

```

1 # =====
2 # FILE: DnB_Signal.pm
3 #
4 # SERVICES: DnB SIGNAL FUNCTIONS
5 #
6 # DESCRIPTION:
7 #     This perl module provides signal related functions used by the DnB model
8 #     railroad control program.
9 #
10 # PERL VERSION: 5.24.1
11 #
12 # =====
13 use strict;
14 #
15 # Package Declaration
16 #
17 package DnB_Signal;
18 require Exporter;
19 our @ISA = qw(Exporter);
20
21 our @EXPORT = qw(
22     Init_SignalDriver
23     SetSignalColor
24     SetSemaphoreSignal
25     SignalChildProcess
26     TestSignals
27 );
28
29 use DnB_Turnout;
30 use DnB_Message;
31 use Forks::Super;
32 use Time::HiRes qw(sleep);
33
34 # =====
35 # FUNCTION: Init_SignalDriver
36 #
37 # DESCRIPTION:
38 #     This routine initializes the GPIO pins associated with the LED driver on
39 #     the DnB model railroad. A shift register utilizing multiple 74HC595 chips
40 #     is used. Data is shifted in serially using GPIO pins connected to the data
41 #     and clock inputs of the shift register.
42 #
43 #     A second group of GPIOs is used to control the track power polarity relays.
44 #
45 # CALLING SYNTAX:
46 #     $result = &Init_SignalDriver(\%GpioData, $RegisterLength);
47 #
48 # ARGUMENTS:
49 #     $GpioData          Pointer to GPIO data.
50 #     $RegisterLength    Shift register bit length.
51 #
52 # RETURNED VALUES:
53 #     0 = Success, 1 = Error.
54 #
55 # ACCESSED GLOBAL VARIABLES:
56 #     None.
57 # =====
58 sub Init_SignalDriver {
59     my($GpioData, $RegisterLength) = @_;
60     my($x, $pin);

```

```

61     &DisplayDebug(2, "Init_SignalDriver, RegisterLength: $RegisterLength");
62
63 # Create a Raspberry Pi object for each GPIO and set to defaults.
64
65 foreach my $gpio (sort keys %$GpioData) {
66     if ($$GpioData{$gpio}{'Obj'} == 0) {
67         if ($gpio =~ m/^GPIO(\d*)$/) {
68             $pin = $1;
69             $$GpioData{$gpio}{'Obj'} = RPi::Pin->new($pin);
70             if ($$GpioData{$gpio}{'Obj'} != 0) {
71                 &DisplayDebug(1, "Init_SignalDriver, $gpio object " .
72                             "successfully created.");
73                 $$GpioData{$gpio}{'Obj'}->mode($$GpioData{$gpio}{'Mode'});
74                 if ($$GpioData{$gpio}{'Mode'} == 0) {
75                     # 0=None, 1=Pulldown, 2=Pullup
76                     $$GpioData{$gpio}{'Obj'}->pull(2);    # Enable pullup on pin.
77                 }
78                 elsif ($$GpioData{$gpio}{'Mode'} == 1) {
79                     $$GpioData{$gpio}{'Obj'}->write(0);           # Set GPIO low.
80                 }
81             }
82         }
83         else {
84             &DisplayError("Init_SignalDriver, failed to create " .
85                         "$gpio object. $!");
86             return 1;
87         }
88     }
89     else {
90         &DisplayError("Init_SignalDriver, failed to parse pin " .
91                         "number from '$gpio'.");
92         return 1;
93     }
94 }
95 else {
96     &DisplayWarning("Init_SignalDriver, $gpio object already active.");
97 }
98 }
99
100 # Test toggle.
101 # while (1) {
102 #     foreach my $gpio (sort keys %$GpioData) {
103 #         $$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 "") {

```

```

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 }

# =====
# FUNCTION: SetSemaphoreSignal
# 
# DESCRIPTION:
#     This routine sets the specified Semaphore signal to the specified color.
#     SetSignalColor is called to set the lamp on (red) or off as required.
#     MoveTurnout is called to position the servo attached to the semaphore flag
#     board.
#
#     This routine is call for each iteration of the main loop until the 'Position'
#     value for the semaphore in SemaphoreData is set to the necessary color.
#
# CALLING SYNTAX:
#     $result = &SetSemaphoreSignal($Signal, $Color, $SignalChildPid, \%SignalData,
#                                     \%SemaphoreData, \%TurnoutData);
#
# ARGUMENTS:
#     $Signal          Signal number to set.
#     $Color           Signal color, 'Red', 'Grn', 'Yel', or 'Off'
#     $SignalChildPid PID of child signal refresh process.
#     $SignalData      Pointer to %SignalData hash.
#     $SemaphoreData   Pointer to %SemaphoreData hash.
#     $TurnoutData    Pointer to %TurnoutData hash.
#
# RETURNED VALUES:
#     0 = Success, 1 = Error.
#
# ACCESSED GLOBAL VARIABLES:

```

```

241 #     None.
242 # =====
243 sub SetSemaphoreSignal {
244     my($Signal, $Color, $SignalChildPid, $SignalData, $SemaphoreData,
245         $TurnoutData) = @_;
246     my($moveResult, $servo);
247     my(%flagPosition) = ('Grn' => 'Open', 'Yel' => 'Middle', 'Red' => 'Close',
248                           'Off' => 'Open');
249
250     &DisplayDebug(1, "SetSemaphoreSignal, Signal: $Signal Color: $Color");
251
252     if ($Signal ne "" and exists($$SemaphoreData{$Signal})) {
253         $servo = $$SemaphoreData{$Signal}{'Servo'};
254         if ($$SemaphoreData{$Signal}{'InMotion'} == 1) {
255             if ($$TurnoutData{$servo}{Pid} == 0) {
256                 $$SemaphoreData{$Signal}{'InMotion'} = 0;
257                 &DisplayDebug(2, "SetSemaphoreSignal, semaphore $Signal ".
258                               "move completed.");
259
260             # Turn on lamp unless color is off.
261             if ($Color ne 'Off') {
262                 if (&SetSignalColor($Signal, 'Grn', $SignalChildPid, $SignalData,
263                                     'semaphore')) {
264                     &DisplayError("SetSemaphoreSignal, SetSignalColor ".
265                               "$Signal 'Grn' returned error.");
266                     return 1;
267                 }
268                 $$SemaphoreData{$Signal}{'Lamp'} = 'On';
269             }
270             $$SignalData{$Signal}{'Current'} = $Color;
271             &DisplayMessage("SetSemaphoreSignal, semaphore $Signal ".
272                             "set to $Color.");
273         }
274     }
275     else {
276         if ($$SignalData{$Signal}{'Current'} ne $Color) {
277             &DisplayDebug(1, "SetSemaphoreSignal, moving semaphore ".
278                           "$Signal to position $Color");
279
280         # Turn off lamp.
281         if (&SetSignalColor($Signal, 'Off', $SignalChildPid, $SignalData,
282                           'semaphore')) {
283             &DisplayError("SetSemaphoreSignal, SetSignalColor ".
284                           "$Signal 'Off' returned error.");
285             return 1;
286         }
287         $$SemaphoreData{$Signal}{'Lamp'} = 'Off';
288
289         # Move semaphore flag board to requested position.
290         $moveResult = &MoveTurnout($flagPosition{$Color}, $servo, $TurnoutData);
291         if ($moveResult == 0) {
292             $$SemaphoreData{$Signal}{'InMotion'} = 1;
293             &DisplayDebug(2, "SetSemaphoreSignal, semaphore ".
294                           "$Signal move inprogress.");
295         }
296         elsif ($moveResult == 1) {
297             &DisplayError("SetSemaphoreSignal, MoveTurnout $servo ".
298                           "$flagPosition{$Color} . '' returned error.");
299             return 1;
300         }

```

```

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 characteristics.
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

```

```

361 #   The SuperForks 'child_fh' functionality is used for communication between
362 #   the parent and child processes. The parent sends new signal settings to the
363 #   child's stdin. The new data is stored in the child variables and used until
364 #   subsequently updated.
365 #
366 #   To minimize input processing within this subroutine, the data message must
367 #   be formatted as follows.
368 #
369 #   <sigMask>,<sigColor1>,<sigColor2>,<terminator>
370 #
371 #       <sigMask> - 32 bit mask, all 1's, signal position two bits set to 0.
372 #       <sigColor1> - 32 bit mask, all 1's, signal position set to color value.
373 #       <sigColor2> - 32 bit mask, all 1's, signal position set to color value.
374 #       <terminator> - "-\n".
375 #
376 # SEND DATA TO CHILD:
377 #   Forks::Super::write_stdin($SignalChildPid, join(", ", $sigMask, $sigColor1,
378 #                                         $sigColor2, "-\n"));
379 #
380 # RETURNED VALUES:
381 #   PID of child process = Success, 0 = Error
382 #
383 # ACCESSED GLOBAL VARIABLES:
384 #   $main::ChildName
385 # =====
386 sub SignalChildProcess {
387     my($GpioData) = @_;
388     my($x, @buffer, $yellowSig);
389
390     # Default shift register bits.
391     my($registerBits1) = 0x00000000;
392     my($registerBits2) = 0x00000000;
393
394     $main::ChildName = 'SignalChildProcess';
395     &DisplayMessage("SignalChildProcess started.");
396
397     while (1) {
398         push(@buffer, <STDIN>);
399
400         # Check for a new complete message and process if found.
401         if ($buffer[0] =~ m/(.+?),( .+?),( .+?),-/) {
402             for ($x = 0; $x <= $#buffer; $x++) {
403                 print "x: $x - $buffer[$x]";
404             }
405             $registerBits1 = (($registerBits1 & $1) | $2);
406             $registerBits2 = (($registerBits2 & $1) | $3);
407             $yellowSig = $registerBits1 ^ $registerBits2;
408
409             &DisplayDebug(3, "SignalChildProcess, 1: " .
410                         sprintf("%0.32b", $1));
411             &DisplayDebug(3, "SignalChildProcess, 2: " .
412                         sprintf("%0.32b", $2));
413             &DisplayDebug(3, "SignalChildProcess, 3: " .
414                         sprintf("%0.32b", $3));
415             &DisplayDebug(1, "SignalChildProcess, registerBits1: " .
416                         sprintf("%0.32b", $registerBits1));
417             &DisplayDebug(1, "SignalChildProcess, registerBits2: " .
418                         sprintf("%0.32b", $registerBits2));
419             splice(@buffer, 0, 1);    # Remove processed record.
420         }
}

```

```

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 # =====

```

```

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') {

```

```

601             Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
602                                         'stop');
603         }
604     elsif ($color eq 'Red') {
605         Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
606                                         'stop');
607     }
608 }
609 &DisplayMessage("TestSignals, Signal: $signal Color: $color");
610 if (exists ($$SemaphoreData{$signal})) {
611     &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
612     while ($$SignalData{$signal}{Current} ne $color) {
613         return 1 if (&SetSemaphoreSignal($signal, $color,
614                                         $SignalChildPid, $SignalData, $$SemaphoreData,
615                                         $TurnoutData));
616         sleep 0.5;                                # Wait for servo move.
617     }
618 }
619 else {
620     return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
621                                         $SignalData, ''));
622 }
623 &DisplayDebug(1, "TestSignals, Signal $signal is set to ".
624                 "$$SignalData{$signal}{Current}");
625 sleep 0.5;
626 }
627 }
628 else {
629     # Create colorList test sequence.
630     if ($#colorList < 0) {
631         foreach my $nmbr (sort keys %colorHash) {
632             push (@colorList, $colorHash{$nmbr});
633         }
634     }
635     foreach my $color (@colorList) {
636         if ($gradecrossing == 1) {
637             if ($color eq 'Grn') {
638                 Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
639                                         'start:apr');
640             }
641             elsif ($color eq 'Yel') {
642                 Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
643                                         'start:road');
644             }
645             elsif ($color eq 'Off') {
646                 Forks::Super::write_stdin($$GradeCrossingData{'01'}{'Pid'},
647                                         'stop');
648             }
649             elsif ($color eq 'Red') {
650                 Forks::Super::write_stdin($$GradeCrossingData{'02'}{'Pid'},
651                                         'stop');
652             }
653         }
654     foreach my $signal (@signalList) {
655         last unless ($main::MainRun);
656         $signal = "0${signal}" if (length($signal) == 1);
657         &DisplayMessage("TestSignals, Signal: $signal Color: $color");
658         if (exists ($$SemaphoreData{$signal})) {
659             &DisplayDebug(1, "TestSignals, Semaphore signal: $signal");
660             while ($$SignalData{$signal}{Current} ne $color) {

```

```

661             return 1 if (&SetSemaphoreSignal($signal, $color,
662                                         $SignalChildPid, $SignalData, $SemaphoreData,
663                                         $TurnoutData));
664             sleep 0.5;                                # Wait for servo move.
665         }
666     }
667     else {
668         return 1 if (&SetSignalColor($signal, $color, $SignalChildPid,
669                                         $SignalData, '1'));
670     }
671     &DisplayDebug(1, "TestSignals, Signal $signal is set" .
672                   " to $$SignalData{$signal}{Current}");
673     sleep 0.75;
674   }
675 }
676 sleep 2;    # Show set signal color(s) for this time delay.
677 }

# Set signal related servos to their open position.
680 foreach my $nmbr (keys(%$TurnoutData)) {
681     if ($$TurnoutData{$nmbr}{'Id'} =~ m_semaphore/i or
682         $$TurnoutData{$nmbr}{'Id'} =~ m_gate/i) {
683         $result = &MoveTurnout('Open', $nmbr, $TurnoutData);
684         while ($$TurnoutData{$nmbr}{'Pid'}) {
685             sleep 0.25;
686         }
687     }
688 }
689 }
690 return 0;
691 }

return 1;
693
694

```