```perl
#!/usr/bin/perl
# ============================================================================
# FILE: DnB_Webserver.pm                                          10-12-2020
#
# 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 web 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 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  #    Depending on the browser version, the raw IP:Port might be needed for the
62  #    initial connection. 'sudo ifconfig' will display the network interfaces
63  #    on the Rpi.
64  #
65  # CALLING SYNTAX:   (using Super::Fork)
66  #    $pid = fork {sub => \&Webserver, args => [ $WebRoot, $ListenPort,
67  #                                               $WebDataDir ] };
68  #
69  # ARGUMENTS:
70  #    $WebRoot            Webserver document root directory.
71  #    $ListenPort         Port to listen to for connections.
72  #    $WebDataDir         Directory for dynamic data content.
73  #
74  # RETURNED VALUES:
75  #    non-zero pid = Success,  undef($pid) = Error.
76  #
77  # ACCESSED GLOBAL VARIABLES:
78  #    $main::ChildName
79  # ============================================================================
80  sub Webserver {
81     my($WebRoot, $ListenPort, $WebDataDir) = @_;
82     my($result, $host, $url, $ipp, $daemon, $connect, $getRequest, $method);
83
84     # Assume we just booted. Give time for WIFI to setup.
85     sleep 10;
86
87     # Remove any previous dynamic HTML files. The other data files are handled
88     # by the main loop code.
89     unlink glob("$WebDataDir/file_*.html");
90
91     # Define some working variables and display webserver connection point.
92     $host = `/bin/hostname`;
93     if (($? >> 8) != 0) {
94        &DisplayError("Failed to get hostname. Webserver not started.");
95        exit(1);
96     }
97     chomp($host);
98     $url = join(':', $host, $ListenPort);
99
100    # Determine raw ip:port for alternate browser connection point.
101    $host = `/bin/hostname -I`;
102    if ($host =~ m/^(\d{1,3}\.\d{1,3}\.\d{1,3}\.\d{1,3})\s/) {
103       $host = $1;
104    }
105    $ipp = join(':', &Trim($host), $ListenPort);
106    $main::ChildName = "Webserver Process $ipp";
107
108    # Establish connection point and start the webserver.
109    unless($daemon = HTTP::Daemon->new(LocalPort => $ListenPort, ReuseAddr => 1,
110                     Family => AF_INET, Type => SOCK_STREAM,  Listen => 5)) {
111       &DisplayError("Webserver failed to start: $!");
112    }
113    else {
114       &DisplayMessage("Webserver started.  Client connection url: $url or $ipp");
115
116       # Process client connections.
117       while ($connect = $daemon->accept) {
118          while ($getRequest = $connect->get_request) {
119             $method = $getRequest->method;
120             if ($method eq 'GET') {
```

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

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

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

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

```
#
#    A 'refresh(node)' function is launched for each overlay image when it is
#    initially displayed by the 'window.onload = function()'. The initial
#    display of the image is immediate because its URL does not contain a
#    timestamp string. Subsequent URLs include a new timestamp so the browser
#    is forced to re-GET the image from the webserver and not redisplay it
#    from cache.
#
# CALLING SYNTAX:
#    $result = &ScriptData($Array, $Request);
#
# ARGUMENTS:
#    $Array            Pointer to array for records.
#    $Request          Pointer to request data hash.
#
# RETURNED VALUES:
#    0 = Success.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ScriptData {
   my($Array, $Request) = @_;

   push(@$Array, qq(<script>));
   if ($$Request{PAGE} =~ m/main/i) {      # Add javascript for mainline live page.
      push(@$Array, qq(function refresh(node) { ));
      push(@$Array, qq(  var timer = 2000;   // delay in msec ));
      push(@$Array, '  (function startRefresh() { ');  # perl doesn't like single (
      push(@$Array, qq(    var address; ));
      push(@$Array, qq(    if(node.src.indexOf('?')>-1) ));
      push(@$Array, qq(      address = node.src.split('?')[0]; ));
      push(@$Array, qq(    else ));
      push(@$Array, qq(      address = node.src; ));
      push(@$Array, qq(      node.src = address+"?time="+new Date().getTime(); ));
      push(@$Array, qq(      setTimeout(startRefresh,timer); ));
      push(@$Array, '  })(); ');                        # perl doesn't like single )
      push(@$Array, qq(} ));
      push(@$Array, qq(window.onload = function() { ));
      push(@$Array, qq(  var node = document.getElementById('y-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('m-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('h-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));

      push(@$Array, qq(  var node = document.getElementById('L01-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('L02-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('L03-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('L04-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('L05-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('L06-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
      push(@$Array, qq(  var node = document.getElementById('L07-Ovr'); ));
      push(@$Array, qq(  refresh(node); ));
```

```
421         push(@$Array, qq(  var node = document.getElementById('L08-Ovr'); ));
422         push(@$Array, qq(  refresh(node); ));
423         push(@$Array, qq(  var node = document.getElementById('L09-Ovr'); ));
424         push(@$Array, qq(  refresh(node); ));
425         push(@$Array, qq(  var node = document.getElementById('L10-Ovr'); ));
426         push(@$Array, qq(  refresh(node); ));
427         push(@$Array, qq(  var node = document.getElementById('L11-Ovr'); ));
428         push(@$Array, qq(  refresh(node); ));
429         push(@$Array, qq(  var node = document.getElementById('L12-Ovr'); ));
430         push(@$Array, qq(  refresh(node); ));
431
432         push(@$Array, qq(  var node = document.getElementById('GC01-Ovr'); ));
433         push(@$Array, qq(  refresh(node); ));
434         push(@$Array, qq(  var node = document.getElementById('GC02-Ovr'); ));
435         push(@$Array, qq(  refresh(node); ));
436         push(@$Array, qq(} ));
437     }
438
439     elsif ($$Request{PAGE} =~ m/yard/i) {   # Add javascript for yard live page.
440         push(@$Array, qq(function refresh(node) { ));
441         push(@$Array, qq(  var timer = 2000;   // delay in msec ));
442         push(@$Array, '  (function startRefresh() { ');  # perl doesn't like single (
443         push(@$Array, qq(    var address; ));
444         push(@$Array, qq(    if(node.src.indexOf('?')>-1) ));
445         push(@$Array, qq(      address = node.src.split('?')[0]; ));
446         push(@$Array, qq(    else ));
447         push(@$Array, qq(      address = node.src; ));
448         push(@$Array, qq(      node.src = address+"?time="+new Date().getTime(); ));
449         push(@$Array, qq(      setTimeout(startRefresh,timer); ));
450         push(@$Array, '  })(); ');                        # perl doesn't like single )
451         push(@$Array, qq(} ));
452         push(@$Array, qq(window.onload = function() { ));
453         push(@$Array, qq(  var node = document.getElementById('S1-Ovr'); ));
454         push(@$Array, qq(  refresh(node); ));
455         push(@$Array, qq(  var node = document.getElementById('S2-Ovr'); ));
456         push(@$Array, qq(  refresh(node); ));
457         push(@$Array, qq(  var node = document.getElementById('S3-Ovr'); ));
458         push(@$Array, qq(  refresh(node); ));
459         push(@$Array, qq(  var node = document.getElementById('S4-Ovr'); ));
460         push(@$Array, qq(  refresh(node); ));
461         push(@$Array, qq(  var node = document.getElementById('S5-Ovr'); ));
462         push(@$Array, qq(  refresh(node); ));
463         push(@$Array, qq(  var node = document.getElementById('S6-Ovr'); ));
464         push(@$Array, qq(  refresh(node); ));
465         push(@$Array, qq(} ));
466     }
467
468     # The following pages have javaScript added to periodically auto-refresh
469     # their entire content at 5 second intervals. Since the page data is minimal,
470     # this technique results in manageable overhead.
471     elsif ($$Request{PAGE} =~ m/block/i or $$Request{PAGE} =~ m/grade/i or
472            $$Request{PAGE} =~ m/sensor/i or $$Request{PAGE} =~ m/signal/i or
473            $$Request{PAGE} =~ m/turnout/i) {
474         push(@$Array, qq(window.onload = setupRefresh;));
475         push(@$Array, qq(function setupRefresh() {));
476         push(@$Array, qq(  setTimeout("refreshPage();", 5000); // milliseconds));
477         push(@$Array, qq(}));
478         push(@$Array, qq(function refreshPage() {));
479         push(@$Array, qq(  window.location = location.href;));
480         push(@$Array, qq(}));
```

```
481     }
482     push(@$Array, qq(</script>));
483     return 0;
484  }
485
486  # ============================================================================
487  # FUNCTION:  TopPageData
488  #
489  # DESCRIPTION:
490  #    This routine is called to add top page data to the specified array. This
491  #    is the first page that is output when a user connects. It contains the
492  #    button controls for accessing the other data pages.
493  #
494  # CALLING SYNTAX:
495  #    $result = &TopPageData($Array, $Request);
496  #
497  # ARGUMENTS:
498  #    $Array              Pointer to array for records.
499  #    $Request            Pointer to request data hash.
500  #
501  # RETURNED VALUES:
502  #    0 = Success,  1 = Error.
503  #
504  # ACCESSED GLOBAL VARIABLES:
505  #    None.
506  # ============================================================================
507  sub TopPageData {
508     my($Array, $Request) = @_;
509
510     push(@$Array, qq(<div class="TopTitle"><h1>D&amp;B Model Railroad</h1>));
511     push(@$Array, qq(<div id="ImageContainer"><img class="TopImage" src=) .
512                   qq("loco-490x260RT.gif" alt="loco-490x260RT.gif"></div>));
513     push(@$Array, qq(<h4>Select from the following to see additional information.) .
514                   qq(  </h4></div>));
515     &GenNavBar($Array, $Request);
516     push(@$Array, qq(</div>));
517     push(@$Array, qq(<p class="copy">D&amp;B Model Railroad webserver ));
518     push(@$Array, qq(v1.5<br>Copyright &copy; 2020 Don Buczynski));
519     return 0;
520  }
521
522  # ============================================================================
523  # FUNCTION:  BlockPageData
524  #
525  # DESCRIPTION:
526  #    This routine is called to write the block detector related HTML and
527  #    data to the specified array. Block detector status is obtained from
528  #    the sensor.dat file. Refer to the DnB.pl %SensorBit hash.
529  #
530  #    sensor.dat       (generated by main loop)
531  #       Sensor: 32 sensor bits as a numeric value.
532  #          bit position: 1 = active, 0 = idle.
533  #
534  # CALLING SYNTAX:
535  #    $result = &BlockPageData($Array, $Request);
536  #
537  # ARGUMENTS:
538  #    $Array              Pointer to array for records.
539  #    $Request            Pointer to request data hash.
540  #
```

```perl
# 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(@$Array, qq(<div class="BlockTitle"><h1>D&amp;amp;B Block Detector Status</h1>) .
                 qq(</div>));
   push(@$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)) {
      &DisplayDebug(1, "BlockPageData,    bitmask: " . sprintf("%0.32b", $bitMask) .
                       "  $block   And result: " . ($sensorBits & $bitMask));
      if (($sensorBits & $bitMask) != 0) {
         push(@$Array, qq(<tr><td> $block</td><td class="Blu">Active</td>) .
                       qq(<td>$blockDesc{$block}</td></tr>));
      }
      else {
         push(@$Array, qq(<tr><td> $block</td><td class="blu">  ) .
                       qq(idle</td><td>$blockDesc{$block}</td></tr>));
      }
      $bitMask = $bitMask << 1;  # Move mask to next block bit position.
   }

   # Finish the HTML page.
   push(@$Array, qq(</table></div><br><br>));
```

```
601     &GenNavBar($Array, $Request);
602     push(@$Array, qq(</div>));
603     return 0;
604  }
605
606  # ============================================================================
607  # FUNCTION:  GradePageData
608  #
609  # DESCRIPTION:
610  #    This routine is called to write the grade crossing related HTML and data
611  #    to the specified array. Data is obtained from the grade.dat file. Refer
612  #    to the DnB.pl %GradeCrossingData hash.
613  #
614  # grade.dat        (generated by ProcessGradeCrossing)
615  #    GC01: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
616  #    GC02: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
617  #       <state> = 'idle', 'gateLower', 'approach', 'road', 'gateRaise' or 'depart'
618  #       <lamps> = 'on' or 'off'.
619  #       <gates> = 'Open', 'Closed', or '- none -'
620  #       <sensor> = 1 (active>) or 0 (idle).
621  #
622  # CALLING SYNTAX:
623  #    $result = &GradePageData($Array, $Request);
624  #
625  # ARGUMENTS:
626  #    $Array               Pointer to array for records.
627  #    $Request             Pointer to request data hash.
628  #
629  # RETURNED VALUES:
630  #    0 = Success,  1 = Error.
631  #
632  # ACCESSED GLOBAL VARIABLES:
633  #    None.
634  # ============================================================================
635  sub GradePageData {
636     my($Array, $Request) = @_;
637     my(@data, @gcs, $name, $state, $lamps, $gates, $aprW, $road, $aprE);
638     my($tStr) = strftime "%r", localtime;
639     my(%gcDesc) = ('GC01' => 'Lakeside shipping.',
640                    'GC02' => 'Columbia Feed Mill.');
641
642     # Start the HTML page.
643     push(@$Array, qq(<div class="GradeTitle"><h1>D&amp;B Grade Crossing Status) .
644                   qq(</h1></div>));
645     push(@$Array, qq(<p class="GradeSnap"><b>  Snapshot time:</b>  ) .
646                   qq( $tStr</p>));
647     push(@$Array, qq(<br><table style="font-family:Sans-serif">));
648     push(@$Array, qq(<tr><td width=300px>));
649
650     # Get the grade crossing data from the file.
651     unless (&ReadFile("$$Request{SHARE}/grade.dat", \@data, "NoTrim")) {
652        @gcs = grep /GC\d\d: /, @data;
653
654     # Build the table records HTML.
655        foreach my $gc (sort @gcs) {
656           chomp($gc);
657           &DisplayDebug(1, "GradePageData, gc: '$gc'");
658
659           #  GCxx: <state>:<lamps>:<gates>:<aprW>:<road>:<aprE>
660           if ($gc =~ m/(GC\d\d):\s*(.+?):(.+?):(.+?):(\d):(\d):(\d)/) {
```

```
661             ($name, $state, $lamps, $gates, $aprW, $road, $aprE) = ($1, $2, $3, $4,
662                                                                    $5, $6, $7);
663             push(@$Array, qq(<div class="GradeData"><table><tr><td align="right">) .
664                           qq(<b>Signal: </b></td><td>$name</td></tr>));
665             push(@$Array, qq(<tr><td align="right"><b>Location: </b></td>) .
666                           qq(<td>$gcDesc{$name}</td></tr>));
667
668             push(@$Array, qq(<tr><td align="right"><b>State: </b></td>) .
669                           qq(<td>) . ucfirst($state) . qq(</td></tr>));
670
671             push(@$Array, qq(<tr><td align="right"><b>Lamps: </b></td>) .
672                           qq(<td>) . ucfirst($lamps) . qq(</td></tr>));
673
674             push(@$Array, qq(<tr><td align="right"><b>Gates: </b></td>));
675             # ---
676             if ($gates eq 'none') {
677                push(@$Array, qq(<td class="blu">$gates</td></tr>));
678             }
679             else {
680                push(@$Array, qq(<td>) . ucfirst($gates) . qq(</td></tr>));
681             }
682             # ---
683             if ($aprW == 1) {
684                push(@$Array, qq(<tr><td align="right"><b>AprW: </b></td>) .
685                              qq(<td class="Blu">Active</td></tr>));
686             }
687             else {
688                push(@$Array, qq(<tr><td align="right"><b>AprW: </b></td>) .
689                              qq(<td class="blu">idle</td></tr>));
690             }
691             # ---
692             if ($road == 1) {
693                push(@$Array, qq(<tr><td align="right"><b>Road: </b></td>) .
694                              qq(<td class="Blu">Active</td></tr>));
695             }
696             else {
697                push(@$Array, qq(<tr><td align="right"><b>Road: </b></td>) .
698                              qq(<td class="blu">idle</td></tr>));
699             }
700             # ---
701             if ($aprE == 1) {
702                push(@$Array, qq(<tr><td align="right"><b>AprE: </b></td>) .
703                              qq(<td class="Blu">Active</td></tr>));
704             }
705             else {
706                push(@$Array, qq(<tr><td align="right"><b>AprE: </b></td>) .
707                              qq(<td class="blu">idle</td></tr>));
708             }
709             # ---
710             push(@$Array, qq(</table></div><br>));
711
712          }
713       }
714       # Next table data row.
715       @$Array[$#$Array] =~ s#<br>$#</td><td width=200px align="right">#;
716    }
717
718    # Finish the HTML page.
719    push(@$Array, qq(<div id="ImageContainer"><img class="GradeImage" src=) .
720                  qq("WigWag.gif" ALT="WigWag.gif"></div>));
```

```
721     push(@$Array, qq(</td></tr></table><br>));
722     &GenNavBar($Array, $Request);
723     push(@$Array, qq(</div>));
724     return 0;
725  }
726
727  # ============================================================================
728  # FUNCTION:  SensorPageData
729  #
730  # DESCRIPTION:
731  #    This routine is called to write the sensor related HTML and data to the
732  #    specified array. Sensor status is obtained from the sensor.dat file.
733  #    Refer to the DnB.pl %SensorBit hash.
734  #
735  #    sensor.dat       (generated by main loop)
736  #       Sensor: 32 sensor bits as a numeric value.
737  #          bit position: 1 = active, 0 = idle.
738  #
739  # CALLING SYNTAX:
740  #    $result = &SensorPageData($Array, $Request);
741  #
742  # ARGUMENTS:
743  #    $Array               Pointer to array for records.
744  #    $Request             Pointer to request data hash.
745  #
746  # RETURNED VALUES:
747  #    0 = Success,  1 = Error.
748  #
749  # ACCESSED GLOBAL VARIABLES:
750  #    None.
751  # ============================================================================
752  sub SensorPageData {
753     my($Array, $Request) = @_;
754     my(@data, @bits);
755     my($sensorBits) = 0;                    # No sensor bits set.
756     my($bitMask) = 0x10000;                 # Start at S01 bit position.
757     my($tStr) = strftime "%r", localtime;
758     my(%sensorDesc) = ('S01' => '2-GPIOA0: B03 to holdover entry.',
759                        'S02' => '2-GPIOA1: Holdover track 2 exit.',
760                        'S03' => '2-GPIOA2: Holdover track 1 exit.',
761                        'S04' => '2-GPIOA3: spare.',
762                        'S05' => '2-GPIOA4: B04 exit to B03 (Close T05).',
763                        'S06' => '2-GPIOA5: B05 exit to B06 (Open T06).',
764                        'S07' => '2-GPIOA6: B06 to Wye entry.',
765                        'S08' => '2-GPIOA7: B07 to Wye entry via yard track 1.',
766                        'S09' => '2-GPIOB0: B08 to Wye entry via yard track 2.',
767                        'S10' => '2-GPIOB1: Holdover track 1 exit yellow.',
768                        'S11' => '2-GPIOB2: Holdover track 1 exit red.',
769                        'S12' => '2-GPIOB3: Holdover track 2 exit yellow.',
770                        'S13' => '2-GPIOB4: Holdover track 2 exit red.');
771
772     # Start the HTML page.
773     push(@$Array, qq(<div class="SensorTitle"><h1>D&amp;B Sensor Status</h1></div>));
774     push(@$Array, qq(<div class="SensorBack">));
775     push(@$Array, qq(<table align="center"><tr><td class="SensorSnap"><b>Snapshot ) .
776                   qq(time:</b>  $tStr</td></tr><tr><td> </td></tr></table>));
777     push(@$Array, qq(<table class="Sensor"));
778     push(@$Array, qq(<colgroup><col width=70px><col width=80px></colgroup>));
779     push(@$Array, qq(<tr><th>Sensor</th><th>State</th><th>Description</th></tr>));
780
```

```
781     # Get the sensor bit data from the file.
782     unless (&ReadFile("$$Request{SHARE}/sensor.dat", \@data, "NoTrim")) {
783        @bits = grep /Sensor:/, @data;
784        if ($bits[0] =~ m/^Sensor:\s*(\d+)/) {
785           $sensorBits = $1;
786        }
787     }
788     &DisplayDebug(1, "SensorPageData, sensorBits: " . sprintf("%0.32b", $sensorBits));
789
790     # Build the table records HTML.
791     foreach my $sensor (sort keys(%sensorDesc)) {
792        &DisplayDebug(1, "SensorPageData,    bitmask: " . sprintf("%0.32b", $bitMask) .
793                         "  $sensor");
794        if ($sensorDesc{$sensor} =~ m/spare/i) {
795           push(@$Array, qq(<tr class="grayout"><td> $sensor</td><td> ) .
796                         qq( idle</td><td>$sensorDesc{$sensor}</td></tr>));
797        }
798        elsif (($sensorBits & $bitMask) != 0) {
799           push(@$Array, qq(<tr><td> $sensor</td><td class="Blu">Active</td>) .
800                         qq(<td>$sensorDesc{$sensor}</td></tr>));
801        }
802        else {
803           push(@$Array, qq(<tr><td> $sensor</td><td class="blu">  ) .
804                         qq(idle</td><td>$sensorDesc{$sensor}</td></tr>));
805        }
806        $bitMask = $bitMask << 1;  # Move mask to next sensor bit position.
807     }
808
809     # Finish the HTML page.
810     push(@$Array, qq(</table></div><br><br>));
811     &GenNavBar($Array, $Request);
812     push(@$Array, qq(</div>));
813     return 0;
814  }
815
816  # ============================================================================
817  # FUNCTION:  SignalPageData
818  #
819  # DESCRIPTION:
820  #    This routine is called to write the signal related HTML and data to the
821  #    specified array. Signal status is obtained from the sensor.dat file.
822  #    Refer to the DnB.pl %SignalData hash.
823  #
824  #    sensor.dat       (generated by main loop)
825  #       Signal: L01=x,L02=x, ... L12=x
826  #          x = 'Off', 'Grn', 'Yel', or 'Red'.
827  #
828  # CALLING SYNTAX:
829  #    $result = &SignalPageData($Array, $Request);
830  #
831  # ARGUMENTS:
832  #    $Array              Pointer to array for records.
833  #    $Request            Pointer to request data hash.
834  #
835  # RETURNED VALUES:
836  #    0 = Success,  1 = Error.
837  #
838  # ACCESSED GLOBAL VARIABLES:
839  #    None.
840  # ============================================================================
```

```
841  sub SignalPageData {
842     my($Array, $Request) = @_;
843     my(@data, @signals, $sigList, $color);
844     my($tStr) = strftime "%r", localtime;
845     my(%signalDesc) = ('L01' => '00,01: Holdover to B03 upgrade.',
846                        'L02' => '02,03: B04 / B05 to B03 downgrade.',
847                        'L03' => '04,05: B03 to B04 upgrade.',
848                        'L04' => '06,07: B06 to B04 downgrade.',
849                        'L05' => '08,09: B03 to B05 upgrade.',
850                        'L06' => '10,11: B06 to B05 downgrade.',
851                        'L07' => '12,13: B04 / B05 to B06 upgrade.',
852                        'L08' => '14,15: B07 / B08 to B06 downgrade. (sem)',
853                        'L09' => '16,17: B06 to B07 upgrade.',
854                        'L10' => '18,19: B09 / B10 to B07 downgrade.',
855                        'L11' => '20,21: B06 to B08 upgrade.',
856                        'L12' => '22,23: B09 / B10 to B08 downgrade.');
857
858     # Start the HTML page.
859     push(@$Array, qq(<div class="SignalTitle"><h1>D&amp;B Signal Status</h1></div>));
860     push(@$Array, qq(<div class="SignalBack">));
861     push(@$Array, qq(<table align="center"><tr><td class="SignalSnap"><b>Snapshot) .
862                   qq( time:</b>  $tStr</td></tr><tr><td> </td></tr>) .
863                   qq(</table>));
864     push(@$Array, qq(<table class="Signal"));
865     push(@$Array, qq(<colgroup><col width=70px><col width=70px></colgroup>));
866     push(@$Array, qq(<tr><th>Signal</th><th>State</th><th>Description</th></tr>));
867
868     # Get the signal data from the file.
869     unless (&ReadFile("$$Request{SHARE}/sensor.dat", \@data, "NoTrim")) {
870        @signals = grep /Signal:/, @data;
871        if ($signals[0] =~ m/^Signal:\s*(.+)/) {
872           $sigList = $1;
873        }
874     }
875
876     # Build the table records HTML.
877     foreach my $signal (sort keys(%signalDesc)) {
878        if ($sigList =~ m/$signal=(.{3})/) {
879           $color = $1;
880        }
881        else {
882           $color = '===';    # If we don't match.
883        }
884
885        if ($color =~ m/Red/i) {
886           push(@$Array, qq(<tr><td> $signal</td><td class="red"> Red</td>) .
887                         qq(</td><td>$signalDesc{$signal}</td></tr>));
888        }
889        elsif ($color =~ m/Yel/i) {
890           push(@$Array, qq(<tr><td> $signal</td><td class="yel"> Yel</td>) .
891                         qq(</td><td>$signalDesc{$signal}</td></tr>));
892        }
893        elsif ($color =~ m/Grn/i) {
894           push(@$Array, qq(<tr><td> $signal</td><td class="grn"> Grn</td>) .
895                         qq(</td><td>$signalDesc{$signal}</td></tr>));
896        }
897        elsif ($color =~ m/===/i) {
898           push(@$Array, qq(<tr><td> $signal</td><td class="blu"> $color) .
899                         qq(</td></td><td>$signalDesc{$signal}</td></tr>));
900        }
```

```
901         else {
902            push(@$Array, qq(<tr><td> $signal</td><td class="blu"> Off</td>) .
903                          qq(</td><td>$signalDesc{$signal}</td></tr>));
904         }
905      }
906
907      # Finish the HTML page.
908      push(@$Array, qq(</table></div><br><br>));
909      &GenNavBar($Array, $Request);
910      push(@$Array, qq(</div>));
911      return 0;
912   }
913
914   # ============================================================================
915   # FUNCTION:  TurnoutPageData
916   #
917   # DESCRIPTION:
918   #    This routine is called to write the turnout related HTML and data to the
919   #    specified array. Turnout status is obtained from the sensor.dat file.
920   #    Refer to the DnB.pl %TurnoutData hash.
921   #
922   #    sensor.dat       (generated by main loop)
923   #       T01=<value1>:<value2>: ... <value8>
924   #       T02=<value1>:<value2>: ... <value8>
925   #       ...
926   #
927   #       value order = Pos, Rate, Open, Middle, Close, MinPos, MaxPos, Id
928   #
929   # CALLING SYNTAX:
930   #    $result = &TurnoutPageData($Array, $Request);
931   #
932   # ARGUMENTS:
933   #    $Array               Pointer to array for records.
934   #    $Request             Pointer to request data hash.
935   #
936   # RETURNED VALUES:
937   #    0 = Success,  1 = Error.
938   #
939   # ACCESSED GLOBAL VARIABLES:
940   #    None.
941   # ============================================================================
942   sub TurnoutPageData {
943      my($Array, $Request) = @_;
944      my(@data, @tData, @tParm, $html, $x, $pos, $spare);
945      my($tStr) = strftime "%r", localtime;
946
947      # Start the HTML page.
948      push(@$Array, qq(<div class="TurnoutTitle"><h1>D&amp;amp;B Turnout Status</h1></div>));
949      push(@$Array, qq(<div class="TurnoutBack">));
950      push(@$Array, qq(<table align="center"><tr><td class="TurnoutSnap"><b>Snapshot ) .
951                    qq(time:</b>  $tStr<br><br></td></tr></table>));
952      push(@$Array, qq(<table class="Turnout"));
953      push(@$Array, qq(<colgroup><col width=45px><col width=45px><col width=45px>));
954      push(@$Array, qq(<col width=45px><col width=45px><col width=45px><col width=45px>));
955      push(@$Array, qq(<col width=45px><col width=230px></colgroup>));
956
957      push(@$Array, qq(<tr><th>Id</th><th>Pos</th><th>Rate</th><th>Open</th>));
958      push(@$Array, qq(<th>Midl</th><th>Close</th><th>MinP</th><th>MaxP</th>));
959      push(@$Array, qq(<th>Description</th></tr>));
960
```

```
961      # Get the turnout data from the file.
962      unless (&ReadFile("$$Request{SHARE}/sensor.dat", \@data, "NoTrim")) {
963
964      # Build the table records HTML.
965         foreach my $tNmbr (1..32) {
966            $tNmbr = "0${tNmbr}" if (length($tNmbr) == 1);
967            $tNmbr = join('', 'T', $tNmbr);
968            @tData = grep /^$tNmbr=/, @data;
969            chomp($tData[0]);
970            &DisplayDebug(1, "TurnoutPageData, tNmbr: $tNmbr   tData[0]: '$tData[0]'");
971            if ($tData[0] =~ m/^$tNmbr=(.+)/) {
972               @tParm = split(':', $1);
973               if ($tParm[$#tParm] =~ m/spare/i) {                  # Grayout spares
974                  $html = qq(<tr class="grayout"><td>$tNmbr</td>);
975                  $spare = 1;
976               }
977               else {
978                  $html = qq(<tr><td>$tNmbr</td>);
979                  $spare = 0;
980               }
981               for ($x = 0; $x <= $#tParm; $x++) {
982                  $pos = $tParm[$x] if ($x == 0);  # Copy pos for open/close color check.
983
984                  # Account for temperature adjusted pos value.
985                  if ($x == 2 and $spare == 0 and $pos > ($tParm[$x]-10) and
986                      $pos < ($tParm[$x]+10)) {
987                     $html = join('', $html, qq(<td class="red">$tParm[$x]</td>));
988                  }
989                  elsif ($x == 3 and $spare == 0 and $pos > ($tParm[$x]-10) and
990                         $pos < ($tParm[$x]+10)) {
991                     $html = join('', $html, qq(<td class="yel">$tParm[$x]</td>));
992                  }
993                  elsif ($x == 4 and $spare == 0 and $pos > ($tParm[$x]-10) and
994                         $pos < ($tParm[$x]+10)) {
995                     $html = join('', $html, qq(<td class="grn">$tParm[$x]</td>));
996                  }
997                  else {
998                     $html = join('', $html, qq(<td>$tParm[$x]</td>));
999                  }
1000              }
1001              $html = join('', $html, '</tr>');
1002              push(@$Array, $html);
1003           }
1004        }
1005     }
1006
1007     # Finish the HTML page.
1008     push(@$Array, qq(</table></div><br>));
1009     &GenNavBar($Array, $Request);
1010     push(@$Array, qq(</div>));
1011     return 0;
1012  }
1013
1014  # ============================================================================
1015  # FUNCTION:  MainLiveData
1016  #
1017  # DESCRIPTION:
1018  #    This routine is called to add mainline live data to the specified array.
1019  #    This page displays layout information in near real time in the browser. Java
1020  #    script is added to the HTML page header to instruct the browser to refresh
```

```
1021  #    the overlay images about every two seconds.
1022  #
1023  #    The overlay images are specified as .dat files. The NewConnection code
1024  #    substitutes the current main line specified overlay file when processing
1025  #    the request.  CSS z-index is used to stack the overlays for proper display.
1026  #
1027  # CALLING SYNTAX:
1028  #    $result = &MainLiveData($Array, $Request);
1029  #
1030  # ARGUMENTS:
1031  #    $Array              Pointer to array for records.
1032  #    $Request            Pointer to request data hash.
1033  #
1034  # RETURNED VALUES:
1035  #    0 = Success,  1 = Error.
1036  #
1037  # ACCESSED GLOBAL VARIABLES:
1038  #    None.
1039  # =============================================================================
1040  sub MainLiveData {
1041     my($Array, $Request) = @_;
1042
1043     push(@$Array, qq(<div class="MainLiveTitle"><h1>D&amp;B Model Railroad Mainline ) .
1044                   qq(Live</h1></div>));
1045     push(@$Array, qq(<img class="MainBackImage" src="TrackPlan.png" ) .
1046                   qq(alt="TrackPlan.png"></div>));
1047
1048     push(@$Array, qq(<img id="y-Ovr" class="yardImage" src="y-overlay.dat" ) .
1049                   qq(alt="y-overlay.dat">));
1050     push(@$Array, qq(<img id="m-Ovr" class="midwayImage" src="m-overlay.dat" ) .
1051                   qq(alt="m-overlay.dat">));
1052     push(@$Array, qq(<img id="h-Ovr" class="holdoverImage" src="h-overlay.dat" ) .
1053                   qq(alt="h-overlay.dat">));
1054
1055     push(@$Array, qq(<img id="L01-Ovr" class="L01color" src="L01-overlay.dat" ) .
1056                   qq(alt="L01-overlay.dat">));
1057     push(@$Array, qq(<img id="L02-Ovr" class="L02color" src="L02-overlay.dat" ) .
1058                   qq(alt="L02-overlay.dat">));
1059     push(@$Array, qq(<img id="L03-Ovr" class="L03color" src="L03-overlay.dat" ) .
1060                   qq(alt="L03-overlay.dat">));
1061     push(@$Array, qq(<img id="L04-Ovr" class="L04color" src="L04-overlay.dat" ) .
1062                   qq(alt="L04-overlay.dat">));
1063     push(@$Array, qq(<img id="L05-Ovr" class="L05color" src="L05-overlay.dat" ) .
1064                   qq(alt="L05-overlay.dat">));
1065     push(@$Array, qq(<img id="L06-Ovr" class="L06color" src="L06-overlay.dat" ) .
1066                   qq(alt="L06-overlay.dat">));
1067     push(@$Array, qq(<img id="L07-Ovr" class="L07color" src="L07-overlay.dat" ) .
1068                   qq(alt="L07-overlay.dat">));
1069     push(@$Array, qq(<img id="L08-Ovr" class="L08color" src="L08-overlay.dat" ) .
1070                   qq(alt="L08-overlay.dat">));
1071     push(@$Array, qq(<img id="L09-Ovr" class="L09color" src="L09-overlay.dat" ) .
1072                   qq(alt="L09-overlay.dat">));
1073     push(@$Array, qq(<img id="L10-Ovr" class="L10color" src="L10-overlay.dat" ) .
1074                   qq(alt="L10-overlay.dat">));
1075     push(@$Array, qq(<img id="L11-Ovr" class="L11color" src="L11-overlay.dat" ) .
1076                   qq(alt="L11-overlay.dat">));
1077     push(@$Array, qq(<img id="L12-Ovr" class="L12color" src="L12-overlay.dat" ) .
1078                   qq(alt="L12-overlay.dat">));
1079
1080     push(@$Array, qq(<img id="GC01-Ovr" class="GC01Image" src="GC01-overlay.dat" ) .
```

```
                    qq(alt="GC01-overlay.dat">));
    push(@$Array, qq(<img id="GC02-Ovr" class="GC02Image" src="GC02-overlay.dat" ) .
                    qq(alt="GC02-overlay.dat">));
    &GenNavBar($Array, $Request);
    push(@$Array, qq(<div class="LiveEndPad"> </div>));
    return 0;
}

# ============================================================================
# FUNCTION:  YardLiveData
#
# DESCRIPTION:
#    This routine is called to add yard live data to the specified array. This
#    page displays layout information in near real time in the browser. Java
#    script is added to the HTML page header to instruct the browser to refresh
#    the overlay images about every two seconds.
#
#    The yard trackage diagram is divided into multiple sections based upon
#    certain turnouts. The tracks in each section are colored with overlayss
#    using the turnout positions within the section. CSS z-index is used to
#    stack the overlays for proper display.
#
# CALLING SYNTAX:
#    $result = &MainLiveData($Array, $Request);
#
# ARGUMENTS:
#    $Array               Pointer to array for records.
#    $Request             Pointer to request data hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub YardLiveData {
    my($Array, $Request) = @_;

    push(@$Array, qq(<div class="YardLiveTitle"><h1>D&amp;B Model Railroad Yard ) .
                    qq(Live</h1></div>));
    push(@$Array, qq(<img class="YardBackImage" src="YardWebBase.png" ) .
                    qq(alt="YardWebBase.png"></div>));
    push(@$Array, qq(<img id="S1-Ovr" class="Yard-S1" src="Yard-S1-overlay.dat" ) .
                    qq(alt="Yard-S1-overlay.dat"></div>));
    push(@$Array, qq(<img id="S2-Ovr" class="Yard-S2" src="Yard-S2-overlay.dat" ) .
                    qq(alt="Yard-S2-overlay.dat"></div>));
    push(@$Array, qq(<img id="S3-Ovr" class="Yard-S3" src="Yard-S3-overlay.dat" ) .
                    qq(alt="Yard-S3-overlay.dat"></div>));
    push(@$Array, qq(<img id="S4-Ovr" class="Yard-S4" src="Yard-S4-overlay.dat" ) .
                    qq(alt="Yard-S4-overlay.dat"></div>));
    push(@$Array, qq(<img id="S5-Ovr" class="Yard-S5" src="Yard-S5-overlay.dat" ) .
                    qq(alt="Yard-S5-overlay.dat"></div>));
    push(@$Array, qq(<img id="S6-Ovr" class="Yard-S6" src="Yard-S6-overlay.dat" ) .
                    qq(alt="Yard-S6-overlay.dat"></div>));
    &GenNavBar($Array, $Request);
    push(@$Array, qq(<div class="LiveEndPad"> </div>));
    return 0;
}

# ============================================================================
```

```
1141  # FUNCTION:  GenNavBar
1142  #
1143  # DESCRIPTION:
1144  #    This routine is called to add the navigation button HTML to the specified
1145  #    array. The 'Top' page gets the page link buttons. All other pages have the
1146  #    'Home' and 'Refresh' buttons added to the page link buttons.
1147  #
1148  #
1149  # CALLING SYNTAX:
1150  #    $result = &GenNavBar($Array, $Request);
1151  #
1152  # ARGUMENTS:
1153  #    $Array              Pointer to array for records.
1154  #    $Request            Pointer to request data hash.
1155  #
1156  # RETURNED VALUES:
1157  #    0 = Success,  1 = Error.
1158  #
1159  # ACCESSED GLOBAL VARIABLES:
1160  #    None.
1161  # =============================================================================
1162  sub GenNavBar {
1163     my($Array, $Request) = @_;
1164
1165     if ($$Request{PAGE} =~ m/main/i) {
1166        push(@$Array, qq(<div class="navGroupLiveMain"><table class="navTable"><tr>));
1167     }
1168     elsif ($$Request{PAGE} =~ m/yard/i) {
1169        push(@$Array, qq(<div class="navGroupLiveYard"><table class="navTable"><tr>));
1170     }
1171     else {
1172        push(@$Array, qq(<div class="navGroup"><table class="navTable"><tr>));
1173     }
1174
1175     push(@$Array, qq(<td><button><a href="/block" class="navButton">Block</a>) .
1176                   qq(</button></td>));
1177     push(@$Array, qq(<td><button><a href="/grade" class="navButton">Grade</a>) .
1178                   qq(</button></td>));
1179     push(@$Array, qq(<td><button><a href="/sensor" class="navButton">Sensor</a>) .
1180                   qq(</button></td>));
1181     push(@$Array, qq(<td><button><a href="/signal" class="navButton">Signal</a>) .
1182                   qq(</button></td>));
1183     push(@$Array, qq(<td><button><a href="/turnout" class="navButton">Turnout</a>) .
1184                   qq(</button></td>));
1185
1186     if ($$Request{PAGE} =~ m/top/i) {
1187        push(@$Array, qq(</tr><tr><td> </td>));
1188        push(@$Array, qq(<td><button><a href="/main" class="navButton">Main</a>) .
1189                      qq(</button></td>));
1190        push(@$Array, qq(<td> </td>));
1191        push(@$Array, qq(<td><button><a href="/yard" class="navButton">Yard</a>) .
1192                      qq(</button></td><td> </td>));
1193     }
1194     elsif ($$Request{PAGE} =~ m/main/i or $$Request{PAGE} =~ m/yard/i) {
1195        push(@$Array, qq(</tr><tr><td> </td><td> </td>));
1196        push(@$Array, qq(<td><button><a href="/top" class="navButton">Home</a>) .
1197                      qq(</button></td>));
1198        push(@$Array, qq(<td> </td><td> </td>));
1199     }
1200     else {
```

```
      push(@$Array, qq(</tr><tr><td><button><a href="/top" class="navButton">) .
                    qq(Home</a></button></td>));
      push(@$Array, qq(<td><button><a href="/main" class="navButton">Main</a>) .
                    qq(</button></td>));
      push(@$Array, qq(<td> </td>));
      push(@$Array, qq(<td><button><a href="/yard" class="navButton">Yard</a>) .
                    qq(</button></td>));
      push(@$Array, qq(<td><button><a href="/$$Request{PAGE}" class="navButton">) .
                    qq(Refresh</a></button></td>));
   }
   push(@$Array, qq(</tr></table></div>));
   return 0;
}

# ============================================================================
# FUNCTION:  ExtractVariables
#
# DESCRIPTION:
#    This routine is called to parse the specified string for URL name/value
#    pairs and return them in the specified hash. Name/value pairs, if any,
#    begin after the first '?' character. Name and value are seperated by the
#    '=' character. Multiple name/value pairs are '&' seperated.
#
# CALLING SYNTAX:
#    $result = &ExtractVariables($Url, \%Variables);
#
# ARGUMENTS:
#    $Url                String to process.
#    $Variables          Pointer to hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ExtractVariables {
   my($Url, $Variables) = @_;
   my($data, @pairs, $name, $value);

   %$Variables = ();
   if ($Url =~ m/^(.+?)\?(.+)/) {
      $data = $2;
      if ($data ne '') {
         @pairs = split('&', $data);
         foreach my $pair (@pairs) {
            if ($pair =~ m/^(.+?)=(.+)$/) {
               $name = $1;
               $value = $2;
               $name =~ s/%(..)/chr(hex($1))/eg;
               $value =~ s/%(..)/chr(hex($1))/eg;
               $$Variables{$name} = $value;
            }
         }
      }
   }
   return 0;
}

return 1;
```