

```

1  # =====
2  # FILE: DnB_Yard.pm                                                    7/07/2020
3  #
4  # SERVICES:  DnB TRACK PROCESSING FUNCTIONS
5  #
6  # DESCRIPTION:
7  #   This perl module provides yard track processing related functions used
8  #   by the DnB model railroad control program.
9  #
10 # PERL VERSION: 5.24.1
11 #
12 # =====
13 use strict;
14 # -----
15 # Package Declaration
16 # -----
17 package DnB_Yard;
18 require Exporter;
19 our @ISA = qw(Exporter);
20
21 our @EXPORT = qw(
22     GetYardRoute
23     YardRoute
24     TestSound
25     TestRelay
26 );
27
28 use DnB_Message;
29 use DnB_Sensor;
30 use DnB_Turnout;
31 use Time::HiRes qw(sleep);
32
33 # =====
34 # FUNCTION:  GetYardRoute
35 #
36 # DESCRIPTION:
37 #   This routine processes yard route keypad input from the user. Input is
38 #   obtained by reading the stderr output of the KeypadChild process. This
39 #   data is a single character, 0 through F, corresponding to track numbers
40 #   1 through 16. Two track numbers define a from/to route. A route is valid
41 #   if present in the %YardRouteData hash. The route identifier is set for
42 #   processing by the YardRoute routine.
43 #
44 #   Some routes are special cases in that turnout positions are train move
45 #   direction dependent. In these cases, if the same route is consecutively
46 #   entered, the turnouts for the alternate move direction are set.
47 #
48 #   Normally, only the turnouts specific to the entered route are set. If
49 #   routes for the ends of tracks 3, 4, or 5 are specified within five
50 #   seconds of each other, the turnouts within the track will also be set
51 #   for yard pass-through. The local %routeCheck hash is used to check for
52 #   the possible user input permutations that are possible.
53 #
54 # CALLING SYNTAX:
55 #   $result = &GetYardRoute(\%YardRouteData, \%KeypadData, \%GpioData,
56 #                           $KeypadChildPid);
57 #
58 # ARGUMENTS:
59 #   $YardRouteData      Pointer to %YardRouteData hash.
60 #   $KeypadData         Pointer to %KeypadData hash.

```

```

61 # $GpioData          Pointer to %GpioData hash.
62 # $KeypadChildPid    Pid of Keypad child process.
63 #
64 # RETURNED VALUES:
65 #   0 = Success,  1 = Error.
66 #
67 # ACCESSED GLOBAL VARIABLES:
68 #   None.
69 # =====
70 sub GetYardRoute {
71   my($YardRouteData, $KeypadData, $GpioData, $KeypadChildPid) = @_;
72   my($pressedKey, $route, $altRoute, @checkData);
73   my($keypadId) = '01';
74   my($cTime) = time;
75   my(%routeCheck) = ( 'R02' => 'R12,R21,X02', 'R20' => 'R12,R21,X02',
76                       'R12' => 'R02,R20,X12', 'R21' => 'R02,R20,X12',
77                       'R03' => 'R13,R31,X03', 'R30' => 'R13,R31,X03',
78                       'R13' => 'R03,R30,X13', 'R31' => 'R03,R30,X13',
79                       'R04' => 'R14,R41,X04', 'R40' => 'R14,R41,X04',
80                       'R14' => 'R04,R40,X14', 'R41' => 'R04,R40,X14' );
81
82   &DisplayDebug(2, "GetYardRoute entry ...");
83
84   if ($$YardRouteData{'Control'}{'Inprogress'} == 0) {
85     $pressedKey = Forks::Super::read_stderr($KeypadChildPid);
86
87     if ($pressedKey ne '') {
88       $pressedKey = substr($pressedKey, 0, 1);      # 1st character only.
89       &DisplayDebug(1, "GetYardRoute, pressedKey: $pressedKey");
90
91       if ($$KeypadData{$keypadId}{'Entry1'} == -1) {
92         $$KeypadData{$keypadId}{'Entry1'} = $pressedKey;
93
94         # Turn on 1st entry LED.
95         $$GpioData{ $$KeypadData{$keypadId}{'Gpio'} }{'Obj'}->write(1);
96         $$KeypadData{$keypadId}{'PressTime'} = $cTime + 5;
97         &PlaySound("C.wav");
98       }
99
100      # Got 'from' and 'to' entries.
101      else {
102        $route = join(' ', 'R', $$KeypadData{$keypadId}{'Entry1'},
103                     $pressedKey);
104        $altRoute = join(' ', 'r', $$KeypadData{$keypadId}{'Entry1'},
105                        $pressedKey);
106        if (exists $$YardRouteData{$route}) {
107          &PlaySound("G.wav");
108
109          # Handle special route cases which involve %YardRouteData
110          # entries with Rxx and rxx keys. A consecutive route entry
111          # uses the alternate route key if available.
112          if (exists($$YardRouteData{$altRoute}) and
113              $$YardRouteData{'Control'}{'Route'} eq $route) {
114            $route = $altRoute;
115          }
116
117          # Handle track 3, 4, and 5 end-to end routes. If the yard track
118          # opposite end was entered < 5 seconds ago, change the route to
119          # include the extra turnouts.
120          elsif (exists $routeCheck{$route}) {

```

```

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

```

```

181 #
182 # ACCESSED GLOBAL VARIABLES:
183 #     None.
184 # =====
185 sub YardRoute {
186     my($YardRouteData, $TurnoutData) = @_;
187     my($route, @routeList, $step, $turnout, $position, $moveResult);
188
189     &DisplayDebug(2, "YardRoute entry ...");
190
191     if ($$YardRouteData{'Control'}{'Inprogress'} == 1) {
192         $route = $$YardRouteData{'Control'}{'Route'};
193         if ($route ne "") {
194             @routeList = split(',', $$YardRouteData{$route});
195             &DisplayDebug(2, "YardRoute, route: $route    routeList: @routeList");
196             if ($#routeList >= 0) {
197                 $step = $$YardRouteData{'Control'}{'Step'};
198                 if ($step <= $#routeList) {
199                     if ($routeList[$step] =~ m/^(\\d\\d):(\\d+)$/ ) {
200                         $turnout = $1;
201                         $position = $2;
202                         &DisplayMessage("YardRoute, Route: $route, Step: " .
203                                     "$step - $turnout:$position");
204                         $$YardRouteData{'Control'}{'Step'}++; # Increment step.
205                         $moveResult = &MoveTurnout($position, $turnout, $TurnoutData);
206                         if ($moveResult == 1) {
207                             &DisplayError("YardRoute, Failed to set turnout " .
208                                         "$turnout to $position");
209                         }
210                     }
211                 } else {
212                     &DisplayError("YardRoute, Invalid route: $route step: $step.");
213                     $$YardRouteData{'Control'}{'Route'} = "";
214                     $$YardRouteData{'Control'}{'Inprogress'} = 0;
215                 }
216             }
217             else {
218                 # === Route is fully processed. ===
219                 $$YardRouteData{'Control'}{'Inprogress'} = 0;
220                 # Retain 'Route'. Last needed for detection of special cases.
221             }
222             else {
223                 &DisplayError("YardRoute, No turnout entries in route '$route'.");
224                 $$YardRouteData{'Control'}{'Route'} = "";
225                 $$YardRouteData{'Control'}{'Inprogress'} = 0;
226             }
227         }
228         else {
229             $$YardRouteData{'Control'}{'Inprogress'} = 0;
230         }
231     }
232
233     return 0;
234 }
235
236 # =====
237 # FUNCTION: TestSound
238 #
239 # DESCRIPTION:
240 #     This routine is used to select and audition the sound files in the sound

```

```

241 # file directory when the -p command line option is specified.
242 #
243 # CALLING SYNTAX:
244 # $result = &TestSound($SoundDir);
245 #
246 # ARGUMENTS:
247 # $SoundDir Directory holding sound files.
248 #
249 # RETURNED VALUES:
250 # 0 = Success, 1 = Error.
251 #
252 # ACCESSED GLOBAL VARIABLES:
253 # $main::MainRun, $main::SoundPlayer, $main::AudioVolume
254 # =====
255 sub TestSound {
256     my($SoundDir) = @_;
257     my(@fileList, $cnt, $key, $resp, $volume);
258     my(%select) = ('00' => 'Exit test.');
```

```

259
260     &DisplayDebug(1, "TestSound entry ... SoundDir: $SoundDir " .
261                   "SoundPlayer: $main::SoundPlayer");
262
263     if (-d $SoundDir) {
264
265         # Get wav file names, sort, and build user picklist.
266         @fileList = sort grep { -f } glob "$SoundDir/*.wav";
267         $cnt = 1;
268         foreach my $file (@fileList) {
269             $key = $cnt++;
270             $key = "0$key" if (length($key) == 1);
271             $select{$key} = substr($file, rindex($file, "/")+1);
272         }
273
274         #Display list to user and get selection.
275         while ($main::MainRun) {
276             &DisplayMessage("TestSound, -----");
277             &DisplayMessage("TestSound, Enter file number to audition.");
278             &DisplayMessage("TestSound, Include ,xx to change volume");
279             &DisplayMessage("TestSound, from default $main::AudioVolume%.");
280             foreach my $key (sort keys(%select)) {
281                 &DisplayMessage("TestSound, $key: $select{$key}");
282             }
283             &DisplayMessage("TestSound, -----");
284             print "$$ TestSound, Enter selection: ";
285             $resp = <>;
286             chomp($resp);
287             if ($resp =~ m/(\d+),(.+)/) {
288                 $resp = $1;
289                 $volume = $2;
290             }
291             else {
292                 $volume = '';
293             }
294             $resp = "0$resp" if (length($resp) == 1);
295             return 0 if ($resp eq '00');
296             if (exists $select{$resp}) {
297                 if ($volume ne '') {
298                     if ($volume > 0 and $volume <= 99) {
299                         &PlaySound($select{$resp}, $volume);
300                     }

```

```

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/^\d+GP.+?_PR(\d\d)/) {
360                 $relayNum = sprintf("%d", $1);

```

```

361     if ($Relay == $relayNum or $Relay == 0) {
362         $$GpioData{$gpio}{'Obj'}->write($value); # Set relay GPIO.
363         $check = $$GpioData{$gpio}{'Obj'}->read; # Readback and check.
364         if ($check != $value) {
365             &DisplayError("TestRelay, Failed to set $gpio (" .
366                 $$GpioData{$gpio}{'Desc'} . ") to $value");
367         }
368         else {
369             &DisplayMessage("TestRelay, $gpio (" . $$GpioData{$gpio}{'Desc'} .
370                 ") set to $value");
371         }
372         sleep 0.5; # Delay
373     }
374 }
375 }
376 sleep 5;
377 $value = (~$value) & 1; # Compliment the working value.
378 }
379 return 0;
380 }
381
382 return 1;
383

```