```perl
# ============================================================================
# FILE: DnB_Yard.pm                                                  10/18/2020
#
# SERVICES:  DnB YARD PROCESSING FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides yard track processing related functions used
#    by the DnB model railroad control program.
#
# PERL VERSION: 5.24.1
#
# ============================================================================
use strict;
# ----------------------------------------------------------------------------
# Package Declaration
# ----------------------------------------------------------------------------
package DnB_Yard;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   GetYardRoute
   YardRoute
   YardLiveOverlay
   TestSound
   TestRelay
);

use DnB_Message;
use DnB_Sensor;
use DnB_Turnout;
use Time::HiRes qw(sleep);

# ============================================================================
# FUNCTION:  GetYardRoute
#
# DESCRIPTION:
#    This routine processes yard route keypad input from the user. Input is
#    obtained by reading the stderr output of the KeypadChild process. This
#    data is a single character, 0 through F, corresponding to track numbers
#    1 through 16. Two track numbers define a from/to route. A route is valid
#    if present in the %YardRouteData hash. The route identifier is set for
#    processing by the YardRoute routine.
#
#    Some routes are special cases in that turnout positions are train move
#    direction dependent. In these cases, if the same route is consecutively
#    entered, the turnouts for the alternate move direction are set.
#
#    Normally, only the turnouts specific to the entered route are set. If
#    routes for the ends of tracks 3, 4, or 5 are specified within five
#    seconds of each other, the turnouts within the track will also be set
#    for yard pass-through. The local %routeCheck hash is used to check for
#    the possible user input permutations that are possible.
#
# CALLING SYNTAX:
#    $result = &GetYardRoute(\%YardRouteData, \%KeypadData, \%GpioData,
#                            $KeypadChildPid);
#
# ARGUMENTS:
#    $YardRouteData      Pointer to %YardRouteData hash.
```

```
#    $KeypadData         Pointer to %KeypadData hash.
#    $GpioData           Pointer to %GpioData hash.
#    $KeypadChildPid     Pid of Keypad child process.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub GetYardRoute {
   my($YardRouteData, $KeypadData, $GpioData, $KeypadChildPid) = @_;
   my($pressedKey, $route, $altRoute, @checkData);
   my($keypadId) = '01';
   my($cTime) = time;
   my(%routeCheck) = ('R02' => 'R12,R21,X02', 'R20' => 'R12,R21,X02',
                      'R12' => 'R02,R20,X12', 'R21' => 'R02,R20,X12',
                      'R03' => 'R13,R31,X03', 'R30' => 'R13,R31,X03',
                      'R13' => 'R03,R30,X13', 'R31' => 'R03,R30,X13',
                      'R04' => 'R14,R41,X04', 'R40' => 'R14,R41,X04',
                      'R14' => 'R04,R40,X14', 'R41' => 'R04,R40,X14');

   &DisplayDebug(2, "GetYardRoute entry ...");

   if ($$YardRouteData{'Control'}{'Inprogress'} == 0) {
      $pressedKey = Forks::Super::read_stderr($KeypadChildPid);

      if ($pressedKey ne '') {
         $pressedKey = substr($pressedKey, 0, 1);      # 1st character only.
         &DisplayDebug(1, "GetYardRoute, pressedKey: $pressedKey");

         if ($$KeypadData{$keypadId}{'Entry1'} == -1) {
            $$KeypadData{$keypadId}{'Entry1'} = $pressedKey;

            # Turn on 1st entry LED.
            $$GpioData{ $$KeypadData{$keypadId}{'Gpio'} }{'Obj'}->write(1);
            $$KeypadData{$keypadId}{'PressTime'} = $cTime + 5;
            &PlaySound("C.wav");
         }

         # Got 'from' and 'to' entries.
         else {
            $route = join('', 'R', $$KeypadData{$keypadId}{'Entry1'},
                          $pressedKey);
            $altRoute = join('', 'r', $$KeypadData{$keypadId}{'Entry1'},
                             $pressedKey);
            if (exists $$YardRouteData{$route}) {
               &PlaySound("G.wav");

               # Handle special route cases which involve %YardRouteData
               # entries with Rxx and rxx keys. A consecutive route entry
               # uses the alternate route key if available.
               if (exists($$YardRouteData{$altRoute}) and
                   $$YardRouteData{'Control'}{'Route'} eq $route) {
                  $route = $altRoute;
               }

               # Handle track 3, 4, and 5 end-to end routes. If the yard track
               # opposite end was entered < 5 seconds ago, change the route to
               # include the extra turnouts.
```

```
121                elsif (exists $routeCheck{$route}) {
122                   if ($cTime < $$YardRouteData{'Control'}{'RouteTime'}) {
123                      @checkData = split(',', $routeCheck{$route});
124                      if ($$YardRouteData{'Control'}{'Route'} eq $checkData[0] or
125                          $$YardRouteData{'Control'}{'Route'} eq $checkData[1]) {
126                         $route = $checkData[2];
127                      }
128                      $$YardRouteData{'Control'}{'RouteTime'} = $cTime;
129                   }
130                   else {
131                      $$YardRouteData{'Control'}{'RouteTime'} = $cTime + 5;
132                   }
133                }
134
135                # Initiate turnout setting for specified route.
136                $$YardRouteData{'Control'}{'Route'} = $route;
137                $$YardRouteData{'Control'}{'Inprogress'} = 1;
138                $$YardRouteData{'Control'}{'Step'} = 0;
139             }
140             else {
141                &PlaySound("CA.wav");
142             }
143
144             # Turn off 1st entry LED.
145             $$GpioData{ $$KeypadData{$keypadId}{'Gpio'} }{'Obj'}->write(0);
146             $$KeypadData{$keypadId}{'Entry1'} = -1;
147          }
148       }
149       elsif ($$KeypadData{$keypadId}{'Entry1'} != -1) {
150
151          # Abort 1st entry if a second keypress is not entered before
152          # timeout expiration.
153          if ($cTime > $$KeypadData{$keypadId}{'PressTime'}) {
154
155             # Turn off 1st entry LED.
156             $$GpioData{ $$KeypadData{$keypadId}{'Gpio'} }{'Obj'}->write(0);
157             $$KeypadData{$keypadId}{'Entry1'} = -1;
158          }
159       }
160    }
161    return 0;
162 }
163
164 # ============================================================================
165 # FUNCTION:  YardRoute
166 #
167 # DESCRIPTION:
168 #    This routine performs the operational functions related to yard trackage
169 #    routing. Only one turnout of a valid route list is positioned for each
170 #    call to minimize CPU loading. 'Inprogress' is reset when all turnouts for
171 #    the route have be positioned.
172 #
173 # CALLING SYNTAX:
174 #    $result = &YardRoute(\%YardRouteData, \%TurnoutData);
175 #
176 # ARGUMENTS:
177 #    $YardRouteData      Pointer to %YardRouteData hash.
178 #    $TurnoutData        Pointer to %TurnoutData hash.
179 #
180 # RETURNED VALUES:
```

```perl
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub YardRoute {
   my($YardRouteData, $TurnoutData) = @_;
   my($route, @routeList, $step, $turnout, $position, $moveResult);

   &DisplayDebug(2, "YardRoute entry ...");

   if ($$YardRouteData{'Control'}{'Inprogress'} == 1) {
      $route = $$YardRouteData{'Control'}{'Route'};
      if ($route ne "") {
         @routeList = split(',', $$YardRouteData{$route});
         &DisplayDebug(2, "YardRoute, route: $route   routeList: @routeList");
         if ($#routeList >= 0) {
            $step = $$YardRouteData{'Control'}{'Step'};
            if ($step <= $#routeList) {
               if ($routeList[$step] =~ m/^T(\d\d):(.+)/) {
                  $turnout = $1;
                  $position = $2;
                  &DisplayMessage("YardRoute, Route: $route, Step: " .
                                          "$step - $turnout:$position");
                  $$YardRouteData{'Control'}{'Step'}++;   # Increment step.
                  $moveResult = &MoveTurnout($position, $turnout, $TurnoutData);
                  if ($moveResult == 1) {
                     &DisplayError("YardRoute, Failed to set turnout " .
                                   "$turnout to $position");
                  }
               }
               else {
                  &DisplayError("YardRoute, Invalid route: $route step: $step.");
                  $$YardRouteData{'Control'}{'Route'} = "";
                  $$YardRouteData{'Control'}{'Inprogress'} = 0;
               }
            }
            else {                          # === Route is fully processed. ===
               $$YardRouteData{'Control'}{'Inprogress'} = 0;
               # Retain 'Route'. Last needed for detection of special cases.
            }
         }
         else {
            &DisplayError("YardRoute, No turnout entries in route '$route'.");
            $$YardRouteData{'Control'}{'Route'} = "";
            $$YardRouteData{'Control'}{'Inprogress'} = 0;
         }
      }
      else {
         $$YardRouteData{'Control'}{'Inprogress'} = 0;
      }
   }

   return 0;
}

# ============================================================================
# FUNCTION:  YardLiveOverlay
#
# DESCRIPTION:
```

```
241  #    This routine is periodically called by the main loop to set the image
242  #    overlay files used by the Yard Live webpage. These overlay files color the
243  #    yard tracks to show the current turnout lined routes. This is accomplished
244  #    by reading the turnout positions within each yard section and selecting the
245  #    appropriate image overlay file.
246  #
247  #    The @turnout position array must be formatted as follows.
248  #
249  #       T01=<value1>:<value2>: ... <value8>
250  #       T02=<value1>:<value2>: ... <value8>
251  #       ...
252  #
253  #       value order = Pos, Rate, Open, Middle, Close, MinPos, MaxPos, Id
254  #
255  # CALLING SYNTAX:
256  #    $result = &YardLiveOverlay(\@TurnoutPos, $WebDataDir);
257  #
258  # ARGUMENTS:
259  #    $TurnoutPos        Pointer to turnout position data array.
260  #    $WebDataDir        Directory path for output file.
261  #
262  # RETURNED VALUES:
263  #    0 = Success,  1 = Error.
264  #
265  # ACCESSED GLOBAL VARIABLES:
266  #    None.
267  # ============================================================================
268  sub YardLiveOverlay {
269     my($TurnoutPos, $WebDataDir) = @_;
270     my(@tData, @tParm, @tPos, @overlayFile, $posList, $cnt, $file);
271
272     # The %sections hash holds the turnouts that must be taken into consideration
273     # for each section.
274     my(%sections) = ('S1' => ['T22','T23','T24','T25'],
275                      'S2' => ['T12','T16','T18','T19','T23'],
276                      'S3' => ['T15','T16','T17','T20','T21'],
277                      'S4' => ['T08','T09','T13','T14','T26'],
278                      'S5' => ['T10','T11','T14','T15','T17'],
279                      'S6' => ['T11','T27']);
280
281     # The overlay hash holds the mapping between the section's turnout positions
282     # and the corresponding overlay image file. The matching positions are specified
283     # by the secondary hash index.
284     my(%overlay) = (
285        'S1' => {'T22c:T23o:T24c' => 'S1-T22cT23oT24c.png',
286                 'T22c:T23o:T24o:T25c' => 'S1-T22cT23oT24oT25c.png',
287                 'T22c:T23o:T24o:T25o' => 'S1-T22cT23oT24oT25o.png',
288                 'T22o:T24c' => 'S1-T22oT24c.png',
289                 'T22o:T24o:T25c' => 'S1-T22oT24oT25c.png',
290                 'T22o:T24o:T25o' => 'S1-T22oT24oT25o.png'},
291        'S2' => {'T12c:T16c' => 'S2-T12cT16c.png',
292                 'T12c:T16o:T18c' => 'S2-T12cT16oT18c.png',
293                 'T12c:T16o:T18o:T19c' => 'S2-T12cT16oT18oT19c.png',
294                 'T12c:T16o:T18o:T19o:T23c' => 'S2-T12cT16oT18oT19oT23c.png',
295                 'T12o' => 'S2-T12o.png'},
296        'S3' => {'T15c:T16c:T17c' => 'S3-T15cT16cT17c.png',
297                 'T17o:T20c' => 'S3-T17oT20c.png',
298                 'T17o:T20o:T21c' => 'S3-T17oT20oT21c.png',
299                 'T17o:T20o:T21o' => 'S3-T17oT20oT21o.png'},
300        'S4' => {'T08c:T09o:T26c' => 'S4-T08cT09oT26c.png',
```

```
301                 'T08c:T09o:T26o' => 'S4-T08cT09oT26o.png',
302                 'T08c:T09c:T13c' => 'S4-T08cT09cT13c.png',
303                 'T08c:T09c:T13c:T14c' => 'S4-T08cT09cT13cT14c.png',
304                 'T08o' => 'S4-T08o.png'},
305          'S5' => {'T10c:T11o:T14c' => 'S5-T10cT11oT14c.png',
306                 'T10c:T11o:T14o' => 'S5-T10cT11oT14o.png',
307                 'T10o:T15c:T17c' => 'S5-T10oT15cT17c.png',
308                 'T10o:T15c:T17o' => 'S5-T10oT15cT17o.png',
309                 'T10o:T15o' => 'S5-T10oT15o.png'},
310          'S6' => {'T11c:T27c' => 'S6-T11cT27c.png',
311                 'T11c:T27o' => 'S6-T11cT27o.png',
312                 'T11o' => 'S6-T11o.png'});
313
314       foreach my $section (keys(%sections)) {
315          @tPos = ();
316          @overlayFile = (join('-', $section, 'NoTrack.png'));
317
318          # Get the current positions of the section turnouts.
319          foreach my $tNmbr (@{ $sections{$section} }) {
320             @tData = grep /^$tNmbr=/, @$TurnoutPos;
321             chomp($tData[0]);
322             if ($tData[0] =~ m/^$tNmbr=(.+)/) {
323                @tParm = split(':', $1);
324
325                # Account for temperature adjusted pos value.
326                if (@tParm[0] > ($tParm[2]-10) and @tParm[0] < ($tParm[2]+10)) {
327                   push (@tPos, "${tNmbr}o");
328                }
329                elsif (@tParm[0] > ($tParm[4]-10) and @tParm[0] < ($tParm[4]+10)) {
330                   push (@tPos, "${tNmbr}c");
331                }
332             }
333          }
334          $posList = join(',', @tPos);
335
336          # Check the section's overlay hash for a match and update @overlayFile
337          # value if found.
338          foreach my $indx (keys(%{$overlay{$section}})) {
339             @tPos = split(':', $indx);
340             $cnt = 0;
341             foreach my $t (@tPos) {
342                $cnt++ if ($posList =~ m/$t/);
343             }
344             if ($cnt == scalar @tPos) {
345                @overlayFile = ($overlay{$section}{$indx});
346                last;
347             }
348          }
349
350          # Store the overlay file name for Yard Live use.
351          $file = join('', $WebDataDir, '/Yard-', $section, '-overlay.dat');
352          &WriteFile($file, \@overlayFile, '');
353       }
354
355       return 0;
356    }
357
358    # ============================================================================
359    # FUNCTION:  TestSound
360    #
```

```perl
# DESCRIPTION:
#    This routine is used to select and audition the sound files in the sound
#    file directory when the -p command line option is specified.
#
# CALLING SYNTAX:
#    $result = &TestSound($SoundDir);
#
# ARGUMENTS:
#    $SoundDir     Directory holding sound files.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::MainRun, $main::SoundPlayer, $main::AudioVolume
# ==============================================================================
sub TestSound {
   my($SoundDir) = @_;
   my(@fileList, $cnt, $key, $resp, $volume);
   my(%select) = ('00' => 'Exit test.');

   &DisplayDebug(1, "TestSound entry ...   SoundDir: $SoundDir   " .
                    "SoundPlayer: $main::SoundPlayer");

   if (-d $SoundDir) {

      # Get wav file names, sort, and build user picklist.
      @fileList = sort grep { -f } glob "$SoundDir/*.wav";
      $cnt = 1;
      foreach my $file (@fileList) {
         $key = $cnt++;
         $key = "0$key" if (length($key) == 1);
         $select{$key} = substr($file, rindex($file, "/")+1);
      }

      #Display list to user and get selection.
      while ($main::MainRun) {
         &DisplayMessage("TestSound, ----------------------------");
         &DisplayMessage("TestSound, Enter file number to audition.");
         &DisplayMessage("TestSound, Include ,xx to change volume"  );
         &DisplayMessage("TestSound, from default $main::AudioVolume%.");
         foreach my $key (sort keys(%select)) {
            &DisplayMessage("TestSound,    $key: $select{$key}");
         }
         &DisplayMessage("TestSound, ----------------------------");
         print "$$ TestSound, Enter selection: ";
         $resp = <>;
         chomp($resp);
         if ($resp =~ m/(\d+),(.+)/) {
            $resp = $1;
            $volume = $2;
         }
         else {
            $volume = '';
         }
         $resp = "0$resp" if (length($resp) == 1);
         return 0 if ($resp eq '00');
         if (exists $select{$resp}) {
            if ($volume ne '') {
               if ($volume > 0 and $volume <= 99) {
```

```
421                     &PlaySound($select{$resp}, $volume);
422                  }
423                  else {
424                     &DisplayError("TestSound, Invalid volume: $volume");
425                  }
426               }
427               else {
428                  &PlaySound($select{$resp});
429                  &DisplayMessage("TestSound, playing selection $resp ...");
430               }
431            }
432            else {
433               &DisplayError("TestSound, Entry '$resp' not found.");
434            }
435         }
436      }
437      else {
438         &DisplayError("TestSound, Sound file directory not found: $SoundDir");
439         return 1;
440      }
441
442      return 0;
443   }
444
445   # ============================================================================
446   # FUNCTION:  TestRelay
447   #
448   # DESCRIPTION:
449   #    This routine is called by the DnB_main code to test the power polarity
450   #    relays when the -r command line option is specified. The specified relay,
451   #    or all if 0, is sequentially energized and de-energized at a five second
452   #    on/off rate. This test runs until terminated by ctrl-c.
453   #
454   # CALLING SYNTAX:
455   #    $result = &TestRelay($Relay, \%GpioData);
456   #
457   # ARGUMENTS:
458   #    $Relay         Relay number to test, 0 for all.
459   #    $GpioData      Pointer to %GpioData hash. (polarity relays)
460   #
461   # RETURNED VALUES:
462   #    0 = Success,  1 = Error.
463   #
464   # ACCESSED GLOBAL VARIABLES:
465   #    $main::MainRun
466   # ============================================================================
467   sub TestRelay {
468      my($Relay, $GpioData) = @_;
469      my($check, $relayNum);
470      my($value) = 1;
471
472      &DisplayDebug(1, "TestRelay entry ...   Relay: $Relay");
473      if ($Relay !~ m/^\d+$/ or $Relay < 0 or $Relay > 3) {
474         &DisplayError("TestRelay, Invalid relay number specified: '$Relay'");
475         return 1;
476      }
477
478   # Run test loop until terminated.
479      while ($main::MainRun) {
480         foreach my $gpio (sort keys(%$GpioData)) {
```

```
481             if ($gpio =~ m/^GP.+?_PR(\d\d)/) {
482                $relayNum = sprintf("%d", $1);
483                if ($Relay == $relayNum or $Relay == 0) {
484                   $$GpioData{$gpio}{'Obj'}->write($value);  # Set relay GPIO.
485                   $check = $$GpioData{$gpio}{'Obj'}->read;  # Readback and check.
486                   if ($check != $value) {
487                      &DisplayError("TestRelay, Failed to set $gpio (" .
488                                    $$GpioData{$gpio}{'Desc'} . ") to $value");
489                   }
490                   else {
491                      &DisplayMessage("TestRelay, $gpio (" . $$GpioData{$gpio}{'Desc'} .
492                                      ") set to $value");
493                   }
494                   sleep 0.5;         # Delay
495                }
496             }
497          }
498          sleep 5;
499          $value = (~$value) & 1;     # Compliment the working value.
500       }
501       return 0;
502    }
503
504    return 1;
505
```