```
# ============================================================================
# FILE: DnB_Turnout.pm                                              7/07/2020
#
# SERVICES:  DnB TURNOUT FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides turnout related functions used by the DnB model
#    railroad control program.
#
# PERL VERSION: 5.24.1
#
# ============================================================================
use strict;
# ----------------------------------------------------------------------------
# Package Declaration
# ----------------------------------------------------------------------------
package DnB_Turnout;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   I2C_InitServoDriver
   ProcessTurnoutFile
   InitTurnouts
   MoveTurnout
   SetTurnoutPosition
   TestTurnouts
);

use DnB_Message;
use Forks::Super;
use POSIX 'WNOHANG';
use Time::HiRes qw(sleep);

# ============================================================================
# FUNCTION:  I2C_InitServoDriver
#
# DESCRIPTION:
#    This routine initializes the turnout servo I2C driver boards on the DnB
#    model railroad. It sets parameters that are common to all servo ports. The
#    Adafruit 16 Channel Servo Driver utilizes the PCA9685 chip. The pre_scale
#    calculation is from the PCA9685 documentation.
#
#    Initialization sequence.
#       1. Get current ModeReg1.
#       2. Put PCA9685 into sleep mode.
#       3. Set servo refresh rate.
#       4. Normal mode + register auto increment.
#       5. Put PCA9685 into normal mode.
#
# CALLING SYNTAX:
#    $result = &I2C_InitServoDriver($BoardNmbr, $I2C_Address);
#
# ARGUMENTS:
#    $BoardNmbr      Drive board number being initialized.
#    $I2C_Address    I2C Address
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
```

```
# ACCESSED GLOBAL VARIABLES:
#    None.
# ==========================================================================
sub I2C_InitServoDriver {

   my($BoardNmbr, $I2C_Address) = @_;
   my($result, $driver, $mode_data);

   my($minAddr, $maxAddr) = (0x40, 0x7F);  # AdaFruit 16 Channel PWM board range.
   my(%PCA9685) = ('ModeReg1' => 0x00, 'ModeReg2' => 0x01, 'AllLedOffH' => 0xFD,
                   'PreScale' => 0xFE);
   my($normal_mode) = 0xEF;   my($sleep_mode) = 0x10;   my($auto_inc) = 0xA1;

   my($freq) = 105;   # Refresh rate; 105 = 300-900 SG90 min/max position.

   my($pre_scale) = int((25000000.0 / (4096 * $freq)) - 1);

   &DisplayDebug(2, "I2C_InitServoDriver, BoardNmbr: $BoardNmbr   " .
                    "I2C_Address: $I2C_Address   pre_scale: $pre_scale");

# Validate that address is within the Adafruit 16-channel driver range.
   if ($I2C_Address >= $minAddr and $I2C_Address <= $maxAddr) {
      $driver = RPi::I2C->new($I2C_Address);
      unless ($driver->check_device($I2C_Address)) {
         &DisplayError("I2C_InitServoDriver, Failed to initialize " .
                       "I2C address: " . sprintf("0x%.2x",$I2C_Address));
         return 1;
      }
      $driver->write_byte(0x10, $PCA9685{'AllLedOffH'});  # Orderly shutdown.
      sleep 0.01;                                  # Wait for channels to stop.
      $mode_data = $driver->read_byte($PCA9685{'ModeReg1'});
      $driver->write_byte(($mode_data | $sleep_mode), $PCA9685{'ModeReg1'});
      $driver->write_byte($pre_scale, $PCA9685{'PreScale'});
      $mode_data = ($mode_data & $normal_mode) | $auto_inc;
      $driver->write_byte(($mode_data), $PCA9685{'ModeReg1'});
      &DisplayDebug(2, "I2C_InitServoDriver, PreScale: " .
                       $driver->read_byte($PCA9685{'PreScale'}));
      undef($driver);
   }
   else {
      &DisplayError("I2C_InitServoDriver, Invalid I2C address: " .
                    "$I2C_Address   Board: $BoardNmbr");
      return 1;
   }
   return 0;
}

# ==========================================================================
# FUNCTION:  ProcessTurnoutFile
#
# DESCRIPTION:
#    This routine reads or writes the specified turnout data file. Used to
#    retain turnout operational data between program starts.
#
# CALLING SYNTAX:
#    $result = &ProcessTurnoutFile($FileName, $Function, \%TurnoutData);
#
# ARGUMENTS:
#    $FileName       File to Read/Write
#    $Function       "Read" or "Write"
```

```
#    $TurnoutData    Pointer to %TurnoutData hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub ProcessTurnoutFile {

   my($FileName, $Function, $TurnoutData) = @_;
   my($turnout, $rec);
   my(@fileData) = ();

   my(@keyList) = ('Pid','Addr','Port','Pos','Rate','Open','Middle','Close',
                   'MinPos','MaxPos','Id');

   &DisplayDebug(2, "ProcessTurnoutFile, Function: $Function   " .
                    "keyList: '@keyList'");

   if ($Function =~ m/^Read$/i) {
      if (-e $FileName) {
         if (&ReadFile($FileName, \@fileData)) {
            &DisplayWarning("ProcessTurnoutData, Using default " .
                            "turnout data.");
         }
         else {
            %$TurnoutData = ();
            foreach my $rec (@fileData) {
               next if ($rec =~ m/^\s*$/ or $rec =~ m/^#/);
               if ($rec =~ m/Turnout:\s*(\d+)/i) {
                  $turnout = sprintf("%2s",$1);
                  $$TurnoutData{$turnout}{'Pid'} = 0;
                  foreach my $key (@keyList) {
                     if ($key eq 'Id') {
                        if ($rec =~ m/$key:(.+)/) {
                           $$TurnoutData{$turnout}{$key} = &Trim($1);
                        }
                        else {
                           &DisplayWarning("ProcessTurnoutData, " .
                                           "'$key' not found: '$rec'");
                           next;
                        }
                     }
                     else {
                        if ($rec =~ m/$key:\s*(\d+)/) {
                           $$TurnoutData{$turnout}{$key} = $1;
                        }
                        else {
                           &DisplayWarning("ProcessTurnoutData, " .
                                           "'$key' not found: '$rec'");
                           next
                        }
                     }
                     &DisplayDebug(2, "ProcessTurnoutFile, " .
                                 "Turnout: $turnout   key: $key   value: " .
                                 "$$TurnoutData{$turnout}{$key}");
                  }
               }
               else {
```

```
                    &DisplayWarning("ProcessTurnoutData, 'Turnout' key " .
                                    "not found: '$rec'");
                }
             }
          }
          $rec = scalar keys %$TurnoutData;
          &DisplayDebug(1, "ProcessTurnoutFile, Function: $Function " .
                           "$rec turnout records.");
       }
       else {
          &DisplayWarning("ProcessTurnoutData: File not found: $FileName.");
          &DisplayWarning("ProcessTurnoutData: Using default turnout data.");
       }
    }
    elsif ($Function =~ m/^Write$/i) {
       push (@fileData, "# ================================================");
       push (@fileData, "# Turnout data file. Loaded during program start.");
       push (@fileData, "# Edited values will be used upon next start. See");
       push (@fileData, "# DnB.pl 'Turnout Related Data' section for more ");
       push (@fileData, "# information.");
       push (@fileData, "# ================================================");

       $rec = scalar keys %$TurnoutData;
       &DisplayDebug(1, "ProcessTurnoutFile, Function: $Function $rec " .
                        "turnout records.");

       foreach my $turnout (sort keys %$TurnoutData) {
          next if ($turnout =~ m/^\s*$/);
          $rec = join(":", "Turnout", $turnout);
          $$TurnoutData{$turnout}{'Pid'} = 0;
          foreach my $key (@keyList) {
             $rec = join(" ", $rec, join(":", $key,
                         $$TurnoutData{$turnout}{$key}));
          }
          push (@fileData, $rec);
          &DisplayDebug(2, "ProcessTurnoutFile, $Function: $rec");
       }
       &WriteFile($FileName, \@fileData);
    }
    else {
       &DisplayWarning("ProcessTurnoutData, Unsupported function: $Function");
    }
    return 0;
}

# ============================================================================
# FUNCTION:  InitTurnouts
#
# DESCRIPTION:
#    Called once during DnB startup, this routine initializes all turnouts to
#    the PWM position specified in %TurnoutData. This ensures that all servo
#    driver board channels are synchronized to the %TurnoutData specified PWM
#    position.
#
#    A check of the %TurnoutData PWM values is performed since these values are
#    normally loaded from the user editable TurnoutDataFile. If an out-of-range
#    value is detected, initialization is aborted and an error is returned.
#
#    If optional data is specified, the servo is set to the specified PWM
#    position. This position is used for physical turnout point adjustment.
```

```
#
# CALLING SYNTAX:
#    $result = &InitTurnouts(\%ServoBoardAddress, \%TurnoutData, $Turnout,
#                            $Position);
#
# ARGUMENTS:
#    $ServoBoardAddress      Pointer to %ServoBoardAddress hash.
#    $TurnoutData            Pointer to %TurnoutData hash.
#    $Turnout                Optional; turnout to position.
#    $Position               Optional; position to set.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub InitTurnouts {
   my($ServoBoardAddress, $TurnoutData, $Turnout, $Position) = @_;
   my($board, $pwm);
   my($min,$max) = (300,900);                # Absolute PWM values.
   my($rmin,$rmax) = (1,850);                # Absolute Rate values.
   my($fail) = 0;

   &DisplayDebug(2, "InitTurnouts, $Turnout: $Turnout   $Position: " .
                            "'$Position'");

   if ($Turnout ne '') {
      $Turnout = "0${Turnout}" if (length($Turnout) == 1);
   }
   if ($Position ne 'Open' and $Position ne 'Close') {
      $Position = 'Middle';
   }

   # Validate the %TurnoutData PWM values.
   &DisplayMessage("Validate turnout PWM working values ...");
   foreach my $tNmbr (sort keys %$TurnoutData) {
      foreach my $pos ('MinPos','MaxPos','Open','Middle','Close','Pos') {
         $pwm = $$TurnoutData{$tNmbr}{$pos};
         if ($pwm < $min or $pwm > $max) {
            &DisplayError("InitTurnouts, turnout $tNmbr $pos " .
                          "value out of range: $pwm");
            $fail = 1;
         }
         elsif ($pwm < $$TurnoutData{$tNmbr}{'MinPos'} or
            $pwm > $$TurnoutData{$tNmbr}{'MaxPos'}) {
            &DisplayError("InitTurnouts, turnout $tNmbr $pos " .
                          "value out of min/max limit: $pwm");
            $fail = 1;
         }
      }
      $pwm = $$TurnoutData{$tNmbr}{'Rate'};
      if ($pwm < $rmin or $pwm > $rmax) {
         &DisplayError("InitTurnouts, turnout $tNmbr Rate " .
                       "value out of range: $pwm");
         $fail = 1;
      }
   }
   return 1 if ($fail == 1);    # Error return if failure.

```

```
301      # Initialize servo channel on the driver boards.
302      for ($board = 1; $board <= scalar keys(%$ServoBoardAddress); $board++) {
303         if ($$ServoBoardAddress{$board} == 0) {
304            &DisplayDebug(1, "InitTurnouts, Skip board $board " .
305                             "I2C_Address 0, code debug.");
306            next;
307         }
308         &DisplayMessage("Initializing turnout I2C board $board ...");
309         return 1 if (&I2C_InitServoDriver($board, $$ServoBoardAddress{$board}));
310
311         &DisplayMessage("Initializing turnout positions on board $board ...");
312
313         foreach my $tNmbr (sort keys %$TurnoutData) {
314            if ($$TurnoutData{$tNmbr}{'Addr'} == $$ServoBoardAddress{$board}) {
315               if ($Turnout eq '00' or $Turnout eq $tNmbr) {
316                  $$TurnoutData{$tNmbr}{'Pos'} = $$TurnoutData{$tNmbr}{$Position};
317               }
318               elsif ($$TurnoutData{$tNmbr}{'Pos'} != $$TurnoutData{$tNmbr}{'Open'}
319                      and $$TurnoutData{$tNmbr}{'Pos'} != $$TurnoutData{$tNmbr}
320                      {'Close'}) {
321                  $$TurnoutData{$tNmbr}{'Pos'} = $$TurnoutData{$tNmbr}{'Middle'};
322               }
323
324               if (&SetTurnoutPosition($$TurnoutData{$tNmbr}{'Pos'}, $tNmbr,
325                                       $TurnoutData)) {
326                  &DisplayWarning("InitTurnouts, Failed to set " .
327                                  "turnout.   board $board   Turnout: $tNmbr" .
328                                  "Position: $$TurnoutData{$tNmbr}{'Pos'}");
329                  $fail = 1;
330               }
331
332               $$TurnoutData{$tNmbr}{'Pid'} = 0;  # Ensure the Pid value is 0.
333               sleep 0.1;                         # Delay so we don't overtax
334                                                  # the servo power supply.
335            }
336         }
337         &DisplayMessage("All board $board turnouts initialized.");
338      }
339      if ($Turnout ne '') {
340         if ($Turnout eq '00') {
341            &DisplayMessage("All turnouts set to $Position position.");
342         }
343         else {
344            &DisplayMessage("Turnout $Turnout set to $Position position.");
345         }
346      }
347      return 1 if ($fail == 1);    # Error return if failure.
348      return 0;
349   }
350
351   # ============================================================================
352   # FUNCTION:  MoveTurnout
353   #
354   # DESCRIPTION:
355   #    This routine moves the turnout servo using the specified data. It is used
356   #    to perform a slow motion position change. This is done by forking to a
357   #    child process and calling SetTurnoutPosition 50 times a second until the
358   #    move is complete. Each call positions the turnout servo toward the final
359   #    position by a move step amount ('Rate'/50). Once the move is completed,
360   #    the turnout position is updated in the TurnoutData hash and the child
```

```
#    exits. A 'Rate' value of 450 positions the turnout from Open (350) to
#    Close (850) in about 1.1 seconds.
#
# CALLING SYNTAX:
#    $result = &MoveTurnout($Function, $TurnoutNmbr, \%TurnoutData);
#
# ARGUMENTS:
#    $Function       'Open', 'Middle', or 'Close'.
#    $TurnoutNmbr    Turnout number; two digit hash index.
#    $TurnoutData    Pointer to TurnoutData hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error, 2 = Already in position.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub MoveTurnout {
   my($Function, $TurnoutNmbr, $TurnoutData) = @_;
   my($result, $pwmCurrent, $pwmFinal, $moveRate, $moveStep, $pid);
   my($timeout) = 40;    # Wait 10 seconds (40/.25) for move to complete.

   &DisplayDebug(2, "MoveTurnout, Entry ...   $Function $TurnoutNmbr");

   if ($TurnoutNmbr ne "") {
      if ($Function =~ m/Open/i) {
         $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'Open'};
      }
      elsif ($Function =~ m/Middle/i) {
         $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'Middle'};
      }
      elsif ($Function =~ m/Close/i) {
         $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'Close'};
      }
      else {
         &DisplayError("MoveTurnout, invalid function: '$Function'");
         return 1;
      }

      # Make sure the requested move will not exceed a min/max limit.
      $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'MinPos'}
                  if ($pwmFinal < $$TurnoutData{$TurnoutNmbr}{'MinPos'});
      $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'MaxPos'}
                  if ($pwmFinal > $$TurnoutData{$TurnoutNmbr}{'MaxPos'});

      # Check and wait for turnout to be idle.
      while ($$TurnoutData{$TurnoutNmbr}{'Pid'} > 0 and $timeout > 0) {
         if (($timeout % 4) == 0) {
            &DisplayDebug(2, "MoveTurnout, waiting for previous move " .
                             "to complete. timeout: $timeout   Pid: " .
                             "$$TurnoutData{$TurnoutNmbr}{'Pid'}   Pos: " .
                             "$$TurnoutData{$TurnoutNmbr}{'Pos'}");
         }
         $timeout--;
         sleep 0.25;                # Wait quarter sec.
      }

      # Abort turnout move if still active.
      if ($$TurnoutData{$TurnoutNmbr}{Pid} > 0) {
         &DisplayError("MoveTurnout, Turnout $TurnoutNmbr, Previous " .
```

```
421                         "move still in progress, pid: " .
422                         "$$TurnoutData{$TurnoutNmbr}{'Pid'}.");
423
424            # Check if the process is running, $result == 0. If so, kill it.
425            # Cleanup state data and continue new turnout move.
426            $result = waitpid($$TurnoutData{$TurnoutNmbr}{'Pid'}, WNOHANG);
427            system("kill -9 $$TurnoutData{$TurnoutNmbr}{'Pid'}") if ($result == 0);
428            $$TurnoutData{$TurnoutNmbr}{'Pid'} = 0;
429         }
430
431         $pwmCurrent = $$TurnoutData{$TurnoutNmbr}{'Pos'};
432         if ($pwmCurrent == $pwmFinal) {           # Done if already in position.
433            &DisplayDebug(2, "MoveTurnout, $TurnoutNmbr already in " .
434                             "requested position: $pwmFinal");
435            return 2;
436         }
437
438         $moveRate = $$TurnoutData{$TurnoutNmbr}{'Rate'};
439
440         if ($moveRate > 0) {
441            # Fork program to complete the move. Use Forks::Super which is a go
442            # between the parent and child. It has a function for writing child
443            # data back to the main program using child STDOUT and STDERR. It is
444            # not necessary to 'reap' the child when using Forks::Super. Also,
445            # SIG{CHILD} should not be set by this program. It is set/used by
446            # Forks::Super. Do no other printing, including debug output.
447            #
448            # STDERR: move complete. $TurnoutData{<tNmbr>}{'Pid'} set to 0.
449            # STDOUT: new turnout position. $TurnoutData{<tNmbr>}{'Pos'}.
450
451            &DisplayDebug(2, "MoveTurnout, pre-fork: $Function " .
452                             "$TurnoutNmbr   pwmCurrent: $pwmCurrent" .
453                             "   pwmFinal: $pwmFinal" .
454                             "   moveRate: $moveRate");
455
456            $pid = fork { os_priority => 1,
457                          stdout => \$$TurnoutData{$TurnoutNmbr}{'Pos'},
458                          stderr => \$$TurnoutData{$TurnoutNmbr}{'Pid'} };
459            if (!defined($pid)) {
460               &DisplayError("TurnoutChildProcess, Failed to create " .
461                             "child process. $!");
462               return 1;
463            }
464   #----------
465            elsif ($pid == 0) {          # fork returned 0, so this is the child
466               $moveStep = $moveRate/50;              # Step increment
467               while ($pwmCurrent != $pwmFinal) {
468                  if ($pwmCurrent < $pwmFinal) {      # Determine move direction
469                     $pwmCurrent += $moveStep;
470                     $pwmCurrent = $pwmFinal if ($pwmCurrent > $pwmFinal);
471                  }
472                  else {
473                     $pwmCurrent -= $moveStep;
474                     $pwmCurrent = $pwmFinal if ($pwmCurrent < $pwmFinal);
475                  }
476
477                  if (&SetTurnoutPosition($pwmCurrent, $TurnoutNmbr, $TurnoutData)) {
478                     # Retain previous pwmCurrent in Pos if error is returned.
479                     print STDERR 0;           # Clear Pid, move has completed.
480                     exit(1);                  # Starting position is retained.
```

```
481                }
482                sleep 0.02;
483             }
484             print STDOUT $pwmCurrent;        # Store position of turnout
485             print STDERR 0;                  # Clear Pid, move has completed.
486             exit(0);
487          }
488   #----------
489          $$TurnoutData{$TurnoutNmbr}{'Pid'} = $pid;  # Parent: Move in-progress.
490          &DisplayDebug(1, "MoveTurnout, $Function $TurnoutNmbr " .
491                           "forked pid: $$TurnoutData{$TurnoutNmbr}{'Pid'}");
492       }
493       else {
494          &DisplayWarning("MoveTurnout, Rate value must be greater than 0.");
495          return 1;
496       }
497    }
498    else {
499       &DisplayError("MoveTurnout, invalid turnout number: $TurnoutNmbr");
500       return 1;
501    }
502    return 0;
503 }
504
505 # ============================================================================
506 # FUNCTION:  SetTurnoutPosition
507 #
508 # DESCRIPTION:
509 #    This routine sets the turnout servo using the specified data. This
510 #    routune writes the I2C interface with the needed command bytes.
511 #
512 #    This routine checks the Position value to provide some servo protection
513 #    due to a possible program runtime error.
514 #
515 # CALLING SYNTAX:
516 #    $result = &SetTurnoutPosition($Position, $TurnoutNmbr, \%TurnoutData);
517 #
518 # ARGUMENTS:
519 #    $Position       PWM position to set.
520 #    $TurnoutNmbr    Turnout number.
521 #    $TurnoutData    Pointer to TurnoutData hash.
522 #
523 # RETURNED VALUES:
524 #    0 = Success,  1 = Error.
525 #
526 # ACCESSED GLOBAL VARIABLES:
527 #    None.
528 # ============================================================================
529 sub SetTurnoutPosition {
530    my($Position, $TurnoutNmbr, $TurnoutData) = @_;
531    my($driver, $reg_start, $reg_data_on, $reg_data_off);
532    my(@data) = ();
533
534    &DisplayDebug(2, "SetTurnoutPosition, $TurnoutNmbr - $Position");
535
536    if (exists($$TurnoutData{$TurnoutNmbr})) {
537       $Position = int($Position);
538       if ($Position < $$TurnoutData{$TurnoutNmbr}{'MinPos'}) {
539          $Position = $$TurnoutData{$TurnoutNmbr}{'MinPos'};
540          &DisplayWarning("SetTurnoutPosition, Turnout $TurnoutNmbr " .
```

```
                        "PWM value beyond MinPos limit. Set to " .
                        "MinPos $Position");
      }
      if ($Position > $$TurnoutData{$TurnoutNmbr}{'MaxPos'}) {
         $Position = $$TurnoutData{$TurnoutNmbr}{'MaxPos'};
         &DisplayWarning("SetTurnoutPosition, Turnout $TurnoutNmbr " .
                        "PWM value beyond MaxPos limit. Set to ".
                        "MaxPos $Position");
      }

      $reg_start = (($$TurnoutData{$TurnoutNmbr}{'Port'} % 16) * 4) + 6;

      # Stagger pulse start (* 10) to minimuze power drops.
      $reg_data_on = $$TurnoutData{$TurnoutNmbr}{'Port'} * 10;
      push (@data, ($reg_data_on & 0xFF));            # on_L
      push (@data, (($reg_data_off >> 8) & 0x0F));    # on_H
      $reg_data_off = $reg_data_on + $Position;
      push (@data, ($reg_data_off & 0xFF));           # off_L
      push (@data, (($reg_data_off >> 8) & 0x0F));    # off_H

      $driver = RPi::I2C->new($$TurnoutData{$TurnoutNmbr}{'Addr'});
      unless ($driver->check_device($$TurnoutData{$TurnoutNmbr}{'Addr'})) {
         &DisplayError("SetTurnoutPosition, Failed to initialize " .
                       "I2C address: " .
                       sprintf("%.2x",$$TurnoutData{$TurnoutNmbr}{'Addr'}));
         return 1;
      }
      $driver->write_block(\@data, $reg_start);
      undef($driver);
   }
   else {
      &DisplayError("SetTurnoutPosition, invalid turnout number: $TurnoutNmbr");
      return 1;
   }
   return 0;
}

# ============================================================================
# FUNCTION:  TestTurnouts
#
# DESCRIPTION:
#    This routine cycles the specified turnout range between the open and
#    closed positions.
#
# CALLING SYNTAX:
#    $result = &TestTurnouts($Range, \%TurnoutData);
#
# ARGUMENTS:
#    $Range           Turnout number or range to use.
#    $TurnoutData     Pointer to TurnoutData hash.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    $main::MainRun
# ============================================================================
sub TestTurnouts {

   my($Range, $TurnoutData) = @_;
```

```
601     my($moveResult, $turnout, $start, $end, $nmbr, $oper, $pid, $cnt,
602        @turnoutNumbers, @inProgress, $position);
603     my($cntTurnout) = scalar keys %$TurnoutData;
604     my(%operation) = (1 => 'Open ', 2 => 'Close');
605     my(@turnoutList) = ();
606     my($random, $wait) = (0, 0);
607
608     &DisplayDebug(1, "TestTurnouts, Entry ... Range: '$Range'   " .
609                      "cntTurnout: $cntTurnout");
610
611     # ============================
612     # Set specified position and exit.
613
614     if ($Range =~ m/^(Open):(\d+)/i or $Range =~ m/^(Close):(\d+)/i or
615         $Range =~ m/^(Middle):(\d+)/i) {
616        $position = ucfirst(lc $1);
617        $turnout = $2;
618        $turnout = "0${turnout}" if (length($turnout) == 1);
619
620        # The %TurnoutData Id string must contain the word turnout.
621        if ($$TurnoutData{$turnout}{'Id'} =~ m/turnout/) {
622           &MoveTurnout($position, $turnout, $TurnoutData);
623           &DisplayMessage("Turnout $turnout set to '$position'.");
624        }
625        else {
626           &DisplayError("TestTurnouts, invalid turnout number: $turnout");
627        }
628        exit(0);
629     }
630     elsif ($Range =~ m/^(Open)$/i or $Range =~ m/^(Close)$/i or
631            $Range =~ m/^(Middle)$/i) {
632        $position = ucfirst(lc $1);
633
634        # The %TurnoutData Id string must contain the word turnout.
635        foreach my $turnout (sort keys %$TurnoutData) {
636           if ($$TurnoutData{$turnout}{'Id'} =~ m/turnout/) {
637              &MoveTurnout($position, $turnout, $TurnoutData);
638              &DisplayDebug(1, "TestTurnouts, turnout: $turnout set " .
639                               "to $position");
640           }
641        }
642        &DisplayMessage("All turnouts set to '$position'.");
643        exit(0);
644     }
645
646     # ============================
647     # Process special modifiers and then setup for looped testing.
648
649     if ($Range =~ m/r/i) {
650        $random = 1;
651        $Range =~ s/r//i;
652     }
653     if ($Range =~ m/w/i) {
654        $wait = 1;
655        $Range =~ s/w//i;
656     }
657
658     if ($Range =~ m/(\d+):(\d+)/) {   # Range specified.
659        $start = $1;
660        $end = $2;
```

```
661         if ($start > $end or $start <= 0 or $start > $cntTurnout or $end <= 0 or
662             $end > $cntTurnout) {
663            &DisplayError("TestTurnouts, invalid turnout range: '$Range'" .
664                          "   cntTurnout: $cntTurnout");
665            return 1;
666         }
667         for ($turnout = $start; $turnout <= $end; $turnout++) {
668            push (@turnoutList, $turnout);
669         }
670      }
671      else {
672         @turnoutList = split(",", $Range);
673      }
674      &DisplayDebug(1, "TestTurnouts, random: $random   wait: $wait   " .
675                       "turnoutList: '@turnoutList'");
676
677      # Identify the servos being used for turnouts. The %TurnoutData Id string
678      # must contain the word turnout.
679      foreach my $key (sort keys %$TurnoutData) {
680         if ($$TurnoutData{$key}{'Id'} =~ m/turnout/) {
681            push (@turnoutNumbers, $key);
682         }
683      }
684
685      $oper = 'Open  ';
686      while ($main::MainRun) {
687         # For random testing, we randomize the turnoutNumbers list and also the
688         # Open/Close operation. For non-random, Open and then Close the turnouts
689         # in the specified order.
690         &ShuffleArray(\@turnoutNumbers) if ($random == 1);
691
692         foreach my $turnout (@turnoutNumbers) {
693            return 0 unless ($main::MainRun);
694            $nmbr = $turnout;
695            $nmbr =~ s/^0//;
696            if (grep /^$nmbr$/, @turnoutList) {  # Move turnout if on the list.
697               $oper = $operation{(int(rand(2))+1)} if ($random == 1);
698               if ($#inProgress < 0) {
699                  &DisplayMessage("TestTurnouts, $oper $turnout   Concurrent " .
700                                  "moves: none");
701               }
702               else {
703                  &DisplayMessage("TestTurnouts, $oper $turnout   Concurrent " .
704                                  "moves: @inProgress");
705               }
706               $moveResult = &MoveTurnout($oper, $turnout, $TurnoutData);
707               return 1 if ($moveResult == 1);
708               if ($moveResult == 2) {
709                  &DisplayDebug(2, "TestTurnouts, MoveTurnout $turnout returned " .
710                                   "already in position.");
711               }
712               elsif ($moveResult == 0) {
713                  if ($wait == 1) {
714                     $cnt = 20;
715                     while ($$TurnoutData{$turnout}{'Pid'}) {
716                        if ($cnt == 0) {
717                           &DisplayError("TestTurnouts, timeout waiting for " .
718                                         "turnout $turnout to complete positioning.");
719                           return 1;
720                        }
```

```
                        &DisplayDebug(2, "TestTurnouts, waiting for " .
                                         "pid: $$TurnoutData{$turnout}{'Pid'}");
                        sleep 0.5;
                        $cnt--;
                     }
                     &DisplayDebug(2, "TestTurnouts, Turnout $turnout new position: " .
                                      "$$TurnoutData{$turnout}{'Pos'}");
                  }
               }
               @inProgress = ();
               foreach my $key (sort keys(%$TurnoutData)) {
                  push (@inProgress, $key) if ($$TurnoutData{$key}{'Pid'} != 0);
               }
               sleep 0.05 unless ($moveResult == 2);
            }
         }

         if ($random == 0) {   # Change if doing sequential testing.
            if ($oper =~ m/Open/) {
               $oper = 'Close ';
            }
            else {
               $oper = 'Open  ';
            }
         }
         sleep 2;
      }
      return 0;
   }

   return 1;
```