
195            $request{PAGE} = join('/', $WebRootDir, $request{PAGE});
196            $request{TYPE} = 'text/$1';
197            &RequestHandler($GetRequest, $Connect, \%request);
198         }
199
200         # For live page overlay files, send the file indicated in the corresponding
201         # .dat file that was set by the main loop.
202         elsif ($request{PAGE} =~ m/([h|m|y]-overlay\.dat$)/i or
203                $request{PAGE} =~ m/(L\d\d-overlay\.dat$)/i or
204                $request{PAGE} =~ m/(GC\d\d-overlay\.dat$)/i) {
205            &ReadFile("$WebDataDir/$1", \@array, '');
206            if ($array[0] =~ m/\.(gif)$/i or $array[0] =~ m/\.(jpg)$/i or
207                $array[0] =~ m/\.(png)$/i) {
208               $request{TYPE} = join('/', 'image', $1);
209               $request{PAGE} = join('/', $WebRootDir, $array[0]);
210            }
211            if (-e $request{PAGE}) {
212               &RequestHandler($GetRequest, $Connect, \%request);
213            }
214            else {
215               $Connect->send_error(RC_NOT_FOUND, 'File: $request{PAGE}');
216               &DisplayError("NewConnection, File not found: $request{PAGE}");
217            }
218         }
219         elsif ($request{PAGE} =~ m/\.(gif)$/i or $request{PAGE} =~ m/\.(jpg)$/i or
220                $request{PAGE} =~ m/\.(png)$/i) {
221            $request{TYPE} = join('/', 'image', $1);
222            $request{PAGE} = join('/', $WebRootDir, $request{PAGE});
223            if (-e $request{PAGE}) {
224               &RequestHandler($GetRequest, $Connect, \%request);
225            }
226            else {
227               $Connect->send_error(RC_NOT_FOUND, 'File: $request{PAGE}');
228               &DisplayError("NewConnection, File not found: $request{PAGE}");
229            }
230         }
231         else {
232            $Connect->send_error(RC_BAD_REQUEST, 'File: $request{PAGE}');
233            &DisplayError("NewConnection, Bad request: $request{PAGE}");
234         }
235      }
236      else {
237         $Connect->send_error(RC_BAD_REQUEST, "Can't parse object: $request{OBJECT}");
238         &DisplayError("NewConnection, Can't parse object: $request{OBJECT}");
239      }
240
```

```
241     return 0;
242  }
243
244  # ============================================================================
245  # FUNCTION:  RequestHandler
246  #
247  # DESCRIPTION:
248  #    This routine is called to process requests and send the response data to
249  #    the browser. Page requests utilize subroutines to generate the necessary
250  #    response HTML. Run the program with debug level 1 to see the response
251  #    data on the console. Alternately, enable developer mode in the browser
252  #    (usually F12).
253  #
254  # CALLING SYNTAX:
255  #    $result = &RequestHandler($GetRequest, $Connect, \%Request);
256  #
257  # ARGUMENTS:
258  #    $GetRequest       Request data structure.
259  #    $Connect          Connection socket structure.
260  #    $Request          Pointer to request data hash.
261  #
262  # RETURNED VALUES:
263  #    0 = Success,  1 = Error.
264  #
265  # ACCESSED GLOBAL VARIABLES:
266  #    $main::DebugLevel
267  # ============================================================================
268  sub RequestHandler {
269     my($GetRequest, $Connect, $Request) = @_;
270     my($contentLength, $timestamp);
271     my($contentFile) = "$$Request{SHARE}/file_$$.html";
272     my(@resp) = ();
273
274     &DisplayDebug(1, "RequestHandler, page: '$$Request{PAGE}'");
275
276     # Send response content.
277     if ($$Request{TYPE} =~ m/html/) {
278
279        # Generate the HTML <head> section data.
280        push (@resp, qq(<!DOCTYPE html><html><head>));
281        push (@resp, qq(<title>D&amp;B Model Railroad</title>));
282
283        # Add javaScript.
284        &ScriptData(\@resp, $Request) if ($$Request{PAGE} =~ m/live/i);
285
286        # Add links to CSS and icon files.
287  #      push(@resp, qq(<link rel="stylesheet" href="DnB-large.css">));
288        push(@resp, qq(<link rel="stylesheet" media="screen and (min-height: 801px)") .
289                    qq( href="DnB-large.css">));
290        push(@resp, qq(<link rel="stylesheet" media="screen and (max-height: 800px)") .
291                    qq( href="DnB-small.css">));
292        push(@resp, qq(<link rel="apple-touch-icon" sizes="180x180" ) .
293                    qq(href="/apple-touch-icon.png">));
294        push(@resp, qq(<link rel="icon" type="image/png" sizes="32x32" ) .
295                    qq(href="/favicon-32x32.png">));
296        push(@resp, qq(<link rel="icon" type="image/png" sizes="16x16" ) .
297                    qq(href="/favicon-16x16.png">));
298        push(@resp, qq(<link rel="manifest" href="/site.webmanifest">));
299        push(@resp, qq(</head><body><div class="tab">));
300
```

```
301         # Generate the HTML <body> section data.
302         $$Request{BUILDER}->(\@resp, $Request) if (exists($$Request{BUILDER}));
303
304         # Complete the <html> page.
305         push(@resp,  qq(</div></body></html>));
306
307         # Tried to send the HTML data directly without creating a file but a
308         # number of transmission reliability issues and program crashes were
309         # encountered. Suspect this was due to socket data overload but could
310         # not identify the root cause.
311         if (&WriteFile($contentFile, \@resp, "")) {
312            $Connect->send_error(RC_NO_CONTENT, 'File: $$Request{PAGE}  ' .
313                                                'HTML file creation error.');
314            return 1;
315         }
316         else {
317            $contentLength = -s $contentFile;
318            if ($main::DebugLevel >= 1) {
319               foreach my $rec (@resp) {
320                  &DisplayDebug(1, "RequestHandler, resp: '$rec'");
321               }
322            }
323         }
324
325         # Send the response header to the browser.
326         $timestamp = strftime "%a, %d %b %Y %H:%M:%S GMT", gmtime;
327         $Connect->send_status_line(RC_OK,'OK', 'HTTP/1.1');
328         $Connect->send_header('Date', $timestamp);
329         $Connect->send_header('Server', 'D&B Model Railroad Rpi Webserver');
330         $Connect->send_header('Content-Type', $$Request{TYPE});
331         $Connect->send_header('Cache-Control', 'public');
332         $Connect->send_header('Accept-Ranges', 'bytes');
333         $Connect->send_header('Content-Length', $contentLength);
334         $Connect->send_crlf;
335
336         # Send the HTML data.
337         $Connect->send_file($contentFile);
338         $Connect->send_crlf;
339         &DisplayDebug(1, "RequestHandler, sent html: $contentFile");
340      }
341
342      # Send image data.
343      elsif ($$Request{TYPE} =~ m/image/ or $$Request{TYPE} =~ m/text/) {
344         $Connect->send_file_response( $$Request{PAGE} );
345         &DisplayDebug(1, "RequestHandler, sent file: $$Request{PAGE}");
346      }
347      return 0;
348   }
349
350   # ============================================================================
351   # FUNCTION:  ScriptData
352   #
353   # DESCRIPTION:
354   #    This routine is called to add a <script> section to the specified array.
355   #
356   # CALLING SYNTAX:
357   #    $result = &ScriptData($Array, $Request);
358   #
359   # ARGUMENTS:
360   #    $Array            Pointer to array for records.
```

```
361  #    $Request          Pointer to request data hash.
362  #
363  # RETURNED VALUES:
364  #    0 = Success.
365  #
366  # ACCESSED GLOBAL VARIABLES:
367  #    None.
368  # ============================================================================
369  sub ScriptData {
370     my($Array, $Request) = @_;
371
372     push(@$Array, qq(<script>));
373     if ($$Request{PAGE} =~ m/live/i) {
374
375        # The live page uses javaScript to auto-refresh the images that show the
376        # active track blocks. These transparent images overlay the page background
377        # image and color the active track blocks. The DnB.pl main loop updates the
378        # overlay images about once a second.
379        #
380        # A 'refresh(node)' function is launched for each overlay image when it is
381        # initially displayed by the 'window.onload = function()'. The initial
382        # display of the image is immediate because its URL does not contain a
383        # timestamp string. Subsequent URLs include a new timestamp so the browser
384        # is forced to re-GET the image from the webserver and not redisplay it
385        # from cache.
386        push(@$Array, qq(function refresh(node) { ));
387        push(@$Array, qq(  var timer = 2000;   // delay in msec ));
388        push(@$Array, '  (function startRefresh() { ');  # perl doesn't like single (
389        push(@$Array, qq(    var address; ));
390        push(@$Array, qq(    if(node.src.indexOf('?')>-1) ));
391        push(@$Array, qq(      address = node.src.split('?')[0]; ));
392        push(@$Array, qq(    else ));
393        push(@$Array, qq(      address = node.src; ));
394        push(@$Array, qq(      node.src = address+"?time="+new Date().getTime(); ));
395        push(@$Array, qq(      setTimeout(startRefresh,timer); ));
396        push(@$Array, '  })(); ');                       # perl doesn't like single )
397        push(@$Array, qq(} ));
398        push(@$Array, qq(window.onload = function() { ));
399        push(@$Array, qq(  var node = document.getElementById('y-Ovr'); ));
400        push(@$Array, qq(  refresh(node); ));
401        push(@$Array, qq(  var node = document.getElementById('m-Ovr'); ));
402        push(@$Array, qq(  refresh(node); ));
403        push(@$Array, qq(  var node = document.getElementById('h-Ovr'); ));
404        push(@$Array, qq(  refresh(node); ));
405
406        push(@$Array, qq(  var node = document.getElementById('L01-Ovr'); ));
407        push(@$Array, qq(  refresh(node); ));
408        push(@$Array, qq(  var node = document.getElementById('L02-Ovr'); ));
409        push(@$Array, qq(  refresh(node); ));
410        push(@$Array, qq(  var node = document.getElementById('L03-Ovr'); ));
411        push(@$Array, qq(  refresh(node); ));
412        push(@$Array, qq(  var node = document.getElementById('L04-Ovr'); ));
413        push(@$Array, qq(  refresh(node); ));
414        push(@$Array, qq(  var node = document.getElementById('L05-Ovr'); ));
415        push(@$Array, qq(  refresh(node); ));
416        push(@$Array, qq(  var node = document.getElementById('L06-Ovr'); ));
417        push(@$Array, qq(  refresh(node); ));
418        push(@$Array, qq(  var node = document.getElementById('L07-Ovr'); ));
419        push(@$Array, qq(  refresh(node); ));
420        push(@$Array, qq(  var node = document.getElementById('L08-Ovr'); ));
```

```
421         push(@$Array, qq(  refresh(node); ));
422         push(@$Array, qq(  var node = document.getElementById('L09-Ovr'); ));
423         push(@$Array, qq(  refresh(node); ));
424         push(@$Array, qq(  var node = document.getElementById('L10-Ovr'); ));
425         push(@$Array, qq(  refresh(node); ));
426         push(@$Array, qq(  var node = document.getElementById('L11-Ovr'); ));
427         push(@$Array, qq(  refresh(node); ));
428         push(@$Array, qq(  var node = document.getElementById('L12-Ovr'); ));
429         push(@$Array, qq(  refresh(node); ));
430
431         push(@$Array, qq(  var node = document.getElementById('GC01-Ovr'); ));
432         push(@$Array, qq(  refresh(node); ));
433         push(@$Array, qq(  var node = document.getElementById('GC02-Ovr'); ));
434         push(@$Array, qq(  refresh(node); ));
435         push(@$Array, qq(} ));
436      }
437      push(@$Array, qq(</script>));
438      return 0;
439   }
440
441   # ============================================================================
442   # FUNCTION:  TopPageData
443   #
444   # DESCRIPTION:
445   #    This routine is called to add top page data to the specified array. This
446   #    is the first page that is output when a user connects. It contains the
447   #    button controls for accessing the other data pages.
448   #
449   # CALLING SYNTAX:
450   #    $result = &TopPageData($Array, $Request);
451   #
452   # ARGUMENTS:
453   #    $Array              Pointer to array for records.
454   #    $Request            Pointer to request data hash.
455   #
456   # RETURNED VALUES:
457   #    0 = Success,  1 = Error.
458   #
459   # ACCESSED GLOBAL VARIABLES:
460   #    None.
461   # ============================================================================
462   sub TopPageData {
463      my($Array, $Request) = @_;
464
465      push(@$Array, qq(<div class="TopTitle"><h1>D&amp;B Model Railroad</h1>));
466      push(@$Array, qq(<div id="ImageContainer"><img class="TopImage" src=) .
467                    qq("loco-490x260RT.gif" alt="loco-490x260RT.gif"></div>));
468      push(@$Array, qq(<h4>Select from the following to see additional information.) .
469                    qq(  </h4></div>));
470      &GenNavBar($Array, $Request);
471      push(@$Array, qq(</div>));
472      push(@$Array, qq(<p class="copy">D&amp;B Model Railroad webserver ));
473      push(@$Array, qq(v1.4<br>Copyright &copy; 2020 Don Buczynski));
474      return 0;
475   }
476
477   # ============================================================================
478   # FUNCTION:  BlockPageData
479   #
480   # DESCRIPTION:
```

```perl
#    This routine is called to write the block detector related HTML and
#    data to the specified array. Block detector status is obtained from
#    the sensor.dat file. Refer to the DnB.pl %SensorBit hash.
#
#    sensor.dat       (generated by main loop)
#       Sensor: 32 sensor bits as a numeric value.
#          bit position: 1 = active, 0 = idle.
#
# CALLING SYNTAX:
#    $result = &BlockPageData($Array, $Request);
#
# ARGUMENTS:
#    $Array               Pointer to array for records.
#    $Request             Pointer to request data hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub BlockPageData {
   my($Array, $Request) = @_;
   my(@data, @bits);
   my($sensorBits) = 0;                    # No sensor bits set.
   my($bitMask) = 0x1;                     # Start at B01 bit position.
   my($tStr) = strftime "%r", localtime;
   my(%blockDesc) = ('B01' => '1-GPIOA0: Holdover track 1.',
                     'B02' => '1-GPIOA1: Holdover track 2.',
                     'B03' => '1-GPIOA2: Holdover / Midway transition track.',
                     'B04' => '1-GPIOA3: Midway siding track.',
                     'B05' => '1-GPIOA4: Midway mainline track.',
                     'B06' => '1-GPIOA5: Midway / Wye transition track.',
                     'B07' => '1-GPIOA6: Wye / Yard approach. Yard track 1.',
                     'B08' => '1-GPIOA7: Wye / Yard viaduct approach. Yard track 2.',
                     'B09' => '1-GPIOB0: Yard track 4.',
                     'B10' => '1-GPIOB1: Yard track 3.');

   # Start the HTML page.
   push(@$Array, qq(<div class="BlockTitle"><h1>D&amp;B Block Detector Status</h1>) .
                 qq(</div>));
   push(@$Array, qq(<div class="BlockBack">));
   push(@$Array, qq(<table align="center"><tr><td class="BlockSnap"><b>Snapshot ) .
                 qq(time:</b>  $tStr</td></tr><tr><td> </td></tr>) .
                 qq(</table>));
   push(@$Array, qq(<table class="Block"));
   push(@$Array, qq(<colgroup><col width=70px><col width=80px></colgroup>));
   push(@$Array, qq(<tr><th>Block</th><th>State</th><th>Description</th></tr>));

   # Get the sensor bit data from the file.
   unless (&ReadFile("$$Request{SHARE}/sensor.dat", \@data, "NoTrim")) {
      @bits = grep /Sensor:/, @data;
      if ($bits[0] =~ m/^Sensor:\s*(\d+)/) {
         $sensorBits = $1;
      }
   }
   &DisplayDebug(1, "BlockPageData, sensorBits: " . sprintf("%0.32b", $sensorBits));

   # Build the table records HTML.
   foreach my $block (sort keys(%blockDesc)) {
```

```
541         &DisplayDebug(1, "BlockPageData,    bitmask: " . sprintf("%0.32b", $bitMask) .
542                          "  $block   And result: " . ($sensorBits & $bitMask));
543         if (($sensorBits & $bitMask) != 0) {
544            push(@$Array, qq(<tr><td> $block</td><td class="Blu">Active</td>) .
545                          qq(<td>$blockDesc{$block}</td></tr>));
546         }
547         else {
548            push(@$Array, qq(<tr><td> $block</td><td class="blu">  ) .
549                          qq(idle</td><td>$blockDesc{$block}</td></tr>));
550         }
551         $bitMask = $bitMask << 1;  # Move mask to next block bit position.
552      }
553
554      # Finish the HTML page.
555      push(@$Array, qq(</table></div><br><br>));
556      &GenNavBar($Array, $Request);
557      push(@$Array, qq(</div>));
558      return 0;
559   }
560
561   # ============================================================================
562   # FUNCTION:  GradePageData
563   #
564   # DESCRIPTION:
565   #    This routine is called to write the grade crossing related HTML and data
566   #    to the specified array. Data is obtained from the grade.dat file. Refer
567   #    to the DnB.pl %GradeCrossingData hash.
568   #
569   # grade.dat        (generated by ProcessGradeCrossing)
570   #    GC01: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
571   #    GC02: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
572   #       <state> = 'idle', 'gateLower', 'approach', 'road', 'gateRaise' or 'depart'
573   #       <lamps> = 'on' or 'off'.
574   #       <gates> = 'Open', 'Closed', or '- none -'
575   #       <sensor> = 1 (active>) or 0 (idle).
576   #
577   # CALLING SYNTAX:
578   #    $result = &GradePageData($Array, $Request);
579   #
580   # ARGUMENTS:
581   #    $Array               Pointer to array for records.
582   #    $Request             Pointer to request data hash.
583   #
584   # RETURNED VALUES:
585   #    0 = Success,  1 = Error.
586   #
587   # ACCESSED GLOBAL VARIABLES:
588   #    None.
589   # ============================================================================
590   sub GradePageData {
591      my($Array, $Request) = @_;
592      my(@data, @gcs, $name, $state, $lamps, $gates, $aprW, $road, $aprE);
593      my($tStr) = strftime "%r", localtime;
594      my(%gcDesc) = ('GC01' => 'Lakeside shipping.',
595                     'GC02' => 'Columbia Feed Mill.');
596
597      # Start the HTML page.
598      push(@$Array, qq(<div class="GradeTitle"><h1>D&amp;B Grade Crossing Status) .
599                    qq(</h1></div>));
600      push(@$Array, qq(<p class="GradeSnap"><b>  Snapshot time:</b>  ) .
```

```
                qq( $tStr</p>));
    push(@$Array, qq(<br><table style="font-family:Sans-serif">));
    push(@$Array, qq(<tr><td width=300px>));

    # Get the grade crossing data from the file.
    unless (&ReadFile("$$Request{SHARE}/grade.dat", \@data, "NoTrim")) {
       @gcs = grep /GC\d\d: /, @data;

    # Build the table records HTML.
       foreach my $gc (sort @gcs) {
          chomp($gc);
          &DisplayDebug(1, "GradePageData, gc: '$gc'");

          #  GCxx: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
          if ($gc =~ m/(GC\d\d):\s*(.+?):(.+?):(.+?):(\d):(\d):(\d)/) {
             ($name, $state, $lamps, $gates, $aprW, $road, $aprE) = ($1, $2, $3, $4,
                                                                     $5, $6, $7);
             push(@$Array, qq(<div class="GradeData"><table><tr><td align="right">) .
                           qq(<b>Signal: </b></td><td>$name</td></tr>));
             push(@$Array, qq(<tr><td align="right"><b>Location: </b></td>) .
                           qq(<td>$gcDesc{$name}</td></tr>));

             push(@$Array, qq(<tr><td align="right"><b>State: </b></td>) .
                           qq(<td>) . ucfirst($state) . qq(</td></tr>));

             push(@$Array, qq(<tr><td align="right"><b>Lamps: </b></td>) .
                           qq(<td>) . ucfirst($lamps) . qq(</td></tr>));

             push(@$Array, qq(<tr><td align="right"><b>Gates: </b></td>));
             # ---
             if ($gates eq 'none') {
                push(@$Array, qq(<td class="blu">$gates</td></tr>));
             }
             else {
                push(@$Array, qq(<td>) . ucfirst($gates) . qq(</td></tr>));
             }
             # ---
             if ($aprW == 1) {
                push(@$Array, qq(<tr><td align="right"><b>AprW: </b></td>) .
                              qq(<td class="Blu">Active</td></tr>));
             }
             else {
                push(@$Array, qq(<tr><td align="right"><b>AprW: </b></td>) .
                              qq(<td class="blu">idle</td></tr>));
             }
             # ---
             if ($road == 1) {
                push(@$Array, qq(<tr><td align="right"><b>Road: </b></td>) .
                              qq(<td class="Blu">Active</td></tr>));
             }
             else {
                push(@$Array, qq(<tr><td align="right"><b>Road: </b></td>) .
                              qq(<td class="blu">idle</td></tr>));
             }
             # ---
             if ($aprE == 1) {
                push(@$Array, qq(<tr><td align="right"><b>AprE: </b></td>) .
                              qq(<td class="Blu">Active</td></tr>));
             }
             else {
```

```
661                push(@$Array, qq(<tr><td align="right"><b>AprE: </b></td>) .
662                             qq(<td class="blu">idle</td></tr>));
663             }
664             # ---
665             push(@$Array, qq(</table></div><br>));
666
667          }
668       }
669       # Next table data row.
670       @$Array[$#$Array] =~ s#<br>$#</td><td width=200px align="right">#;
671    }
672
673    # Finish the HTML page.
674    push(@$Array, qq(<div id="ImageContainer"><img class="GradeImage" src=) .
675                  qq("WigWag.gif" ALT="WigWag.gif"></div>));
676    push(@$Array, qq(</td></tr></table><br>));
677    &GenNavBar($Array, $Request);
678    push(@$Array, qq(</div>));
679    return 0;
680 }
681
682 # ============================================================================
683 # FUNCTION:  SensorPageData
684 #
685 # DESCRIPTION:
686 #    This routine is called to write the sensor related HTML and data to the
687 #    specified array. Sensor status is obtained from the sensor.dat file.
688 #    Refer to the DnB.pl %SensorBit hash.
689 #
690 #    sensor.dat       (generated by main loop)
691 #       Sensor: 32 sensor bits as a numeric value.
692 #          bit position: 1 = active, 0 = idle.
693 #
694 # CALLING SYNTAX:
695 #    $result = &SensorPageData($Array, $Request);
696 #
697 # ARGUMENTS:
698 #    $Array              Pointer to array for records.
699 #    $Request            Pointer to request data hash.
700 #
701 # RETURNED VALUES:
702 #    0 = Success,  1 = Error.
703 #
704 # ACCESSED GLOBAL VARIABLES:
705 #    None.
706 # ============================================================================
707 sub SensorPageData {
708    my($Array, $Request) = @_;
709    my(@data, @bits);
710    my($sensorBits) = 0;                    # No sensor bits set.
711    my($bitMask) = 0x10000;                 # Start at S01 bit position.
712    my($tStr) = strftime "%r", localtime;
713    my(%sensorDesc) = ('S01' => '2-GPIOA0: B03 to holdover entry.',
714                       'S02' => '2-GPIOA1: Holdover track 2 exit.',
715                       'S03' => '2-GPIOA2: Holdover track 1 exit.',
716                       'S04' => '2-GPIOA3: spare.',
717                       'S05' => '2-GPIOA4: B04 exit to B03 (Close T05).',
718                       'S06' => '2-GPIOA5: B05 exit to B06 (Open T06).',
719                       'S07' => '2-GPIOA6: B06 to Wye entry.',
720                       'S08' => '2-GPIOA7: B07 to Wye entry via yard track 1.',
```

```
                        'S09' => '2-GPIOB0: B08 to Wye entry via yard track 2.',
                        'S10' => '2-GPIOB1: Holdover track 1 exit yellow.',
                        'S11' => '2-GPIOB2: Holdover track 1 exit red.',
                        'S12' => '2-GPIOB3: Holdover track 2 exit yellow.',
                        'S13' => '2-GPIOB4: Holdover track 2 exit red.');

    # Start the HTML page.
    push(@$Array, qq(<div class="SensorTitle"><h1>D&amp;B Sensor Status</h1></div>));
    push(@$Array, qq(<div class="SensorBack">));
    push(@$Array, qq(<table align="center"><tr><td class="SensorSnap"><b>Snapshot ) .
                  qq(time:</b>  $tStr</td></tr><tr><td> </td></tr></table>));
    push(@$Array, qq(<table class="Sensor"));
    push(@$Array, qq(<colgroup><col width=70px><col width=80px></colgroup>));
    push(@$Array, qq(<tr><th>Sensor</th><th>State</th><th>Description</th></tr>));

    # Get the sensor bit data from the file.
    unless (&ReadFile("$$Request{SHARE}/sensor.dat", \@data, "NoTrim")) {
       @bits = grep /Sensor:/, @data;
       if ($bits[0] =~ m/^Sensor:\s*(\d+)/) {
          $sensorBits = $1;
       }
    }
    &DisplayDebug(1, "SensorPageData, sensorBits: " . sprintf("%0.32b", $sensorBits));

    # Build the table records HTML.
    foreach my $sensor (sort keys(%sensorDesc)) {
       &DisplayDebug(1, "SensorPageData,    bitmask: " . sprintf("%0.32b", $bitMask) .
                        "  $sensor");
       if ($sensorDesc{$sensor} =~ m/spare/i) {
          push(@$Array, qq(<tr class="grayout"><td> $sensor</td><td> ) .
                        qq( idle</td><td>$sensorDesc{$sensor}</td></tr>));
       }
       elsif (($sensorBits & $bitMask) != 0) {
          push(@$Array, qq(<tr><td> $sensor</td><td class="Blu">Active</td>) .
                        qq(<td>$sensorDesc{$sensor}</td></tr>));
       }
       else {
          push(@$Array, qq(<tr><td> $sensor</td><td class="blu">  ) .
                        qq(idle</td><td>$sensorDesc{$sensor}</td></tr>));
       }
       $bitMask = $bitMask << 1;  # Move mask to next sensor bit position.
    }

    # Finish the HTML page.
    push(@$Array, qq(</table></div><br><br>));
    &GenNavBar($Array, $Request);
    push(@$Array, qq(</div>));
    return 0;
}

# ============================================================================
# FUNCTION:  SignalPageData
#
# DESCRIPTION:
#    This routine is called to write the signal related HTML and data to the
#    specified array. Signal status is obtained from the sensor.dat file.
#    Refer to the DnB.pl %SignalData hash.
#
#    sensor.dat       (generated by main loop)
#       Signal: L01=x,L02=x, ... L12=x
```

```
#          x = 'Off', 'Grn', 'Yel', or 'Red'.
#
# CALLING SYNTAX:
#    $result = &SignalPageData($Array, $Request);
#
# ARGUMENTS:
#    $Array               Pointer to array for records.
#    $Request             Pointer to request data hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub SignalPageData {
   my($Array, $Request) = @_;
   my(@data, @signals, $sigList, $color);
   my($tStr) = strftime "%r", localtime;
   my(%signalDesc) = ('L01' => '00,01: Holdover to B03 upgrade.',
                      'L02' => '02,03: B04 / B05 to B03 downgrade.',
                      'L03' => '04,05: B03 to B04 upgrade.',
                      'L04' => '06,07: B06 to B04 downgrade.',
                      'L05' => '08,09: B03 to B05 upgrade.',
                      'L06' => '10,11: B06 to B05 downgrade.',
                      'L07' => '12,13: B04 / B05 to B06 upgrade.',
                      'L08' => '14,15: B07 / B08 to B06 downgrade. (sem)',
                      'L09' => '16,17: B06 to B07 upgrade.',
                      'L10' => '18,19: B09 / B10 to B07 downgrade.',
                      'L11' => '20,21: B06 to B08 upgrade.',
                      'L12' => '22,23: B09 / B10 to B08 downgrade.');

   # Start the HTML page.
   push(@$Array, qq(<div class="SignalTitle"><h1>D&amp;B Signal Status</h1></div>));
   push(@$Array, qq(<div class="SignalBack">));
   push(@$Array, qq(<table align="center"><tr><td class="SignalSnap"><b>Snapshot) .
                 qq( time:</b>  $tStr</td></tr><tr><td> </td></tr>) .
                 qq(</table>));
   push(@$Array, qq(<table class="Signal"));
   push(@$Array, qq(<colgroup><col width=70px><col width=70px></colgroup>));
   push(@$Array, qq(<tr><th>Signal</th><th>State</th><th>Description</th></tr>));

   # Get the signal data from the file.
   unless (&ReadFile("$$Request{SHARE}/sensor.dat", \@data, "NoTrim")) {
      @signals = grep /Signal:/, @data;
      if ($signals[0] =~ m/^Signal:\s*(.+)/) {
         $sigList = $1;
      }
   }

   # Build the table records HTML.
   foreach my $signal (sort keys(%signalDesc)) {
      if ($sigList =~ m/$signal=(.{3})/) {
         $color = $1;
      }
      else {
         $color = '===';    # If we don't match.
      }

      if ($color =~ m/Red/i) {
```

```
841            push(@$Array, qq(<tr><td> $signal</td><td class="red"> Red</td>) .
842                          qq(</td><td>$signalDesc{$signal}</td></tr>));
843         }
844         elsif ($color =~ m/Yel/i) {
845            push(@$Array, qq(<tr><td> $signal</td><td class="yel"> Yel</td>) .
846                          qq(</td><td>$signalDesc{$signal}</td></tr>));
847         }
848         elsif ($color =~ m/Grn/i) {
849            push(@$Array, qq(<tr><td> $signal</td><td class="grn"> Grn</td>) .
850                          qq(</td><td>$signalDesc{$signal}</td></tr>));
851         }
852         elsif ($color =~ m/===/i) {
853            push(@$Array, qq(<tr><td> $signal</td><td class="blu"> $color) .
854                          qq(</td></td><td>$signalDesc{$signal}</td></tr>));
855         }
856         else {
857            push(@$Array, qq(<tr><td> $signal</td><td class="blu"> Off</td>) .
858                          qq(</td><td>$signalDesc{$signal}</td></tr>));
859         }
860      }
861
862      # Finish the HTML page.
863      push(@$Array, qq(</table></div><br><br>));
864      &GenNavBar($Array, $Request);
865      push(@$Array, qq(</div>));
866      return 0;
867   }
868
869   # ============================================================================
870   # FUNCTION:  TurnoutPageData
871   #
872   # DESCRIPTION:
873   #    This routine is called to write the turnout related HTML and data to the
874   #    specified array. Turnout status is obtained from the sensor.dat file.
875   #    Refer to the DnB.pl %TurnoutData hash.
876   #
877   #    sensor.dat       (generated by main loop)
878   #       T01=<value1>:<value2>: ... <value8>
879   #       T02=<value1>:<value2>: ... <value8>
880   #       ...
881   #
882   #       value order = Pos, Rate, Open, Middle, Close, MinPos, MaxPos, Id
883   #
884   # CALLING SYNTAX:
885   #    $result = &TurnoutPageData($Array, $Request);
886   #
887   # ARGUMENTS:
888   #    $Array               Pointer to array for records.
889   #    $Request             Pointer to request data hash.
890   #
891   # RETURNED VALUES:
892   #    0 = Success,  1 = Error.
893   #
894   # ACCESSED GLOBAL VARIABLES:
895   #    None.
896   # ============================================================================
897   sub TurnoutPageData {
898      my($Array, $Request) = @_;
899      my(@data, @tData, @tParm, $html, $x, $pos, $spare);
900      my($tStr) = strftime "%r", localtime;
```

```
901
902     # Start the HTML page.
903     push(@$Array, qq(<div class="TurnoutTitle"><h1>D&amp;B Turnout Status</h1></div>));
904     push(@$Array, qq(<div class="TurnoutBack">));
905     push(@$Array, qq(<table align="center"><tr><td class="TurnoutSnap"><b>Snapshot ) .
906                   qq(time:</b>  $tStr<br><br></td></tr></table>));
907     push(@$Array, qq(<table class="Turnout"));
908     push(@$Array, qq(<colgroup><col width=45px><col width=45px><col width=45px>));
909     push(@$Array, qq(<col width=45px><col width=45px><col width=45px><col width=45px>));
910     push(@$Array, qq(<col width=45px><col width=230px></colgroup>));
911
912     push(@$Array, qq(<tr><th>Id</th><th>Pos</th><th>Rate</th><th>Open</th>));
913     push(@$Array, qq(<th>Midl</th><th>Close</th><th>MinP</th><th>MaxP</th>));
914     push(@$Array, qq(<th>Description</th></tr>));
915
916     # Get the turnout data from the file.
917     unless (&ReadFile("$$Request{SHARE}/sensor.dat", \@data, "NoTrim")) {
918
919     # Build the table records HTML.
920        foreach my $tNmbr (1..32) {
921           $tNmbr = "0${tNmbr}" if (length($tNmbr) == 1);
922           $tNmbr = join('', 'T', $tNmbr);
923           @tData = grep /^$tNmbr=/, @data;
924           chomp($tData[0]);
925           &DisplayDebug(1, "TurnoutPageData, tNmbr: $tNmbr   tData[0]: '$tData[0]'");
926           if ($tData[0] =~ m/^$tNmbr=(.+)/) {
927              @tParm = split(':', $1);
928              if ($tParm[$#tParm] =~ m/spare/i) {                  # Grayout spares
929                 $html = qq(<tr class="grayout"><td>$tNmbr</td>);
930                 $spare = 1;
931              }
932              else {
933                 $html = qq(<tr><td>$tNmbr</td>);
934                 $spare = 0;
935              }
936              for ($x = 0; $x <= $#tParm; $x++) {
937                 $pos = $tParm[$x] if ($x == 0);  # Copy pos for open/close color check.
938
939                 # Account for temperature adjusted pos value.
940                 if ($x == 2 and $spare == 0 and $pos > ($tParm[$x]-10) and
941                     $pos < ($tParm[$x]+10)) {
942                    $html = join('', $html, qq(<td class="red">$tParm[$x]</td>));
943                 }
944                 elsif ($x == 3 and $spare == 0 and $pos > ($tParm[$x]-10) and
945                        $pos < ($tParm[$x]+10)) {
946                    $html = join('', $html, qq(<td class="yel">$tParm[$x]</td>));
947                 }
948                 elsif ($x == 4 and $spare == 0 and $pos > ($tParm[$x]-10) and
949                        $pos < ($tParm[$x]+10)) {
950                    $html = join('', $html, qq(<td class="grn">$tParm[$x]</td>));
951                 }
952                 else {
953                    $html = join('', $html, qq(<td>$tParm[$x]</td>));
954                 }
955              }
956              $html = join('', $html, '</tr>');
957              push(@$Array, $html);
958           }
959        }
960     }
```

```
961
962     # Finish the HTML page.
963     push(@$Array, qq(</table></div><br>));
964     &GenNavBar($Array, $Request);
965     push(@$Array, qq(</div>));
966     return 0;
967   }
968
969   # ============================================================================
970   # FUNCTION:  LivePageData
971   #
972   # DESCRIPTION:
973   #    This routine is called to add live page data to the specified array. This
974   #    page displays layout information in near real time in the browser. Java
975   #    script is added to the HTML page header to instruct the browser to refresh
976   #    the overlay images about every two seconds.
977   #
978   #    The overlay images are specified as .dat files. The NewConnection code
979   #    substitutes the current main line specified overlay file when processing
980   #    the request.
981   #
982   # CALLING SYNTAX:
983   #    $result = &LivePageData($Array, $Request);
984   #
985   # ARGUMENTS:
986   #    $Array               Pointer to array for records.
987   #    $Request             Pointer to request data hash.
988   #
989   # RETURNED VALUES:
990   #    0 = Success,  1 = Error.
991   #
992   # ACCESSED GLOBAL VARIABLES:
993   #    None.
994   # ============================================================================
995   sub LivePageData {
996     my($Array, $Request) = @_;
997
998     push(@$Array, qq(<div class="LiveTitle"><h1>D&amp;B Model Railroad Live</h1>) .
999                   qq(</div>));
1000    push(@$Array, qq(<img class="backImage" src="TrackPlan.png" ) .
1001                  qq(alt="TrackPlan.png"></div>));
1002    push(@$Array, qq(<img id="y-Ovr" class="yardImage" src="y-overlay.dat" ) .
1003                  qq(alt="y-overlay.dat">));
1004    push(@$Array, qq(<img id="m-Ovr" class="midwayImage" src="m-overlay.dat" ) .
1005                  qq(alt="m-overlay.dat">));
1006
1007    push(@$Array, qq(<img id="h-Ovr" class="holdoverImage" src="h-overlay.dat" ) .
1008                  qq(alt="h-overlay.dat">));
1009    push(@$Array, qq(<img id="L01-Ovr" class="L01color" src="L01-overlay.dat" ) .
1010                  qq(alt="L01-overlay.dat">));
1011    push(@$Array, qq(<img id="L02-Ovr" class="L02color" src="L02-overlay.dat" ) .
1012                  qq(alt="L02-overlay.dat">));
1013    push(@$Array, qq(<img id="L03-Ovr" class="L03color" src="L03-overlay.dat" ) .
1014                  qq(alt="L03-overlay.dat">));
1015    push(@$Array, qq(<img id="L04-Ovr" class="L04color" src="L04-overlay.dat" ) .
1016                  qq(alt="L04-overlay.dat">));
1017    push(@$Array, qq(<img id="L05-Ovr" class="L05color" src="L05-overlay.dat" ) .
1018                  qq(alt="L05-overlay.dat">));
1019    push(@$Array, qq(<img id="L06-Ovr" class="L06color" src="L06-overlay.dat" ) .
1020                  qq(alt="L06-overlay.dat">));
```

```
1021     push(@$Array, qq(<img id="L07-Ovr" class="L07color" src="L07-overlay.dat" ) .
1022                   qq(alt="L07-overlay.dat">));
1023     push(@$Array, qq(<img id="L08-Ovr" class="L08color" src="L08-overlay.dat" ) .
1024                   qq(alt="L08-overlay.dat">));
1025     push(@$Array, qq(<img id="L09-Ovr" class="L09color" src="L09-overlay.dat" ) .
1026                   qq(alt="L09-overlay.dat">));
1027     push(@$Array, qq(<img id="L10-Ovr" class="L10color" src="L10-overlay.dat" ) .
1028                   qq(alt="L10-overlay.dat">));
1029     push(@$Array, qq(<img id="L11-Ovr" class="L11color" src="L11-overlay.dat" ) .
1030                   qq(alt="L11-overlay.dat">));
1031     push(@$Array, qq(<img id="L12-Ovr" class="L12color" src="L12-overlay.dat" ) .
1032                   qq(alt="L12-overlay.dat">));
1033
1034     push(@$Array, qq(<img id="GC01-Ovr" class="GC01Image" src="GC01-overlay.dat" ) .
1035                   qq(alt="GC01-overlay.dat">));
1036     push(@$Array, qq(<img id="GC02-Ovr" class="GC02Image" src="GC02-overlay.dat" ) .
1037                   qq(alt="GC02-overlay.dat">));
1038     &GenNavBar($Array, $Request);
1039     push(@$Array, qq(<div class="LiveEndPad"> </div>));
1040     return 0;
1041  }
1042
1043  # ============================================================================
1044  # FUNCTION:  GenNavBar
1045  #
1046  # DESCRIPTION:
1047  #    This routine is called to add the navigation button HTML to the specified
1048  #    array. The 'Top' page gets the page link buttons. All other pages have the
1049  #    'Home' and 'Refresh' buttons added to the page link buttons.
1050  #
1051  #
1052  # CALLING SYNTAX:
1053  #    $result = &GenNavBar($Array, $Request);
1054  #
1055  # ARGUMENTS:
1056  #    $Array              Pointer to array for records.
1057  #    $Request            Pointer to request data hash.
1058  #
1059  # RETURNED VALUES:
1060  #    0 = Success,  1 = Error.
1061  #
1062  # ACCESSED GLOBAL VARIABLES:
1063  #    None.
1064  # ============================================================================
1065  sub GenNavBar {
1066     my($Array, $Request) = @_;
1067
1068     if ($$Request{PAGE} =~ m/live/i) {
1069        push(@$Array, qq(<div class="navGroupLive"><table class="navTable"><tr>));
1070     }
1071     else {
1072        push(@$Array, qq(<div class="navGroup"><table class="navTable"><tr>));
1073     }
1074
1075     push(@$Array, qq(<td><button><a href="/block" class="navButton">Block</a>) .
1076                   qq(</button></td>));
1077     push(@$Array, qq(<td><button><a href="/grade" class="navButton">Grade</a>) .
1078                   qq(</button></td>));
1079     push(@$Array, qq(<td><button><a href="/sensor" class="navButton">Sensor</a>) .
1080                   qq(</button></td>));
```

```
1081     push(@$Array, qq(<td><button><a href="/signal" class="navButton">Signal</a>) .
1082                   qq(</button></td>));
1083     push(@$Array, qq(<td><button><a href="/turnout" class="navButton">Turnout</a>) .
1084                   qq(</button></td>));
1085
1086     if ($$Request{PAGE} =~ m/top/i) {
1087        push(@$Array, qq(</tr><tr><td> </td><td> </td>));
1088        push(@$Array, qq(<td><button><a href="/live" class="navButton">Live</a>) .
1089                      qq(</button></td>));
1090        push(@$Array, qq(<td> </td><td> </td>));
1091     }
1092     elsif ($$Request{PAGE} =~ m/live/i) {
1093        push(@$Array, qq(</tr><tr><td> </td><td> </td>));
1094        push(@$Array, qq(<td><button><a href="/top" class="navButton">Home</a>) .
1095                      qq(</button></td>));
1096        push(@$Array, qq(<td> </td><td> </td>));
1097     }
1098     else {
1099        push(@$Array, qq(</tr><tr><td><button><a href="/top" class="navButton">) .
1100                      qq(Home</a></button></td>));
1101        push(@$Array, qq(<td> </td>));
1102        push(@$Array, qq(<td><button><a href="/live" class="navButton">Live</a>) .
1103                      qq(</button></td>));
1104        push(@$Array, qq(<td> </td>));
1105        push(@$Array, qq(<td><button><a href="/$$Request{PAGE}" class="navButton">) .
1106                      qq(Refresh</a></button></td>));
1107     }
1108     push(@$Array, qq(</tr></table></div>));
1109     return 0;
1110  }
1111
1112  # ================================================================================
1113  # FUNCTION:  ExtractVariables
1114  #
1115  # DESCRIPTION:
1116  #    This routine is called to parse the specified string for URL name/value
1117  #    pairs and return them in the specified hash. Name/value pairs, if any,
1118  #    begin after the first '?' character. Name and value are seperated by the
1119  #    '=' character. Multiple name/value pairs are '&' seperated.
1120  #
1121  # CALLING SYNTAX:
1122  #    $result = &ExtractVariables($Url, \%Variables);
1123  #
1124  # ARGUMENTS:
1125  #    $Url                String to process.
1126  #    $Variables          Pointer to hash.
1127  #
1128  # RETURNED VALUES:
1129  #    0 = Success,  1 = Error.
1130  #
1131  # ACCESSED GLOBAL VARIABLES:
1132  #    None.
1133  # ================================================================================
1134  sub ExtractVariables {
1135     my($Url, $Variables) = @_;
1136     my($data, @pairs, $name, $value);
1137
1138     %$Variables = ();
1139     if ($Url =~ m/^(.+?)\?(.+)/) {
1140        $data = $2;
```

```
1141        if ($data ne '') {
1142           @pairs = split('&', $data);
1143           foreach my $pair (@pairs) {
1144              if ($pair =~ m/^(.+?)=(.+)$/) {
1145                 $name = $1;
1146                 $value = $2;
1147                 $name =~ s/%(..)/chr(hex($1))/eg;
1148                 $value =~ s/%(..)/chr(hex($1))/eg;
1149                 $$Variables{$name} = $value;
1150              }
1151           }
1152        }
1153     }
1154     return 0;
1155  }
1156
1157  return 1;
1158
```