```perl
# ============================================================================
# FILE: DnB_GradeCrossing.pm                                      7/06/2020
#
# SERVICES:  DnB GRADE CROSSING FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides grade crossing related functions used by the
#    DnB model railroad control program.
#
# PERL VERSION: 5.24.1
#
# ============================================================================
use strict;
# ----------------------------------------------------------------------------
# Package Declaration
# ----------------------------------------------------------------------------
package DnB_GradeCrossing;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   ProcessGradeCrossing
   GcChildProcess
   TestGradeCrossing
);

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

# ============================================================================
# FUNCTION:  ProcessGradeCrossing
#
# DESCRIPTION:
#    This routine is used to process the specified grade crossing. It is called
#    once an iteration by the main program loop. State data that is used for
#    grade crossing control is persisted in the %GradeCrossingData hash. Each
#    grade crossing is in one of the following states; 'idle', 'gateLower',
#    'approach', 'road', 'gateRaise' or 'depart'. %GradeCrossingData values,
#    sensor bits, and code within this routine, transition the signal through
#    these states. Operation is as follows.
#
#    1. Configuration and initializations set in %GradeCrossingData hash.
#
#    2. In 'idle' state, a train approaching the grade crossing is detected by
#    sensors 'AprEast', 'AprWest', or 'Road'. This causes the signals to begin
#    flashing. 'SigRun' is set to 'on'. 'GateDelay' is set and the state
#    transitions to 'gateLower'.
#
#    3. In 'gateLower' state, GateDelay is performed and the 'AprTimer' is set.
#    If gates are available, they are lowered. Then the state transitions to
#    'approach'. The GateDelay value is used to better simulate proto-typical
#    signal operation.
#
#    4. In 'approach' state, if 'road' state is not achieved before 'AprTimer'
#    expires, the code transitions to the 'gateRaise' state. This could occur if
#    the train stops or backs away before reaching the 'Road' sensor. An active
```

```
#    'Road' sensor causes transition to the 'road' state.
#
#    5. In 'road' state, a short timeout is set into 'RoadTimer'. Additional
#    'Road' sensor activity reloads this timer. This maintains 'road' state
#    while the train occupies the grade crossing. When no further 'Road' sensor
#    activity is reported, 'RoadTimer' will expire. The state transitions to
#    'gateRaise'.
#
#    6. In 'gateRaise' state, if grade crossing does not have gates, 'DepTimer'
#    is set and the state transitions to 'depart'. Otherwise, the gates are
#    raised. Once completed (servo pid == 0), 'DepTimer' is set and the state
#    transitions to 'depart'.
#
#    7. In the 'depart' state, the signal lamp flashing is stopped and 'SigRun'
#    is set to 'off'. Outbound train 'AprEast' or 'AprWest' sensor activity
#    restarts the 'DepTimer' maintaining the 'depart' state. Once the last car
#    of the outbound train is past the 'AprEast' or 'AprWest' sensor, the
#    'DepTimer' expires and the state transitions to 'idle'.
#
#    If the train backs up, 'Road' sensor activity will transition the state to
#    'idle'. From 'idle', the active 'Road' sensor will start a new signaling
#    cycle.
#
# CALLING SYNTAX:
#    $result = &ProcessGradeCrossing($gc, \%GradeCrossingData, \%SensorBit,
#              \%TurnoutData, \%MCP23017, \%SensorState);
#
# ARGUMENTS:
#    $Gc                 Index to data in %GradeCrossingData.
#    $GradeCrossingData  Pointer to %GradeCrossingData hash.
#    $SensorBit          Pointer to %SensorBit hash.
#    $TurnoutData        Pointer to %TurnoutData hash. (needed for gates and sound)
#    $MCP23017           Pointer to %MCP23017 hash. (GPIO definitions)
#    $SensorState        Pointer to %SensorState hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ProcessGradeCrossing {
   my($Gc, $GradeCrossingData, $SensorBit, $TurnoutData, $MCP23017, $SensorState) = @_;
   my(@gates);

# Isolate the current grade crossing sensor bit values and get the current time.

   my($aprEastSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'AprEast'},
                                      $SensorBit, $SensorState);
   my($roadSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'Road'}, $SensorBit,
                                   $SensorState);
   my($aprWestSensor) = &GetSensorBit($$GradeCrossingData{$Gc}{'AprWest'},
                                      $SensorBit, $SensorState);
   my($cTime) = time;

   &DisplayDebug(2, "ProcessGradeCrossing $Gc, State: " .
             "$$GradeCrossingData{$Gc}{'State'}   aprEastSensor: $aprEastSensor" .
             "   roadSensor: $roadSensor   aprWestSensor: $aprWestSensor   " .
             "cTime: $cTime");

```

```
# Idle state code. ------------------------------------------------------------
   if ($$GradeCrossingData{$Gc}{'State'} eq 'idle') {
      if ($roadSensor == 1 or $aprEastSensor == 1 or $aprWestSensor == 1) {
         if ($$GradeCrossingData{$Gc}{'SigRun'} ne 'on') {
            &DisplayMessage("ProcessGradeCrossing $Gc, '" .
                     $$GradeCrossingData{$Gc}{'State'} .
                     "' start signals");

            # Start lamps and approach sound effect.
            Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'},
               'start:apr');
         }

         $$GradeCrossingData{$Gc}{'SigRun'} = 'on';
         $$GradeCrossingData{$Gc}{'GateDelay'} = $cTime + .5;
         $$GradeCrossingData{$Gc}{'State'} = 'gateLower';
         &DisplayMessage("ProcessGradeCrossing $Gc, 'idle' --> " .
                         "'$$GradeCrossingData{$Gc}{'State'}'.");
      }
   }

# GateLower state code. -------------------------------------------------------
   if ($$GradeCrossingData{$Gc}{'State'} eq 'gateLower') {

      # Wait GateDelay. If gates are available, lower them. Then transition
      # to approach state.
      if ($$GradeCrossingData{$Gc}{'GateDelay'} < $cTime) { # Delay time done?
         if ($$GradeCrossingData{$Gc}{'Gate'} ne '') {
            @gates = split(",", $$GradeCrossingData{$Gc}{'Gate'});
            foreach my $gate (@gates) {
               &DisplayMessage("ProcessGradeCrossing $Gc, '" .
                         $$GradeCrossingData{$Gc}{'State'} . " state' close " .
                         "gate: $gate");
               &MoveTurnout('Close', $gate, $TurnoutData);
            }
         }
         $$GradeCrossingData{$Gc}{'AprTimer'} = $cTime + 10;
         $$GradeCrossingData{$Gc}{'State'} = 'approach';
         &DisplayMessage("ProcessGradeCrossing $Gc, 'gateLower' --> " .
                         "'$$GradeCrossingData{$Gc}{'State'}'.");
      }
   }

# Approach state code. --------------------------------------------------------
   if ($$GradeCrossingData{$Gc}{'State'} eq 'approach') {
      if ($roadSensor == 1) {
         $$GradeCrossingData{$Gc}{'RoadTimer'} = $cTime + 1;    # Set RoadTimer
         $$GradeCrossingData{$Gc}{'State'} = 'road';
         &DisplayMessage("ProcessGradeCrossing $Gc, 'approach' --> " .
                         "'$$GradeCrossingData{$Gc}{'State'}'.");

         # Change to roadside sound effect. Commented out, need better sound
         # module.
#        Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'}, 'start:road');
      }
      elsif ($$GradeCrossingData{$Gc}{'AprTimer'} < $cTime) { # AprTimer timeout?
         $$GradeCrossingData{$Gc}{'State'} = 'gateRaise';
         &DisplayMessage("ProcessGradeCrossing $Gc, 'approach' " .
                         "==> '$$GradeCrossingData{$Gc}{'State'}'.");
      }
```

```
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, '" .
```

```
                        $$GradeCrossingData{$Gc}{'State'} .
                        "' stop signals");
         Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'}, 'stop');
         $$GradeCrossingData{$Gc}{'SigRun'} = 'off';
      }

      # If roadSensor sets, the train backed up. Transition to idle state to
      # start a new grade crossing cycle.
      if ($roadSensor == 1) {
         $$GradeCrossingData{$Gc}{'State'} = 'idle';
         &DisplayMessage("ProcessGradeCrossing $Gc, 'depart' ==> " .
                        "'$$GradeCrossingData{$Gc}{'State'}'.");
      }

      # Stay in depart state until approach sensors are inactive. This prevents
      # the start of a new grade crossing cycle by departing train. We also
      # get here if an approach sensor is blocked by a stopped train.
      elsif ($aprEastSensor == 1 or $aprWestSensor == 1) {
         $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1;   # Set DepTimer
      }

      # Transition to idle state after DepTimer expires.
      elsif ($$GradeCrossingData{$Gc}{'DepTimer'} < $cTime) {
         $$GradeCrossingData{$Gc}{'State'} = 'idle';
         &DisplayMessage("ProcessGradeCrossing $Gc, 'depart' --> " .
                        "'$$GradeCrossingData{$Gc}{'State'}'.");
      }
   }
   return 0;
}

# ============================================================================
# FUNCTION:  GcChildProcess
#
# DESCRIPTION:
#    This routine is launched as a child process during main program startup
#    and is used to start and stop grade crossing signal lamp flash operation.
#    Since Forks::Super does not allow a child to fork to another child, any
#    servo driven gate timing and positioning for the signal must be done by
#    the caller.
#
#    A dedicated GcChildProcess is started for each grade crossing. The returned
#    child Pid value is stored in the %GradeCrossingData hash. This Pid value
#    is used in the Forks::Super::write_stdin message to send commands to the
#    proper GcChildProcess instance.
#
# CALLING SYNTAX:
#    $pid = fork { os_priority => 1, sub => \&GcChildProcess,
#                  child_fh => "in socket",
#                  args => [ $Gc, $SignalChildPid, \%SignalData,
#                            \%GradeCrossingData, \%SensorChip, \%MCP23017 ] };
#
#       $GradeCrossing       The signal to be processed.
#       $SignalChildPid      PID of child signal refresh process.
#       $SignalData          Pointer to %SignalData hash.
#       $GradeCrossingData   Pointer to the %GradeCrossingData hash.
#       $SensorChip          Pointer to the %SensorChip hash.
#       $MCP23017            Pointer to the %MCP23017 hash.
#
#    The SuperForks 'child_fh' functionality is used for communication between
```

```
#    the parent and child processes. The parent sends a start/stop signal message
#    to the child's stdin. The message must be formatted as follows.
#
#       start:apr  - Start flashing lamps with bell sound 1.
#       start:road - Start flashing lamps with bell sound 2.
#       stop       - Stop lamp flash and bell sound.
#       exit       - Terminate GcChildProcess.
#
# SEND DATA TO CHILD:
#    Forks::Super::write_stdin($GcChildPid, 'start:apr'));
#    Forks::Super::write_stdin($GcChildPid, 'start:road'));
#    Forks::Super::write_stdin($GcChildPid, 'stop'));
#    Forks::Super::write_stdin($GcChildPid, 'exit'));
#
# RETURNED VALUES:
#    PID of child process = Success, 0 = Error
#
# ACCESSED GLOBAL VARIABLES:
#    $main::ChildName
# ============================================================================
sub GcChildProcess {
   my($GradeCrossing, $SignalChildPid, $SignalData, $GradeCrossingData,
      $SensorChip, $MCP23017) = @_;
   my($x, @buffer, $lampColor, %sndCtrl, $sndSet, $sndClr, $data);
   my($cmd) = '';  my($lampFlash) = 0;

   $main::ChildName = "GcChildProcess$GradeCrossing";
   &DisplayMessage("GcChildProcess${GradeCrossing} started.");

# Setup grade crossing specific working variables.
   my($signalNmbr) = $$GradeCrossingData{$GradeCrossing}{Signal};
   if ($$GradeCrossingData{$GradeCrossing}{'SoundApr'} =~
         m/^(\d),(GPIO)(.)(\d)$/) {
      $sndCtrl{'apr'}{'chip'} = $1;
      $sndCtrl{'apr'}{'port'} = join("", $2, $3);
      $sndCtrl{'apr'}{'gpio'} = join("", $2, $3, $4);
      $sndCtrl{'apr'}{'olat'} = join("", "OLAT", $3);
      $sndCtrl{'apr'}{'bitSet'} = 1 << $4;
      $sndCtrl{'apr'}{'bitClr'} = ~$sndCtrl{'apr'}{'bitSet'};
   }
   if ($$GradeCrossingData{$GradeCrossing}{'SoundRoad'} =~
         m/^(\d),(GPIO)(.)(\d)$/) {
      $sndCtrl{'road'}{'chip'} = $1;
      $sndCtrl{'road'}{'port'} = join("", $2, $3);
      $sndCtrl{'road'}{'gpio'} = join("", $2, $3, $4);
      $sndCtrl{'road'}{'olat'} = join("", "OLAT", $3);
      $sndCtrl{'road'}{'bitSet'} = 1 << $4;
      $sndCtrl{'road'}{'bitClr'} = ~$sndCtrl{'road'}{'bitSet'};
   }
   &DisplayDebug(1, "GcChildProcess${GradeCrossing}, using " .
                    "signalNmbr: $signalNmbr " .
                    "sndApr: '" . $sndCtrl{'apr'}{'gpio'} . "' " .
                    "sndRoad: '" . $sndCtrl{'road'}{'gpio'} . "'");

# Run the main processing loop.
   while (1) {
      push(@buffer, <STDIN>);
#     if ($#buffer >= 0) {
#        for ($x = 0; $x <= $#buffer; $x++) {
#           print "x: $x - '$buffer[$x]' \n";
```

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

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

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

```
541            @gates = split(",", $$GradeCrossingData{$gc}{'Gate'});
542            foreach my $gate (@gates) {
543               &DisplayDebug(1, "TestGradeCrossing, Close gate: $gate");
544               $result = &MoveTurnout('Close', $gate, $TurnoutData);
545               if ($result == 1) {
546                  &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
547                                   "returned error.");
548               }
549               elsif ($result == 2) {
550                  &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
551                                   "returned already in position.");
552               }
553            }
554            Forks::Super::pause 2;
555         }
556         Forks::Super::pause 4;
557
558   # Change to 'road' grade crossing sound. Commented out, need better sound module.
559         foreach my $gc (@gcList) {
560            $gc = "0${gc}" if (length($gc) == 1);
561   #         &DisplayMessage("TestGradeCrossing, start:road grade " .
562   #                         "crossing $gc");
563   #         Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'start:road');
564         }
565          Forks::Super::pause 4;
566
567   # Stop signal.
568         foreach my $gc (@gcList) {
569            $gc = "0${gc}" if (length($gc) == 1);
570            &DisplayMessage("TestGradeCrossing, stop grade crossing $gc");
571            @gates = split(",", $$GradeCrossingData{$gc}{'Gate'});
572            foreach my $gate (@gates) {
573               $result = &MoveTurnout('Open', $gate, $TurnoutData);
574               if ($result == 1) {
575                  &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
576                                   "returned error.");
577               }
578               elsif ($result == 2) {
579                  &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
580                                   "returned already in position.");
581               }
582            }
583
584            # If gates for this crossing, wait for gate open to complete before
585            # stopping lamp flash.
586            if ($#gates >= 0) {
587               &DisplayDebug(1, "TestGradeCrossing, waiting for gate " .
588                                "$gates[0] move to complete.");
589               while ($$TurnoutData{$gates[0]}{Pid} > 0) {
590                  sleep 0.5;
591               }
592            }
593            Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'stop');
594            Forks::Super::pause 2;
595         }
596         Forks::Super::pause 4;
597      }
598      return 0;
599   }
600
```

```
601   return 1;
602
```