```perl
# ============================================================================
# FILE: DnB_Signal.pm                                              8/03/2020
#
# SERVICES:  DnB SIGNAL FUNCTIONS
#
# DESCRIPTION:
#    This perl module provides signal related functions used by the DnB model
#    railroad control program.
#
# PERL VERSION: 5.24.1
#
# ============================================================================
use strict;
# ----------------------------------------------------------------------------
# Package Declaration
# ----------------------------------------------------------------------------
package DnB_Signal;
require Exporter;
our @ISA = qw(Exporter);

our @EXPORT = qw(
   Init_SignalDriver
   SetSignalColor
   SetSemaphoreSignal
   SignalChildProcess
   TestSignals
);

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

# ============================================================================
# FUNCTION:  Init_SignalDriver
#
# DESCRIPTION:
#    This routine initializes the GPIO pins associated with the LED driver on
#    the DnB model railroad. A shift register utilizing multiple 74HC595 chips
#    is used. Data is shifted in serially using GPIO pins connected to the data
#    and clock inputs of the shift register.
#
#    A second group of GPIOs is used to control the track power polarity relays.
#
# CALLING SYNTAX:
#    $result = &Init_SignalDriver(\%GpioData, $RegisterLength);
#
# ARGUMENTS:
#    $GpioData          Pointer to GPIO data.
#    $RegisterLength    Shift register bit length.
#
# RETURNED VALUES:
#    0 = Success,  1 = Error.
#
# ACCESSED GLOBAL VARIABLES:
#    None.
# ============================================================================
sub Init_SignalDriver {
   my($GpioData, $RegisterLength) = @_;
   my($x, $pin);
```

```
61
62      &DisplayDebug(2, "Init_SignalDriver, RegisterLength: $RegisterLength");
63
64  # Create a Raspberry Pi object for each GPIO and set to defaults.
65
66      foreach my $gpio (sort keys %$GpioData) {
67         if ($$GpioData{$gpio}{'Obj'} == 0) {
68            if ($gpio =~ m/^GPIO(\d*)_/) {
69               $pin = $1;
70               $$GpioData{$gpio}{'Obj'} = RPi::Pin->new($pin);
71               if ($$GpioData{$gpio}{'Obj'} != 0) {
72                  &DisplayDebug(1, "Init_SignalDriver, $gpio object " .
73                                   "successfully created.");
74                  $$GpioData{$gpio}{'Obj'}->mode($$GpioData{$gpio}{'Mode'});
75                  if ($$GpioData{$gpio}{'Mode'} == 0) {
76                     # 0=None, 1=Pulldown, 2=Pullup
77                     $$GpioData{$gpio}{'Obj'}->pull(2);   # Enable pullup on pin.
78                  }
79                  elsif ($$GpioData{$gpio}{'Mode'} == 1) {
80                     $$GpioData{$gpio}{'Obj'}->write(0);          # Set GPIO low.
81                  }
82               }
83               else {
84                  &DisplayError("Init_SignalDriver, failed to create " .
85                                "$gpio object. $!");
86                  return 1;
87               }
88            }
89            else {
90               &DisplayError("Init_SignalDriver, failed to parse pin " .
91                             "number from '$gpio'.");
92               return 1;
93            }
94         }
95         else {
96            &DisplayWarning("Init_SignalDriver, $gpio object already active.");
97         }
98      }
99
100 # Test toggle.
101 #   while (1) {
102 #      foreach my $gpio (sort keys %$GpioData) {
103 #         $$GpioData{$gpio}{'Obj'}->write(1);
104 #      }
105 #      &DisplayDebug(1, "Init_SignalDriver, All GPIOs HIGH.");
106 #      sleep 2;
107 #      foreach my $gpio (sort keys %$GpioData) {
108 #         $$GpioData{$gpio}{'Obj'}->write(0);
109 #      }
110 #      &DisplayDebug(1, "Init_SignalDriver, All GPIOs LOW.");
111 #      sleep 2;
112 #   }
113 #   exit(0);
114
115 # Set all signals to 'Off'. GPIO27_SCLK, GPIO22_DATA, and GPIO17_XLAT are
116 # set to 0 from above GPIO instantiation.
117
118     $$GpioData{'GPIO23_OUTE'}{'Obj'}->write(1);       # Blank outputs.
119     for ($x = 0; $x < $RegisterLength; $x++) {
120        $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(1);    # Set SCLK high (store bit).
```

```
121         $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);    # Set SCLK low.
122      }
123      $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(1);       # Set XLAT high (latch data).
124      $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(0);       # Set XLAT low.
125      $$GpioData{'GPIO23_OUTE'}{'Obj'}->write(0);       # Enable outputs.
126
127      &DisplayMessage("Init_SignalDriver, All signals and relays set to 'Off'.");
128      return 0;
129   }
130
131   # ============================================================================
132   # FUNCTION:  SetSignalColor
133   #
134   # DESCRIPTION:
135   #    This routine sets the specified signal to the specified color. Each signal
136   #    LED is a two lead red/green device wired to the two consecutive register
137   #    bits. Red is illuminated with one current flow direction and green is
138   #    illuminated with the opposite current flow direction. Current direction is
139   #    controlled by which of the two register bits is set high/low. The local
140   #    signalColor hash holds the values for each color.
141   #
142   #    This routine is called by SetSemaphoreSignal to control lamp on/off. The
143   #    SemaphoreFlag argument is used to prevent this routine from setting the new
144   #    color value into $$SignalData{$Signal}{'Current'}. The SetSemaphoreSignal
145   #    routine will set the value once the associated servo move has completed.
146   #
147   #    The necessary mask values are created and sent to SignalChildProcess stdin
148   #    to set the specified signal (01-16) to the specified color.
149   #
150   # CALLING SYNTAX:
151   #    $result = &SetSignalColor($Signal, $Color, $SignalChildPid,
152   #                              \%SignalData, $SemaphoreFlag);
153   #
154   # ARGUMENTS:
155   #    $Signal           Signal number to set.
156   #    $Color            Signal color, 'Red', 'Grn', 'Yel', or 'Off'
157   #    $SignalChildPid   PID of child signal refresh process.
158   #    $SignalData       Pointer to SignalData hash.
159   #    $SemaphoreFlag    Suppresses setting of current color when set.
160   #
161   # RETURNED VALUES:
162   #    0 = Success,  1 = Error.
163   #
164   # ACCESSED GLOBAL VARIABLES:
165   #    None.
166   # ============================================================================
167   sub SetSignalColor {
168      my($Signal, $Color, $SignalChildPid, $SignalData, $SemaphoreFlag) = @_;
169      my($data1, $data2, $mask);
170
171      my(%signalColor1) = ('Off' => 0b00, 'Red' => 0b01, 'Grn' => 0b10,
172                           'Yel' => 0b01);
173      my(%signalColor2) = ('Off' => 0b00, 'Red' => 0b01, 'Grn' => 0b10,
174                           'Yel' => 0b10);
175
176      &DisplayDebug(2, "SetSignalColor, Signal: $Signal   Color: " .
177                       "$Color   SemaphoreFlag: '$SemaphoreFlag'");
178
179      if ($Signal ne "") {
180
```

```
181         # Create mask values for the specified signal.
182         if ($Color eq 'Red' or $Color eq 'Grn' or $Color eq 'Off' or
183             $Color eq 'Yel') {
184            $mask = 0xFFFFFFFF & (~(0b11 << (($Signal - 1) * 2)));
185            $data1 = $signalColor1{$Color} << (($Signal - 1) * 2);
186            $data2 = $signalColor2{$Color} << (($Signal - 1) * 2);
187
188            &DisplayDebug(2, "SetSignalColor, -----  16151413121110 9 " .
189                             "8 7 6 5 4 3 2 1");
190            &DisplayDebug(2, "SetSignalColor, mask:  " .
191                             sprintf("%0.32b", $mask));
192            &DisplayDebug(2, "SetSignalColor, data1: " .
193                             sprintf("%0.32b", $data1));
194            &DisplayDebug(2, "SetSignalColor, data2: " .
195                             sprintf("%0.32b", $data2));
196
197            Forks::Super::write_stdin($SignalChildPid, join(",", $mask, $data1,
198                                                           $data2, "-\n"));
199            $$SignalData{$Signal}{'Current'} = $Color unless ($SemaphoreFlag);
200         }
201         else {
202            &DisplayError("SetSignalColor, invalid signal color: $Color");
203            return 1;
204         }
205      }
206      else {
207         &DisplayError("SetSignalColor, invalid signal number: $Signal");
208         return 1;
209      }
210      return 0;
211   }
212
213   # ============================================================================
214   # FUNCTION:  SetSemaphoreSignal
215   #
216   # DESCRIPTION:
217   #    This routine sets the specified Semaphore signal to the specified color.
218   #    SetSignalColor is called to set the lamp on (color bit pair 'Grn') or off
219   #    as required. MoveTurnout is called to position the servo attached to the
220   #    semaphore flag board.
221   #
222   #   This routine is call for each iteration of the main loop until the 'Position'
223   #   value for the semaphore in SemaphoreData is set to the necessary color.
224   #
225   # CALLING SYNTAX:
226   #    $result = &SetSemaphoreSignal($Signal, $Color, $SignalChildPid, \%SignalData,
227   #                                  \%SemaphoreData, \%TurnoutData);
228   #
229   # ARGUMENTS:
230   #    $Signal           Signal number to set.
231   #    $Color            Signal color, 'Red', 'Grn', 'Yel', or 'Off'
232   #    $SignalChildPid   PID of child signal refresh process.
233   #    $SignalData       Pointer to %SignalData hash.
234   #    $SemaphoreData    Pointer to %SemaphoreData hash.
235   #    $TurnoutData      Pointer to %TurnoutData hash.
236   #
237   # RETURNED VALUES:
238   #    0 = Success,  1 = Error.
239   #
240   # ACCESSED GLOBAL VARIABLES:
```

```
#    None.
# ============================================================================
sub SetSemaphoreSignal {
   my($Signal, $Color, $SignalChildPid, $SignalData, $SemaphoreData,
      $TurnoutData) = @_;
   my($moveResult, $servo);
   my(%flagPosition) = ('Grn' => 'Open', 'Yel' => 'Middle', 'Red' => 'Close',
                        'Off' => 'Open');

   &DisplayDebug(1, "SetSemaphoreSignal, Signal: $Signal   Color: $Color");

   if ($Signal ne "" and exists($$SemaphoreData{$Signal})) {
      $servo = $$SemaphoreData{$Signal}{'Servo'};
      if ($$SemaphoreData{$Signal}{'InMotion'} == 1) {
         if ($$TurnoutData{$servo}{Pid} == 0) {
            $$SemaphoreData{$Signal}{'InMotion'} = 0;
            &DisplayDebug(2, "SetSemaphoreSignal, semaphore $Signal " .
                             "move completed.");

            # Turn on lamp unless color is off.
            if ($Color ne 'Off') {
               if (&SetSignalColor($Signal, 'Grn', $SignalChildPid, $SignalData,
                                   'semaphore')) {
                  &DisplayError("SetSemaphoreSignal, SetSignalColor " .
                                "$Signal 'Grn' returned error.");
                  return 1;
               }
               $$SemaphoreData{$Signal}{'Lamp'} = 'On';
            }
            $$SignalData{$Signal}{'Current'} = $Color;
            &DisplayMessage("SetSemaphoreSignal, semaphore $Signal " .
                            "set to $Color.");
         }
      }
      else {
         if ($$SignalData{$Signal}{'Current'} ne $Color) {
            &DisplayDebug(1, "SetSemaphoreSignal, moving semaphore " .
                             "$Signal to position $Color");

            # Turn off lamp.
            if (&SetSignalColor($Signal, 'Off', $SignalChildPid, $SignalData,
                                'semaphore')) {
               &DisplayError("SetSemaphoreSignal, SetSignalColor " .
                             " $Signal 'Off' returned error.");
               return 1;
            }
            $$SemaphoreData{$Signal}{'Lamp'} = 'Off';

            # Move semaphore flag board to requested position.
            $moveResult = &MoveTurnout($flagPosition{$Color}, $servo, $TurnoutData);
            if ($moveResult == 0) {
               $$SemaphoreData{$Signal}{'InMotion'} = 1;
               &DisplayDebug(2, "SetSemaphoreSignal, semaphore " .
                                "$Signal move inprogress.");
            }
            elsif ($moveResult == 1) {
               &DisplayError("SetSemaphoreSignal, MoveTurnout $servo '" .
                             $flagPosition{$Color} . "' returned error.");
               return 1;
            }
```

```
301
302             # If MoveTurnout uses return 2, the servo is already in the
303             # requested position. Complete the related processing.
304             elsif ($moveResult == 2) {
305
306                # Turn on lamp unless color is off.
307                if ($Color ne 'Off') {
308                   if (&SetSignalColor($Signal, 'Grn', $SignalChildPid,
309                                       $SignalData, 'semaphore')) {
310                      &DisplayError("SetSemaphoreSignal, SetSignalColor " .
311                                    "$Signal 'Grn' returned error.");
312                      return 1;
313                   }
314                   $$SemaphoreData{$Signal}{'Lamp'} = 'On';
315                }
316                $$SignalData{$Signal}{'Current'} = $Color;
317             }
318          }
319       }
320    }
321    else {
322       &DisplayError("SetSemaphoreSignal, invalid signal number: $Signal");
323       return 1;
324    }
325    return 0;
326 }
327
328 # ============================================================================
329 # FUNCTION:  SignalChildProcess
330 #
331 # DESCRIPTION:
332 #    This routine is launched as a child process during main program startup
333 #    and is used to communicate with the 74HC595 shift registers. This frees
334 #    the main code from the constant need to toggle the yellow signals between
335 #    red and green. The LEDs used in the signals should all be of similar
336 #    electrical specifications and color characterists.
337 #
338 #    Two time delays (select statements) are used to balance the red/green 'on'
339 #    time. This provides for coarse level adjustment of the yellow color for all
340 #    signals. These values should be set with the variable resistors on the shift
341 #    register board set to mid position. Then, the variable resistors are used
342 #    for fine adjustment of each signals yellow color.
343 #
344 #    The time delays further control the repetition rate of the while loop.
345 #    This rate should be just high enough to eliminate flicker when the yellow
346 #    color is displayed; about 25-30 cycles per second. The lowest possible
347 #    cycle rate is desired to minimize CPU loading by the while loop.
348 #
349 #    The while loop further optimizes itself by checking for any yellow signal
350 #    indications. Yellow signals, with opposite red/green registerBit variable
351 #    settings, will produce a non-zero result when XOR'd. When no yellow signals
352 #    are being displayed, the while loop repetition rate is reduced to 4 cycles
353 #    per second.
354 #
355 # CALLING SYNTAX:
356 #    $pid = fork { sub => \&SignalChildProcess, child_fh => "in socket",
357 #                  args => [ \%GpioData ] };
358 #
359 #       $GpioData          Pointer to the %GpioData hash.
360 #
```

```
#    The SuperForks 'child_fh' functionality is used for communication between
#    the parent and child processes. The parent sends new signal settings to the
#    child's stdin. The new data is stored in the child variables and used until
#    subsequently updated.
#
#    To minimize input processing within this subroutine, the data message must
#    be formatted as follows.
#
#    <sigMask>,<sigColor1>,<sigColor2>,<terminator>
#
#       <sigMask>    - 32 bit mask, all 1's, signal position two bits set to 0.
#       <sigColor1>  - 32 bit mask, all 1's, signal position set to color value.
#       <sigColor2>  - 32 bit mask, all 1's, signal position set to color value.
#       <terminator> - "-\n".
#
# SEND DATA TO CHILD:
#    Forks::Super::write_stdin($SignalChildPid, join(",", $sigMask, $sigColor1,
#                                                    $sigColor2, "-\n"));
#
# RETURNED VALUES:
#    PID of child process = Success, 0 = Error
#
# ACCESSED GLOBAL VARIABLES:
#    $main::ChildName
# ============================================================================
sub SignalChildProcess {
   my($GpioData) = @_;
   my($x, @buffer, $yellowSig);

# Default shift register bits.
   my($registerBits1) = 0x00000000;
   my($registerBits2) = 0x00000000;

   $main::ChildName = 'SignalChildProcess';
   &DisplayMessage("SignalChildProcess started.");

   while (1) {
      push(@buffer, <STDIN>);

      # Check for a new complete message and process if found.
      if ($buffer[0] =~ m/(.+?),(.+?),(.+?),-/) {
#         for ($x = 0; $x <= $#buffer; $x++) {
#            print "x: $x - $buffer[$x]";
#         }
         $registerBits1 = (($registerBits1 & $1) | $2);
         $registerBits2 = (($registerBits2 & $1) | $3);
         $yellowSig = $registerBits1 ^ $registerBits2;

#         &DisplayDebug(3, "SignalChildProcess, 1: " .
#                          sprintf("%0.32b", $1));
#         &DisplayDebug(3, "SignalChildProcess, 2: " .
#                          sprintf("%0.32b", $2));
#         &DisplayDebug(3, "SignalChildProcess, 3: " .
#                          sprintf("%0.32b", $3));
#         &DisplayDebug(1, "SignalChildProcess, registerBits1: " .
#                          sprintf("%0.32b", $registerBits1));
#         &DisplayDebug(1, "SignalChildProcess, registerBits2: " .
#                          sprintf("%0.32b", $registerBits2));
         splice(@buffer, 0, 1);    # Remove processed record.
      }
```

```
421
422         # Send data to 74HC595s - GPIO17_XLAT, GPIO23_OUTE, GPIO27_SCLK, GPIO22_DATA
423         for my $pos (reverse(0..31)) {
424            $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);       # Set SCLK low.
425            $$GpioData{'GPIO22_DATA'}{'Obj'}->write(($registerBits1 >> $pos) & 0x01);
426            $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(1);       # Set SCLK high
427         }
428         $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);          # Set SCLK low.
429         $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(1);          # Set XLAT high
430         $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(0);          # Set XLAT low.
431
432         sleep 0.25 unless ($yellowSig);
433         sleep 0.006;                          # Adjust for coarse yellow color.
434
435         for my $pos (reverse(0..31)) {
436            $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);       # Set SCLK low.
437            $$GpioData{'GPIO22_DATA'}{'Obj'}->write(($registerBits2 >> $pos) & 0x01);
438            $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(1);       # Set SCLK high
439         }
440         $$GpioData{'GPIO27_SCLK'}{'Obj'}->write(0);          # Set SCLK low.
441         $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(1);          # Set XLAT high
442         $$GpioData{'GPIO17_XLAT'}{'Obj'}->write(0);          # Set XLAT low.
443
444         sleep 0.25 unless ($yellowSig);
445         sleep 0.019;                          # Adjust for coarse yellow color.
446      }
447
448      &DisplayMessage("SignalChildProcess terminated.");
449      exit(0);
450   }
451
452   # ============================================================================
453   # FUNCTION:  TestSignals
454   #
455   # DESCRIPTION:
456   #    This routine cycles the specified signal range between the available colors.
457   #
458   # CALLING SYNTAX:
459   #    $result = &TestSignals($Range, $SignalChildPid, \%SignalData,
460   #                           \%GradeCrossingData, \%SemaphoreData, \%TurnoutData);
461   #
462   # ARGUMENTS:
463   #    $Range               Signal number or range to use.
464   #    $SignalChildPid      PID of child signal refresh process.
465   #    $SignalData          Pointer to %SignalData hash.
466   #    $GradeCrossingData   Pointer to %GradeCrossingData hash.
467   #    $SemaphoreData       Pointer to %SemaphoreData hash.
468   #    $TurnoutData         Pointer to %TurnoutData hash. (semaphore flag board)
469   #
470   # RETURNED VALUES:
471   #    0 = Success,  1 = Error.
472   #
473   # ACCESSED GLOBAL VARIABLES:
474   #    $main::MainRun
475   # ============================================================================
476   sub TestSignals {
477
478      my($Range, $SignalChildPid, $SignalData, $GradeCrossingData, $SemaphoreData,
479         $TurnoutData) = @_;
480      my($result, $signal, $start, $end, $nmbr, $color, @signalNumbers);
```

```
481     my($cntSignal) = scalar keys %$SignalData;
482     my(@signalList) = ();  my(@colorList) = ();
483     my($random, $gradecrossing) = (0,0);
484     my(%colorHash) = (1 => 'Red', 2 => 'Grn', 3 => 'Yel', 4 => 'Off');
485
486     &DisplayDebug(1, "TestSignals, Entry ... SignalChildPid: " .
487                      "$SignalChildPid   Range: '$Range'");
488
489     # ============================
490     # Set specified color and exit.
491
492     if ($Range =~ m/^(Red):(\d+)/i or $Range =~ m/^(Grn):(\d+)/i or
493         $Range =~ m/^(Yel):(\d+)/i or $Range =~ m/^(Off):(\d+)/i) {
494        $color = ucfirst(lc $1);
495        $signal = $2;
496        if ($signal > $cntSignal or $signal <= 0) {
497           &DisplayError("TestSignals, invalid signal number: $signal");
498           return 1;
499        }
500        $signal = "0${signal}" if (length($signal) == 1);
501        if (exists ($$SemaphoreData{$signal})) {
502           &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
503           while ($$SignalData{$signal}{Current} ne $color) {
504              return 1 if (&SetSemaphoreSignal($signal, $color, $SignalChildPid,
505                           $SignalData, $SemaphoreData, $TurnoutData));
506              sleep 0.5;                          # Wait for servo move.
507           }
508        }
509        else {
510           return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
511                                        $SignalData, ''));
512        }
513        &DisplayMessage("Signal $signal set to '$color'.");
514        exit(0);
515     }
516     elsif ($Range =~ m/^(Red).*/i or $Range =~ m/^(Grn).*/i or
517            $Range =~ m/^(Yel).*/i or $Range =~ m/^(Off).*/i) {
518        $color = ucfirst(lc $1);
519        foreach my $signal (1..12) {
520           $signal = "0${signal}" if (length($signal) == 1);
521           if (exists ($$SemaphoreData{$signal})) {
522              &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
523              while ($$SignalData{$signal}{Current} ne $color) {
524                 return 1 if (&SetSemaphoreSignal($signal, $color, $SignalChildPid,
525                              $SignalData, $SemaphoreData, $TurnoutData));
526                 sleep 0.5;                       # Wait for servo move.
527              }
528           }
529           else {
530              return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
531                                           $SignalData, ''));
532           }
533           &DisplayDebug(1, "TestSignals, Signal $signal is set to " .
534                            "$$SignalData{$signal}{Current}");
535        }
536        &DisplayMessage("All signals set to '$color'.");
537        exit(0);
538     }
539
540     # ============================
```

```
541     # Process special modifiers and then setup for looped testing.
542
543     if ($Range =~ m/r.*\d/i) {
544        $random = 1;
545        $Range =~ s/r//i;
546     }
547     if ($Range =~ m/g.*\d/i) {
548        $gradecrossing = 1;
549        $Range =~ s/g//i;
550        sleep 1;                # Give GcChildProcess time to start.
551     }
552
553     if ($Range =~ m/(\d+):(\d+)/) {   # Range specified.
554        $start = $1;
555        $end = $2;
556        if ($start > $end or $start <= 0 or $start > $cntSignal or $end <= 0 or
557            $end > $cntSignal) {
558           &DisplayError("TestSignals, invalid signal range: '$Range'" .
559                         "   cntSignal: $cntSignal");
560           return 1;
561        }
562        for ($signal = $start; $signal <= $end; $signal++) {
563           push (@signalList, $signal);
564        }
565     }
566     else {
567        @signalList = split(",", $Range);
568        foreach my $signal (@signalList) {
569           if ($signal !~ /^\d+$/ or $signal > $cntSignal or $signal <= 0) {
570              &DisplayError("TestSignals, invalid signal number: $signal");
571              return 1;
572           }
573        }
574     }
575
576     &DisplayDebug(1, "TestSignals, signalList: '@signalList'");
577
578     # ============================
579     # Begin looped testing.
580
581     while ($main::MainRun) {
582        # For random testing, we randomize the signalNumbers list and also the
583        # signal color. For non-random, we set each color.
584
585        if ($random == 1) {
586           &ShuffleArray(\@signalList);
587           foreach my $signal (@signalList) {
588              last unless ($main::MainRun);
589              $signal = "0${signal}" if (length($signal) == 1);
590              $color = $colorHash{(int(rand(4))+1)};
591              if ($gradecrossing == 1) {
592                 if ($color eq 'Grn') {
593                    Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
594                                             'start:apr');
595                 }
596                 elsif ($color eq 'Yel') {
597                    Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
598                                             'start:road');
599                 }
600                 elsif ($color eq 'Off') {
```

```
                  Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
                                            'stop');
               }
               elsif ($color eq 'Red') {
                  Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
                                            'stop');
               }
            }
            &DisplayMessage("TestSignals, Signal: $signal   Color: $color");
            if (exists ($$SemaphoreData{$signal})) {
               &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
               while ($$SignalData{$signal}{Current} ne $color) {
                  return 1 if (&SetSemaphoreSignal($signal, $color,
                               $SignalChildPid, $SignalData, $SemaphoreData,
                               $TurnoutData));
                  sleep 0.5;                    # Wait for servo move.
               }
            }
            else {
               return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
                                            $SignalData, ''));
            }
            &DisplayDebug(1, "TestSignals, Signal $signal is set to " .
                             "$$SignalData{$signal}{Current}");
            sleep 0.5;
         }
      }
      else {
         # Create colorList test sequence.
         if ($#colorList < 0) {
            foreach my $nmbr (sort keys %colorHash) {
               push (@colorList, $colorHash{$nmbr});
            }
         }
         foreach my $color (@colorList) {
            if ($gradecrossing == 1) {
               if ($color eq 'Grn') {
                  Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
                                            'start:apr');
               }
               elsif ($color eq 'Yel') {
                  Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
                                            'start:road');
               }
               elsif ($color eq 'Off') {
                  Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
                                            'stop');
               }
               elsif ($color eq 'Red') {
                  Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
                                            'stop');
               }
            }
            foreach my $signal (@signalList) {
               last unless ($main::MainRun);
               $signal = "0${signal}" if (length($signal) == 1);
               &DisplayMessage("TestSignals, Signal: $signal   Color: $color");
               if (exists ($$SemaphoreData{$signal})) {
                  &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
                  while ($$SignalData{$signal}{Current} ne $color) {
```

```
                    return 1 if (&SetSemaphoreSignal($signal, $color,
                                 $SignalChildPid, $SignalData, $SemaphoreData,
                                 $TurnoutData));
                    sleep 0.5;                # Wait for servo move.
                 }
              }
              else {
                 return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
                                              $SignalData, ''));
              }
              &DisplayDebug(1, "TestSignals, Signal $signal is set" .
                               " to $$SignalData{$signal}{Current}");
              sleep 0.75;
           }
        }
     }
     sleep 2;    # Show set signal color(s) for this time delay.
  }

  # Set signal related servos to their open position.
  foreach my $nmbr (keys(%$TurnoutData)) {
     if ($$TurnoutData{$nmbr}{'Id'} =~ m/semaphore/i or
         $$TurnoutData{$nmbr}{'Id'} =~ m/gate/i) {
        $result = &MoveTurnout('Open', $nmbr, $TurnoutData);
        while ($$TurnoutData{$nmbr}{'Pid'}) {
           sleep 0.25;
        }
     }
  }
  return 0;
}

return 1;
```