```perl
# ============================================================================
# FILE: DnB_Yard.pm                                                  7/07/2020
#
# SERVICES:  DnB TRACK 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
   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.
               elsif (exists $routeCheck{$route}) {
```

```
                    if ($cTime < $$YardRouteData{'Control'}{'RouteTime'}) {
                       @checkData = split(',', $routeCheck{$route});
                       if ($$YardRouteData{'Control'}{'Route'} eq $checkData[0] or
                           $$YardRouteData{'Control'}{'Route'} eq $checkData[1]) {
                          $route = $checkData[2];
                       }
                       $$YardRouteData{'Control'}{'RouteTime'} = $cTime;
                    }
                    else {
                       $$YardRouteData{'Control'}{'RouteTime'} = $cTime + 5;
                    }
                 }

                 # Initiate turnout setting for specified route.
                 $$YardRouteData{'Control'}{'Route'} = $route;
                 $$YardRouteData{'Control'}{'Inprogress'} = 1;
                 $$YardRouteData{'Control'}{'Step'} = 0;
              }
              else {
                 &PlaySound("CA.wav");
              }

              # Turn off 1st entry LED.
              $$GpioData{ $$KeypadData{$keypadId}{'Gpio'} }{'Obj'}->write(0);
              $$KeypadData{$keypadId}{'Entry1'} = -1;
           }
        }
        elsif ($$KeypadData{$keypadId}{'Entry1'} != -1) {

           # Abort 1st entry if a second keypress is not entered before
           # timeout expiration.
           if ($cTime > $$KeypadData{$keypadId}{'PressTime'}) {

              # Turn off 1st entry LED.
              $$GpioData{ $$KeypadData{$keypadId}{'Gpio'} }{'Obj'}->write(0);
              $$KeypadData{$keypadId}{'Entry1'} = -1;
           }
        }
     }
     return 0;
  }

  # ============================================================================
  # FUNCTION:  YardRoute
  #
  # DESCRIPTION:
  #    This routine performs the operational functions related to yard trackage
  #    routing. Only one turnout of a valid route list is positioned for each
  #    call to minimize CPU loading. 'Inprogress' is reset when all turnouts for
  #    the route have be positioned.
  #
  # CALLING SYNTAX:
  #    $result = &YardRoute(\%YardRouteData, \%TurnoutData);
  #
  # ARGUMENTS:
  #    $YardRouteData      Pointer to %YardRouteData hash.
  #    $TurnoutData        Pointer to %TurnoutData hash.
  #
  # RETURNED VALUES:
  #    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:  TestSound
#
# 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 $volume <= 99) {
                  &PlaySound($select{$resp}, $volume);
               }
```

```
301                  else {
302                     &DisplayError("TestSound, Invalid volume: $volume");
303                  }
304               }
305               else {
306                  &PlaySound($select{$resp});
307                  &DisplayMessage("TestSound, playing selection $resp ...");
308               }
309            }
310            else {
311               &DisplayError("TestSound, Entry '$resp' not found.");
312            }
313         }
314      }
315      else {
316         &DisplayError("TestSound, Sound file directory not found: $SoundDir");
317         return 1;
318      }
319
320      return 0;
321   }
322
323   # ============================================================================
324   # FUNCTION:  TestRelay
325   #
326   # DESCRIPTION:
327   #    This routine is called by the DnB_main code to test the power polarity
328   #    relays when the -r command line option is specified. The specified relay,
329   #    or all if 0, is sequentially energized and de-energized at a five second
330   #    on/off rate. This test runs until terminated by ctrl-c.
331   #
332   # CALLING SYNTAX:
333   #    $result = &TestRelay($Relay, \%GpioData);
334   #
335   # ARGUMENTS:
336   #    $Relay         Relay number to test, 0 for all.
337   #    $GpioData      Pointer to %GpioData hash. (polarity relays)
338   #
339   # RETURNED VALUES:
340   #    0 = Success,  1 = Error.
341   #
342   # ACCESSED GLOBAL VARIABLES:
343   #    $main::MainRun
344   # ============================================================================
345   sub TestRelay {
346      my($Relay, $GpioData) = @_;
347      my($check, $relayNum);
348      my($value) = 1;
349
350      &DisplayDebug(1, "TestRelay entry ...   Relay: $Relay");
351      if ($Relay !~ m/^\d+$/ or $Relay < 0 or $Relay > 3) {
352         &DisplayError("TestRelay, Invalid relay number specified: '$Relay'");
353         return 1;
354      }
355
356   # Run test loop until terminated.
357      while ($main::MainRun) {
358         foreach my $gpio (sort keys(%$GpioData)) {
359            if ($gpio =~ m/^GP.+?_PR(\d\d)/) {
360               $relayNum = sprintf("%d", $1);
```

```
            if ($Relay == $relayNum or $Relay == 0) {
               $$GpioData{$gpio}{'Obj'}->write($value);  # Set relay GPIO.
               $check = $$GpioData{$gpio}{'Obj'}->read;  # Readback and check.
               if ($check != $value) {
                  &DisplayError("TestRelay, Failed to set $gpio (" .
                                $$GpioData{$gpio}{'Desc'} . ") to $value");
               }
               else {
                  &DisplayMessage("TestRelay, $gpio (" . $$GpioData{$gpio}{'Desc'} .
                                  ") set to $value");
               }
               sleep 0.5;         # Delay
            }
         }
      }
      sleep 5;
      $value = (~$value) & 1;     # Compliment the working value.
   }
   return 0;
}

return 1;
```