

```

1  # =====
2  # FILE: DnB_GradeCrossing.pm                                     8/06/2020
3  #
4  # SERVICES:  DnB GRADE CROSSING FUNCTIONS
5  #
6  # DESCRIPTION:
7  #   This perl module provides grade crossing related functions used by the
8  #   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_GradeCrossing;
18 require Exporter;
19 our @ISA = qw(Exporter);
20
21 our @EXPORT = qw(
22     ProcessGradeCrossing
23     GcChildProcess
24     TestGradeCrossing
25 );
26
27 use DnB_Sensor;
28 use DnB_Signal;
29 use DnB_Turnout;
30 use DnB_Message;
31 use Forks::Super;
32 use Time::HiRes qw(sleep);
33
34 # =====
35 # FUNCTION:  ProcessGradeCrossing
36 #
37 # DESCRIPTION:
38 #   This routine is used to process the specified grade crossing. It is called
39 #   once an iteration by the main program loop. State data that is used for
40 #   grade crossing control is persisted in the %GradeCrossingData hash. Each
41 #   grade crossing is in one of the following states; 'idle', 'gateLower',
42 #   'approach', 'road', 'gateRaise' or 'depart'. %GradeCrossingData values,
43 #   sensor bits, and code within this routine, transition the signal through
44 #   these states. Operation is as follows.
45 #
46 #   1. Configuration and initializations set in %GradeCrossingData hash.
47 #
48 #   2. In 'idle' state, a train approaching the grade crossing is detected by
49 #   sensors 'AprEast', 'AprWest', or 'Road'. This causes the signals to begin
50 #   flashing. 'SigRun' is set to 'on'. 'GateDelay' is set and the state
51 #   transitions to 'gateLower'.
52 #
53 #   3. In 'gateLower' state, GateDelay is performed and the 'AprTimer' is set.
54 #   If gates are available, they are lowered. Then the state transitions to
55 #   'approach'. The GateDelay value is used to better simulate proto-typical
56 #   signal operation.
57 #
58 #   4. In 'approach' state, if 'road' state is not achieved before 'AprTimer'
59 #   expires, the code transitions to the 'gateRaise' state. This could occur if
60 #   the train stops or backs away before reaching the 'Road' sensor. An active

```

```

61 # 'Road' sensor causes transition to the 'road' state.
62 #
63 # 5. In 'road' state, a short timeout is set into 'RoadTimer'. Additional
64 # 'Road' sensor activity reloads this timer. This maintains 'road' state
65 # while the train occupies the grade crossing. When no further 'Road' sensor
66 # activity is reported, 'RoadTimer' will expire. The state transitions to
67 # 'gateRaise'.
68 #
69 # 6. In 'gateRaise' state, if grade crossing does not have gates, 'DepTimer'
70 # is set and the state transitions to 'depart'. Otherwise, the gates are
71 # raised. Once completed (servo pid == 0), 'DepTimer' is set and the state
72 # transitions to 'depart'.
73 #
74 # 7. In the 'depart' state, the signal lamp flashing is stopped and 'SigRun'
75 # is set to 'off'. Outbound train 'AprEast' or 'AprWest' sensor activity
76 # restarts the 'DepTimer' maintaining the 'depart' state. Once the last car
77 # of the outbound train is past the 'AprEast' or 'AprWest' sensor, the
78 # 'DepTimer' expires and the state transitions to 'idle'.
79 #
80 # If the train backs up, 'Road' sensor activity will transition the state to
81 # 'idle'. From 'idle', the active 'Road' sensor will start a new signaling
82 # cycle.
83 #
84 # CALLING SYNTAX:
85 # $result = &ProcessGradeCrossing($gc, \%GradeCrossingData, \%SensorBit,
86 # \%TurnoutData, \%MCP23017, \%SensorState);
87 #
88 # ARGUMENTS:
89 # $Gc Index to data in %GradeCrossingData.
90 # $GradeCrossingData Pointer to %GradeCrossingData hash.
91 # $SensorBit Pointer to %SensorBit hash.
92 # $TurnoutData Pointer to %TurnoutData hash. (needed for gates and sound)
93 # $MCP23017 Pointer to %MCP23017 hash. (GPIO definitions)
94 # $SensorState Pointer to %SensorState hash.
95 #
96 # RETURNED VALUES:
97 # 0 = Success, 1 = Error
98 #
99 # ACCESSED GLOBAL VARIABLES:
100 # None.
101 # =====
102 sub ProcessGradeCrossing {
103     my($Gc, $GradeCrossingData, $SensorBit, $TurnoutData, $MCP23017, $SensorState) = @_;
104     my(@gates);
105
106     # Isolate the current grade crossing sensor bit values and get the current time.
107
108     my($aprEastSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'AprEast'},
109                                         $SensorBit, $SensorState);
110     my($roadSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'Road'}, $SensorBit,
111                                     $SensorState);
112     my($aprWestSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'AprWest'},
113                                         $SensorBit, $SensorState);
114     my($cTime) = time;
115
116     &DisplayDebug(2, "ProcessGradeCrossing $Gc, State: " .
117                   "$$GradeCrossingData{$Gc}{'State'} aprEastSensor: $aprEastSensor" .
118                   " roadSensor: $roadSensor aprWestSensor: $aprWestSensor " .
119                   "cTime: $cTime");
120

```

```

121 # Idle state code. -----
122 if ($$GradeCrossingData{$Gc}{'State'} eq 'idle') {
123     if ($roadSensor == 1 or $aprEastSensor == 1 or $aprWestSensor == 1) {
124         if ($$GradeCrossingData{$Gc}{'SigRun'} ne 'on') {
125             &DisplayMessage("ProcessGradeCrossing $Gc, '" .
126                 $$GradeCrossingData{$Gc}{'State'} .
127                 "' start signals");
128
129             # Start lamps and approach sound effect.
130             Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'},
131                 'start:apr');
132         }
133
134         $$GradeCrossingData{$Gc}{'SigRun'} = 'on';
135         $$GradeCrossingData{$Gc}{'GateDelay'} = $cTime + .5;
136         $$GradeCrossingData{$Gc}{'State'} = 'gateLower';
137         &DisplayMessage("ProcessGradeCrossing $Gc, 'idle' --> " .
138             "$$GradeCrossingData{$Gc}{'State'}'");
139     }
140 }
141
142 # GateLower state code. -----
143 if ($$GradeCrossingData{$Gc}{'State'} eq 'gateLower') {
144
145     # Wait GateDelay. If gates are available, lower them. Then transition
146     # to approach state.
147     if ($$GradeCrossingData{$Gc}{'GateDelay'} < $cTime) { # Delay time done?
148         if ($$GradeCrossingData{$Gc}{'Gate'} ne '') {
149             @gates = split(",", $$GradeCrossingData{$Gc}{'Gate'});
150             foreach my $gate (@gates) {
151                 &DisplayMessage("ProcessGradeCrossing $Gc, '" .
152                     $$GradeCrossingData{$Gc}{'State'} . " state' close " .
153                     "gate: $gate");
154                 &MoveTurnout('Close', $gate, $TurnoutData);
155             }
156         }
157         $$GradeCrossingData{$Gc}{'AprTimer'} = $cTime + 10;
158         $$GradeCrossingData{$Gc}{'State'} = 'approach';
159         &DisplayMessage("ProcessGradeCrossing $Gc, 'gateLower' --> " .
160             "$$GradeCrossingData{$Gc}{'State'}'");
161     }
162 }
163
164 # Approach state code. -----
165 if ($$GradeCrossingData{$Gc}{'State'} eq 'approach') {
166     if ($roadSensor == 1) {
167         $$GradeCrossingData{$Gc}{'RoadTimer'} = $cTime + 1; # Set RoadTimer
168         $$GradeCrossingData{$Gc}{'State'} = 'road';
169         &DisplayMessage("ProcessGradeCrossing $Gc, 'approach' --> " .
170             "$$GradeCrossingData{$Gc}{'State'}'");
171
172         # Change to roadside sound effect. Commented out, need better sound
173         # module.
174         Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'}, 'start:road');
175     }
176     elsif ($$GradeCrossingData{$Gc}{'AprTimer'} < $cTime) { # AprTimer timeout?
177         $$GradeCrossingData{$Gc}{'State'} = 'gateRaise';
178         &DisplayMessage("ProcessGradeCrossing $Gc, 'approach' " .
179             "=> '$$GradeCrossingData{$Gc}{'State'}'");
180     }

```

```

181 }
182
183 # Road state code. -----
184 if ($$GradeCrossingData{$Gc}{'State'} eq 'road') {
185     if ($roadSensor == 1) {
186         $$GradeCrossingData{$Gc}{'RoadTimer'} = $cTime + 1; # Update RoadTimer
187     }
188     else {
189         if ($$GradeCrossingData{$Gc}{'RoadTimer'} < $cTime) { # timeout?
190             $$GradeCrossingData{$Gc}{'State'} = 'gateRaise';
191             &DisplayMessage("ProcessGradeCrossing $Gc, 'road' --> " .
192                 "$$GradeCrossingData{$Gc}{'State'}'");
193
194             # Set back to approach sound effect. Commented out, road sound not
195             # currently used.
196             # Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'},
197             # 'start:apr');
198         }
199     }
200 }
201
202 # GateRaise state code. -----
203 if ($$GradeCrossingData{$Gc}{'State'} eq 'gateRaise') {
204
205     # If no gates, transition to depart state.
206     if ($$GradeCrossingData{$Gc}{'Gate'} eq '') {
207         $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1; # Set DepTimer
208         $$GradeCrossingData{$Gc}{'State'} = 'depart';
209         &DisplayMessage("ProcessGradeCrossing $Gc, 'gateRaise' --> " .
210             "$$GradeCrossingData{$Gc}{'State'}'");
211     }
212     else {
213         if ($$GradeCrossingData{$Gc}{'GateServo'} == 0) {
214             @gates = split(",", $$GradeCrossingData{$Gc}{'Gate'});
215             foreach my $gate (@gates) {
216                 &DisplayMessage("ProcessGradeCrossing $Gc, '" .
217                     $$GradeCrossingData{$Gc}{'State'}' .
218                     " state' open gate: $gate");
219                 &MoveTurnout('Open', $gate, $TurnoutData);
220             }
221             $$GradeCrossingData{$Gc}{'GateServo'} = $gates[0];
222             &DisplayMessage("ProcessGradeCrossing $Gc, '" .
223                 $$GradeCrossingData{$Gc}{'State'}' .
224                 " state' waiting for gate " .
225                 $$GradeCrossingData{$Gc}{'GateServo'} . " to open.");
226         }
227         elsif ($$TurnoutData{$$GradeCrossingData{$Gc}{'GateServo'}}{'Pid'} == 0) {
228             $$GradeCrossingData{$Gc}{'GateServo'} = 0;
229             $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1; # Set DepTimer
230             $$GradeCrossingData{$Gc}{'State'} = 'depart';
231             &DisplayMessage("ProcessGradeCrossing $Gc, 'gateRaise' " .
232                 "--> '$$GradeCrossingData{$Gc}{'State'}'");
233         }
234     }
235 }
236
237 # Depart state code. -----
238 if ($$GradeCrossingData{$Gc}{'State'} eq 'depart') {
239     if ($$GradeCrossingData{$Gc}{'SigRun'} ne 'off') {
240         &DisplayMessage("ProcessGradeCrossing $Gc, '" .

```

```

241         $$GradeCrossingData{$Gc}{'State'} . "' stop signals");
242     Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'}, 'stop');
243     $$GradeCrossingData{$Gc}{'SigRun'} = 'off';
244 }
245
246 # If roadSensor sets, the train backed up. Transition to idle state to
247 # start a new grade crossing cycle.
248 if ($roadSensor == 1) {
249     $$GradeCrossingData{$Gc}{'State'} = 'idle';
250     &DisplayMessage("ProcessGradeCrossing $Gc, 'depart' ==> " .
251         "'$$GradeCrossingData{$Gc}{'State'}'");
252 }
253
254 # Stay in depart state until approach sensors are inactive. This prevents
255 # the start of a new grade crossing cycle by departing train. We also
256 # get here if an approach sensor is blocked by a stopped train.
257 elseif ($aprEastSensor == 1 or $aprWestSensor == 1) {
258     $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1;    # Set DepTimer
259 }
260
261 # Transition to idle state after DepTimer expires.
262 elseif ($$GradeCrossingData{$Gc}{'DepTimer'} < $cTime) {
263     $$GradeCrossingData{$Gc}{'State'} = 'idle';
264     &DisplayMessage("ProcessGradeCrossing $Gc, 'depart' --> " .
265         "'$$GradeCrossingData{$Gc}{'State'}'");
266 }
267 }
268 return 0;
269 }
270
271 # =====
272 # FUNCTION:  GcChildProcess
273 #
274 # DESCRIPTION:
275 #   This routine is launched as a child process during main program startup
276 #   and is used to start and stop grade crossing signal lamp flash operation.
277 #   Since Forks::Super does not allow a child to fork to another child, any
278 #   servo driven gate timing and positioning for the signal must be done by
279 #   the caller.
280 #
281 #   A dedicated GcChildProcess is started for each grade crossing. The returned
282 #   child Pid value is stored in the %GradeCrossingData hash. This Pid value
283 #   is used in the Forks::Super::write_stdin message to send commands to the
284 #   proper GcChildProcess instance.
285 #
286 # CALLING SYNTAX:
287 #   $pid = fork { os_priority => 1, sub => \&GcChildProcess,
288 #       child_fh => "in socket",
289 #       args => [ $Gc, $SignalChildPid, \%SignalData,
290 #           \%GradeCrossingData, \%SensorChip, \%MCP23017 ] };
291 #
292 #   $GradeCrossing      The signal to be processed.
293 #   $SignalChildPid     PID of child signal refresh process.
294 #   $SignalData         Pointer to %SignalData hash.
295 #   $GradeCrossingData  Pointer to the %GradeCrossingData hash.
296 #   $SensorChip         Pointer to the %SensorChip hash.
297 #   $MCP23017          Pointer to the %MCP23017 hash.
298 #
299 #   The SuperForks 'child_fh' functionality is used for communication between
300 #   the parent and child processes. The parent sends a start/stop signal message

```

```

301 # to the child's stdin. The message must be formatted as follows.
302 #
303 # start:apr - Start flashing lamps with bell sound 1.
304 # start:road - Start flashing lamps with bell sound 2.
305 # stop - Stop lamp flash and bell sound.
306 # exit - Terminate GcChildProcess.
307 #
308 # SEND DATA TO CHILD:
309 # Forks::Super::write_stdin($GcChildPid, 'start:apr');
310 # Forks::Super::write_stdin($GcChildPid, 'start:road');
311 # Forks::Super::write_stdin($GcChildPid, 'stop');
312 # Forks::Super::write_stdin($GcChildPid, 'exit');
313 #
314 # RETURNED VALUES:
315 # PID of child process = Success, 0 = Error
316 #
317 # ACCESSED GLOBAL VARIABLES:
318 # $main::ChildName
319 # =====
320 sub GcChildProcess {
321     my($GradeCrossing, $SignalChildPid, $SignalData, $GradeCrossingData,
322        $SensorChip, $MCP23017) = @_;
323     my($x, @buffer, $lampColor, %sndCtrl, $sndSet, $sndClr, $data);
324     my($cmd) = ''; my($lampFlash) = 0;
325
326     $main::ChildName = "GcChildProcess$GradeCrossing";
327     &DisplayMessage("GcChildProcess${GradeCrossing} started.");
328
329     # Setup grade crossing specific working variables.
330     my($signalNbr) = $$GradeCrossingData{$GradeCrossing}{Signal};
331     if ($$GradeCrossingData{$GradeCrossing}{SoundApr} =~
332         m/^(\\d), (GPIO)(.)(\\d)$/) {
333         $sndCtrl{'apr'}{'chip'} = $1;
334         $sndCtrl{'apr'}{'port'} = join("", $2, $3);
335         $sndCtrl{'apr'}{'gpio'} = join("", $2, $3, $4);
336         $sndCtrl{'apr'}{'olat'} = join("", "OLAT", $3);
337         $sndCtrl{'apr'}{'bitSet'} = 1 << $4;
338         $sndCtrl{'apr'}{'bitClr'} = ~$sndCtrl{'apr'}{'bitSet'};
339     }
340     if ($$GradeCrossingData{$GradeCrossing}{SoundRoad} =~
341         m/^(\\d), (GPIO)(.)(\\d)$/) {
342         $sndCtrl{'road'}{'chip'} = $1;
343         $sndCtrl{'road'}{'port'} = join("", $2, $3);
344         $sndCtrl{'road'}{'gpio'} = join("", $2, $3, $4);
345         $sndCtrl{'road'}{'olat'} = join("", "OLAT", $3);
346         $sndCtrl{'road'}{'bitSet'} = 1 << $4;
347         $sndCtrl{'road'}{'bitClr'} = ~$sndCtrl{'road'}{'bitSet'};
348     }
349     &DisplayDebug(1, "GcChildProcess${GradeCrossing}, using " .
350                  "signalNbr: $signalNbr " .
351                  "sndApr: '" . $sndCtrl{'apr'}{'gpio'} . "' " .
352                  "sndRoad: '" . $sndCtrl{'road'}{'gpio'} . "'");
353
354     # Run the main processing loop.
355     while (1) {
356         push(@buffer, <STDIN>);
357         if ($#buffer >= 0) {
358             for ($x = 0; $x <= $#buffer; $x++) {
359                 print "x: $x - '$buffer[$x]' \n";
360             }

```

```

361 # }
362
363 # -----
364 # Check for a new complete message and process if found.
365 if ($buffer[0] =~ m/(start):(apr)/i or $buffer[0] =~ m/(start):(road)/i or
366     $buffer[0] =~ m/(stop)/i or $buffer[0] =~ m/(exit)/i) {
367     $cmd = lc $1;
368     $sndSet = lc $2;
369
370     if ($sndSet eq 'apr') {
371         $sndClr = 'road';
372     }
373     elseif ($sndSet eq 'road') {
374         $sndClr = 'apr';
375     }
376     else {
377         $sndClr = '';
378     }
379     splice(@buffer, 0, 1); # Remove processed record.
380     &DisplayDebug(3, "GcChildProcess${GradeCrossing}, cmd: " .
381 #         "'$cmd'     sndSet: '$sndSet'");
382 }
383
384 # -----
385 # Process new command, if any.
386 if ($cmd ne "") {
387     if ($cmd eq "start") {
388         if ($lampFlash == 0) {
389             $lampColor = 'Red';
390             $lampFlash = 1;
391         }
392
393         # Clear opposite sound activation control bit
394         if ($sndClr ne '') {
395             &ClearControlBit($sndClr, \%sndCtrl, $SensorChip, $MCP23017);
396         }
397
398         # Set new sound activation control bit.
399         if ($sndSet ne '' and exists($sndCtrl{$sndSet}{chip})) {
400             $data = $$SensorChip{ $sndCtrl{$sndSet}{chip} }{'Obj'}
401                 ->read_byte($MCP23017{ $sndCtrl{$sndSet}{port} });
402             $data = $data | $sndCtrl{$sndSet}{bitSet};
403             $$SensorChip{ $sndCtrl{$sndSet}{chip} }{'Obj'}
404                 ->write_byte($data, $MCP23017{ $sndCtrl{$sndSet}{olat} });
405         }
406     }
407     elseif ($cmd eq "stop" and $lampFlash == 1) {
408         $lampColor = 'Off';
409     }
410     elseif ($cmd eq "exit") {
411         &DisplayMessage("GcChildProcess${GradeCrossing} " .
412             "commanded to exit.");
413
414         # Turn off signal lamps.
415         &SetSignalColor($signalNbr, 'Off', $SignalChildPid,
416             $SignalData, '');
417
418         # Clear sound activation control bits
419         &ClearControlBit('apr', \%sndCtrl, $SensorChip, $MCP23017);
420         &ClearControlBit('road', \%sndCtrl, $SensorChip, $MCP23017);

```

```

421         last;          # Break out of while loop and exit.
422     }
423     $cmd = "";          # Remove processed command.
424 }
425
426 # -----
427 # Change lamp state.
428 if ($lampFlash == 1) {
429     if ($lampColor eq 'Off') {
430         $lampFlash = 0;
431
432         # Clear sound activation control bits
433         &ClearControlBit('apr', \%sndCtrl, $SensorChip, $MCP23017);
434         &ClearControlBit('road', \%sndCtrl, $SensorChip, $MCP23017);
435     }
436     elsif ($lampColor eq 'Red') {
437         $lampColor = 'Grn';
438     }
439     else {
440         $lampColor = 'Red';
441     }
442
443     if (&SetSignalColor($signalNmbr, $lampColor, $SignalChildPid,
444         $SignalData, '')) {
445         &DisplayError("GcChildProcess${GradeCrossing}, " .
446             "SetSignalColor returned error.");
447     }
448 }
449 sleep 0.8;          # Sets signal flash rate.
450 }
451 &DisplayMessage("GcChildProcess${GradeCrossing} terminated.");
452 exit(0);
453 }
454
455 # =====
456 # FUNCTION:  ClearControlBit
457 #
458 # DESCRIPTION:
459 #   This routine is used by GcChildProcess for clearing the specified sound
460 #   activation control bit.
461 #
462 # CALLING SYNTAX:
463 #   $result = &ClearControlBit($Snd, $sndCtrlHash, $SensorChip);
464 #
465 # ARGUMENTS:
466 #   $Snd          Hash index, 'apr' or 'road'.
467 #   $sndCtrlHash  Pointer to GcChildProcess sndCtrl hash.
468 #   $SensorChip   Pointer to the %SensorChip hash.
469 #   $MCP23017     Pointer to $MCP23017 hash.
470 #
471 # RETURNED VALUES:
472 #   0 = Success,  1 = Error.
473 #
474 # ACCESSED GLOBAL VARIABLES:
475 #   None.
476 # =====
477 sub ClearControlBit {
478     my($Snd, $sndCtrlHash, $SensorChip, $MCP23017) = @_;
479     my($data);
480

```



```

481     if (exists($$sndCtrlHash{$Snd}{'chip'})) {
482         $data = $$SensorChip{ $$sndCtrlHash{$Snd}{'chip'} }{'Obj'}
483         ->read_byte($$MCP23017{ $$sndCtrlHash{$Snd}{'port'} });
484         $data = $data & $$sndCtrlHash{$Snd}{'bitClr'};
485         $$SensorChip{ $$sndCtrlHash{$Snd}{'chip'} }{'Obj'}
486         ->write_byte($data, $$MCP23017{ $$sndCtrlHash{$Snd}{'olat'} });
487     }
488     return 0;
489 }
490
491 # =====
492 # FUNCTION:  TestGradeCrossing
493 #
494 # DESCRIPTION:
495 #     This routine cycles the specified grade crossing signal ranges.
496 #
497 # CALLING SYNTAX:
498 #     $result = &TestGradeCrossing($Range, \%GradeCrossingData, \%TurnoutData);
499 #
500 # ARGUMENTS:
501 #     $Range          Signal number or range to use.
502 #     $GradeCrossingData  Pointer to GradeCrossingData hash.
503 #     $TurnoutData      Pointer to %TurnoutData hash.
504 #
505 # RETURNED VALUES:
506 #     0 = Success,  1 = Error.
507 #
508 # ACCESSED GLOBAL VARIABLES:
509 #     $main::MainRun
510 # =====
511 sub TestGradeCrossing {
512
513     my($Range, $GradeCrossingData, $TurnoutData) = @_;
514     my($result, @gates, $gate);
515     my(@gcList) = split(",", $Range);
516
517     &DisplayDebug(2, "TestGradeCrossing, Entry ...   Range: '$Range'" .
518                  "   gcList: @gcList");
519
520     while ($main::MainRun) {
521
522         # Start approach signal.
523         foreach my $gc (@gcList) {
524             $gc = "0${gc}" if (length($gc) == 1);
525             if (exists $$GradeCrossingData{$gc}) {
526                 &DisplayMessage("TestGradeCrossing, start:apr grade " .
527                               "crossing $gc");
528                 Forks::Super::write_stdin($$GradeCrossingData{$gc}{ 'Pid' },
529                                           'start:apr');
530                 sleep 1;                # Time for realistic lamp start.
531             }
532             else {
533                 &DisplayError("TestGradeCrossing, invalid grade " .
534                              "crossing: $gc");
535                 return 1;
536             }
537
538             # Lower gates if grade crossing is so equipt.
539             @gates = split(",", $$GradeCrossingData{$gc}{ 'Gate' });
540             foreach my $gate (@gates) {

```

```

541     &DisplayDebug(1, "TestGradeCrossing, Close gate: $gate");
542     $result = &MoveTurnout('Close', $gate, $TurnoutData);
543     if ($result == 1) {
544         &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
545             "returned error.");
546     }
547     elsif ($result == 2) {
548         &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
549             "returned already in position.");
550     }
551 }
552 Forks::Super::pause 2;
553 }
554 Forks::Super::pause 4;
555
556 # Change to 'road' grade crossing sound. Commented out, need better sound module.
557 foreach my $gc (@gcList) {
558     $gc = "0${gc}" if (length($gc) == 1);
559     # &DisplayMessage("TestGradeCrossing, start:road grade crossing $gc");
560     # Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'start:road');
561 }
562 Forks::Super::pause 4;
563
564 # Stop signal.
565 foreach my $gc (@gcList) {
566     $gc = "0${gc}" if (length($gc) == 1);
567     &DisplayMessage("TestGradeCrossing, stop grade crossing $gc");
568     @gates = split(",", $$GradeCrossingData{$gc}{'Gate'});
569     foreach my $gate (@gates) {
570         $result = &MoveTurnout('Open', $gate, $TurnoutData);
571         if ($result == 1) {
572             &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
573                 "returned error.");
574         }
575         elsif ($result == 2) {
576             &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
577                 "returned already in position.");
578         }
579     }
580
581     # If gates for this crossing, wait for gate open to complete before
582     # stopping lamp flash.
583     if ($#gates >= 0) {
584         &DisplayDebug(1, "TestGradeCrossing, waiting for gate " .
585             "$gates[0] move to complete.");
586         while ($$TurnoutData{$gates[0]}{'Pid'} > 0) {
587             sleep 0.5;
588         }
589     }
590     Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'stop');
591     Forks::Super::pause 2;
592 }
593 Forks::Super::pause 4;
594 }
595 return 0;
596 }
597
598 return 1;
599

```