```perl
# ============================================================================
# FILE: DnB_GradeCrossing.pm                                        8/06/2020
#
# 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'}'.");
      }
   }

# Road state code. --------------------------------------------------------------
   if ($$GradeCrossingData{$Gc}{'State'} eq 'road') {
      if ($roadSensor == 1) {
         $$GradeCrossingData{$Gc}{'RoadTimer'} = $cTime + 1;  # Update RoadTimer
      }
      else {
         if ($$GradeCrossingData{$Gc}{'RoadTimer'} < $cTime) { # timeout?
            $$GradeCrossingData{$Gc}{'State'} = 'gateRaise';
            &DisplayMessage("ProcessGradeCrossing $Gc, 'road' --> " .
                            "'$$GradeCrossingData{$Gc}{'State'}'.");

            # Set back to approach sound effect. Commented out, road sound not
            # currently used.
            # Forks::Super::write_stdin($$GradeCrossingData{$Gc}{'Pid'},
            # 'start:apr');
         }
      }
   }

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

      # If no gates, transition to depart state.
      if ($$GradeCrossingData{$Gc}{'Gate'} eq '') {
         $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1;    # Set DepTimer
         $$GradeCrossingData{$Gc}{'State'} = 'depart';
         &DisplayMessage("ProcessGradeCrossing $Gc, 'gateRaise' --> " .
                         "'$$GradeCrossingData{$Gc}{'State'}'.");
      }
      else {
         if ($$GradeCrossingData{$Gc}{'GateServo'} == 0) {
            @gates = split(",", $$GradeCrossingData{$Gc}{'Gate'});
            foreach my $gate (@gates) {
               &DisplayMessage("ProcessGradeCrossing $Gc, '" .
                               $$GradeCrossingData{$Gc}{'State'} .
                               "' state' open gate: $gate");
               &MoveTurnout('Open', $gate, $TurnoutData);
            }
            $$GradeCrossingData{$Gc}{'GateServo'} = $gates[0];
            &DisplayMessage("ProcessGradeCrossing $Gc, '" .
                            $$GradeCrossingData{$Gc}{'State'} .
                            "' state' waiting for gate " .
                            $$GradeCrossingData{$Gc}{'GateServo'} . " to open.");
         }
         elsif ($$TurnoutData{$$GradeCrossingData{$Gc}{'GateServo'}}{Pid} == 0) {
            $$GradeCrossingData{$Gc}{'GateServo'} = 0;
            $$GradeCrossingData{$Gc}{'DepTimer'} = $cTime + 1;  # Set DepTimer
            $$GradeCrossingData{$Gc}{'State'} = 'depart';
            &DisplayMessage("ProcessGradeCrossing $Gc, 'gateRaise' " .
                            "--> '$$GradeCrossingData{$Gc}{'State'}'.");
         }
      }
   }

# Depart state code. ------------------------------------------------------------
   if ($$GradeCrossingData{$Gc}{'State'} eq 'depart') {
      if ($$GradeCrossingData{$Gc}{'SigRun'} ne 'off') {
         &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";
#         }
#      }

      # ----------
      # Check for a new complete message and process if found.
      if ($buffer[0] =~ m/(start):(apr)/i or $buffer[0] =~ m/(start):(road)/i or
          $buffer[0] =~ m/(stop)/i or $buffer[0] =~ m/(exit)/i) {
         $cmd = lc $1;
         $sndSet = lc $2;

         if ($sndSet eq 'apr') {
            $sndClr = 'road';
         }
         elsif ($sndSet eq 'road') {
            $sndClr = 'apr';
         }
         else {
            $sndClr = '';
         }
         splice(@buffer, 0, 1);         # Remove processed record.
#         &DisplayDebug(3, "GcChildProcess${GradeCrossing}, cmd: " .
#                           "'$cmd'   sndSet: '$sndSet'");
      }

      # ----------
      # Process new command, if any.
      if ($cmd ne "") {
         if ($cmd eq "start") {
            if ($lampFlash == 0) {
               $lampColor = 'Red';
               $lampFlash = 1;
            }

            # Clear opposite sound activation control bit
            if ($sndClr ne '') {
               &ClearControlBit($sndClr, \%sndCtrl, $SensorChip, $MCP23017);
            }

            # Set new sound activation control bit.
            if ($sndSet ne '' and exists($sndCtrl{$sndSet}{'chip'})) {
               $data = $$SensorChip{ $sndCtrl{$sndSet}{'chip'} }{'Obj'}
                       ->read_byte($$MCP23017{ $sndCtrl{$sndSet}{'port'} });
               $data = $data | $sndCtrl{$sndSet}{'bitSet'};
               $$SensorChip{ $sndCtrl{$sndSet}{'chip'} }{'Obj'}
               ->write_byte($data, $$MCP23017{ $sndCtrl{$sndSet}{'olat'} });
            }
         }
         elsif ($cmd eq "stop" and $lampFlash == 1) {
            $lampColor = 'Off';
         }
         elsif ($cmd eq "exit") {
            &DisplayMessage("GcChildProcess${GradeCrossing} " .
                                      "commanded to exit.");

            # Turn off signal lamps.
            &SetSignalColor($signalNmbr, 'Off', $SignalChildPid,
                            $SignalData, '');

            # Clear sound activation control bits
            &ClearControlBit('apr', \%sndCtrl, $SensorChip, $MCP23017);
            &ClearControlBit('road', \%sndCtrl, $SensorChip, $MCP23017);
            last;       # Break out of while loop and exit.
         }
         $cmd = "";                     # Remove processed command.
      }

      # ----------
      # Change lamp state.
      if ($lampFlash == 1) {
         if ($lampColor eq 'Off') {
            $lampFlash = 0;

            # Clear sound activation control bits
            &ClearControlBit('apr', \%sndCtrl, $SensorChip, $MCP23017);
            &ClearControlBit('road', \%sndCtrl, $SensorChip, $MCP23017);
         }
         elsif ($lampColor eq 'Red') {
            $lampColor = 'Grn';
         }
         else {
            $lampColor = 'Red';
         }

         if (&SetSignalColor($signalNmbr, $lampColor, $SignalChildPid,
                             $SignalData, '')) {
            &DisplayError("GcChildProcess${GradeCrossing}, " .
                          "SetSignalColor returned error.");
         }
      }
      sleep 0.8;            # Sets signal flash rate.
   }
   &DisplayMessage("GcChildProcess${GradeCrossing} terminated.");
   exit(0);
}

# =============================================================================
# FUNCTION:  ClearControlBit
#
# DESCRIPTION:
#    This routine is used by GcChildProcess for clearing the specified sound
#    activation control bit.
#
# CALLING SYNTAX:
#    $result = &ClearControlBit($Snd, $sndCtrlHash, $SensorChip);
#
# ARGUMENTS:
#    $Snd                Hash index, 'apr' or 'road'.
#    $sndCtrlHash        Pointer to GcChildProcess sndCtrl hash.
#    $SensorChip         Pointer to the %SensorChip hash.
#    $MCP23017           Pointer to $MCP23017 hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# =============================================================================
sub ClearControlBit {
   my($Snd, $sndCtrlHash, $SensorChip, $MCP23017) = @_;
   my($data);

```

```perl
   if (exists($$sndCtrlHash{$Snd}{'chip'})) {
      $data = $$SensorChip{ $$sndCtrlHash{$Snd}{'chip'} }{'Obj'}
              ->read_byte($$MCP23017{ $$sndCtrlHash{$Snd}{'port'} });
      $data = $data & $$sndCtrlHash{$Snd}{'bitClr'};
      $$SensorChip{ $$sndCtrlHash{$Snd}{'chip'} }{'Obj'}
         ->write_byte($data, $$MCP23017{ $$sndCtrlHash{$Snd}{'olat'} });
   }
   return 0;
}

# =============================================================================
# FUNCTION:  TestGradeCrossing
#
# DESCRIPTION:
#    This routine cycles the specified grade crossing signal ranges.
#
# CALLING SYNTAX:
#    $result = &TestGradeCrossing($Range, \%GradeCrossingData, \%TurnoutData);
#
# ARGUMENTS:
#    $Range               Signal number or range to use.
#    $GradeCrossingData   Pointer to GradeCrossingData hash.
#    $TurnoutData         Pointer to %TurnoutData hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::MainRun
# =============================================================================
sub TestGradeCrossing {

   my($Range, $GradeCrossingData, $TurnoutData) = @_;
   my($result, @gates, $gate);
   my(@gcList) = split(",", $Range);

   &DisplayDebug(2, "TestGradeCrossing, Entry ...   Range: '$Range'" .
                    "   gcList: @gcList");

   while ($main::MainRun) {

   # Start approach signal.
      foreach my $gc (@gcList) {
         $gc = "0${gc}" if (length($gc) == 1);
         if (exists $$GradeCrossingData{$gc}) {
            &DisplayMessage("TestGradeCrossing, start:apr grade " .
                            "crossing $gc");
            Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'},
                                      'start:apr');
            sleep 1;              # Time for realistic lamp start.
         }
         else {
            &DisplayError("TestGradeCrossing, invalid grade " .
                          "crossing: $gc");
            return 1;
         }

   # Lower gates if grade crossing is so equipt.
         @gates = split(",", $$GradeCrossingData{$gc}{'Gate'});
         foreach my $gate (@gates) {
            &DisplayDebug(1, "TestGradeCrossing, Close gate: $gate");
            $result = &MoveTurnout('Close', $gate, $TurnoutData);
            if ($result == 1) {
               &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
                                "returned error.");
            }
            elsif ($result == 2) {
               &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
                                "returned already in position.");
            }
         }
         Forks::Super::pause 2;
      }
      Forks::Super::pause 4;

# Change to 'road' grade crossing sound. Commented out, need better sound module.
      foreach my $gc (@gcList) {
         $gc = "0${gc}" if (length($gc) == 1);
#        &DisplayMessage("TestGradeCrossing, start:road grade crossing $gc");
#        Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'start:road');
      }
       Forks::Super::pause 4;

# Stop signal.
      foreach my $gc (@gcList) {
         $gc = "0${gc}" if (length($gc) == 1);
         &DisplayMessage("TestGradeCrossing, stop grade crossing $gc");
         @gates = split(",", $$GradeCrossingData{$gc}{'Gate'});
         foreach my $gate (@gates) {
            $result = &MoveTurnout('Open', $gate, $TurnoutData);
            if ($result == 1) {
               &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
                                "returned error.");
            }
            elsif ($result == 2) {
               &DisplayDebug(1, "TestGradeCrossing, gate: $gate " .
                                "returned already in position.");
            }
         }

         # If gates for this crossing, wait for gate open to complete before
         # stopping lamp flash.
         if ($#gates >= 0) {
            &DisplayDebug(1, "TestGradeCrossing, waiting for gate " .
                             "$gates[0] move to complete.");
            while ($$TurnoutData{$gates[0]}{Pid} > 0) {
               sleep 0.5;
            }
         }
         Forks::Super::write_stdin($$GradeCrossingData{$gc}{'Pid'}, 'stop');
         Forks::Super::pause 2;
      }
      Forks::Super::pause 4;
   }
   return 0;
}

return 1;
```