

```

1  # =====
2  # FILE: DnB_Turnout.pm                                     8/14/2020
3  #
4  # SERVICES:  DnB TURNOUT FUNCTIONS
5  #
6  # DESCRIPTION:
7  #   This perl module provides turnout 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_Turnout;
18 require Exporter;
19 our @ISA = qw(Exporter);
20
21 our @EXPORT = qw(
22     I2C_InitServoDriver
23     ProcessTurnoutFile
24     InitTurnouts
25     MoveTurnout
26     SetTurnoutPosition
27     GetTemperature
28     TestServoAdjust
29     TestTurnouts
30 );
31
32 use DnB_Message;
33 use Forks::Super;
34 use POSIX 'WNOHANG';
35 use Time::HiRes qw(sleep);
36
37 # =====
38 # FUNCTION:  I2C_InitServoDriver
39 #
40 # DESCRIPTION:
41 #   This routine initializes the turnout servo I2C driver boards on the DnB
42 #   model railroad. It sets parameters that are common to all servo ports. The
43 #   Adafruit 16 Channel Servo Driver utilizes the PCA9685 chip. The pre_scale
44 #   calculation is from the PCA9685 documentation.
45 #
46 #   Initialization sequence.
47 #       1. Get current ModeReg1.
48 #       2. Put PCA9685 into sleep mode.
49 #       3. Set servo refresh rate.
50 #       4. Normal mode + register auto increment.
51 #       5. Put PCA9685 into normal mode.
52 #
53 # CALLING SYNTAX:
54 #   $result = &I2C_InitServoDriver($BoardNmbr, $I2C_Address);
55 #
56 # ARGUMENTS:
57 #   $BoardNmbr      Drive board number being initialized.
58 #   $I2C_Address    I2C Address
59 #
60 # RETURNED VALUES:

```

```

61 # 0 = Success, 1 = Error.
62 #
63 # ACCESSED GLOBAL VARIABLES:
64 # None.
65 # =====
66 sub I2C_InitServoDriver {
67
68     my($BoardNmbr, $I2C_Address) = @_;
69     my($result, $driver, $mode_data);
70
71     my($minAddr, $maxAddr) = (0x40, 0x7F); # AdaFruit 16 Channel PWM board range.
72     my(%PCA9685) = ('ModeReg1' => 0x00, 'ModeReg2' => 0x01, 'AllLedOffH' => 0xFD,
73                     'PreScale' => 0xFE);
74     my($normal_mode) = 0xEF; my($sleep_mode) = 0x10; my($auto_inc) = 0xA1;
75
76     my($freq) = 105; # Refresh rate; 105 = 300-900 SG90 min/max position.
77
78     my($pre_scale) = int((25000000.0 / (4096 * $freq)) - 1);
79
80     &DisplayDebug(2, "I2C_InitServoDriver, BoardNmbr: $BoardNmbr " .
81                   "I2C_Address: $I2C_Address pre_scale: $pre_scale");
82
83     # Validate that address is within the Adafruit 16-channel driver range.
84     if ($I2C_Address >= $minAddr and $I2C_Address <= $maxAddr) {
85         $driver = RPi::I2C->new($I2C_Address);
86         unless ($driver->check_device($I2C_Address)) {
87             &DisplayError("I2C_InitServoDriver, Failed to initialize " .
88                           "I2C address: " . sprintf("0x%.2x", $I2C_Address));
89             return 1;
90         }
91         $driver->write_byte(0x10, $PCA9685{'AllLedOffH'}); # Orderly shutdown.
92         sleep 0.01; # Wait for channels to stop.
93         $mode_data = $driver->read_byte($PCA9685{'ModeReg1'});
94         $driver->write_byte(($mode_data | $sleep_mode), $PCA9685{'ModeReg1'});
95         $driver->write_byte($pre_scale, $PCA9685{'PreScale'});
96         $mode_data = ($mode_data & $normal_mode) | $auto_inc;
97         $driver->write_byte(($mode_data), $PCA9685{'ModeReg1'});
98         &DisplayDebug(2, "I2C_InitServoDriver, PreScale: " .
99                       $driver->read_byte($PCA9685{'PreScale'}));
100         undef($driver);
101     }
102     else {
103         &DisplayError("I2C_InitServoDriver, Invalid I2C address: " .
104                       "$I2C_Address Board: $BoardNmbr");
105         return 1;
106     }
107     return 0;
108 }
109
110 # =====
111 # FUNCTION: ProcessTurnoutFile
112 #
113 # DESCRIPTION:
114 # This routine reads or writes the specified turnout data file. Used to
115 # retain turnout operational data between program starts.
116 #
117 # CALLING SYNTAX:
118 # $result = &ProcessTurnoutFile($FileName, $Function, \%TurnoutData);
119 #
120 # ARGUMENTS:

```

```

121 # $FileName      File to Read/Write
122 # $Function       "Read" or "Write"
123 # $TurnoutData    Pointer to %TurnoutData hash.
124 #
125 # RETURNED VALUES:
126 #   0 = Success,  1 = Error.
127 #
128 # ACCESSED GLOBAL VARIABLES:
129 #   None.
130 # =====
131 sub ProcessTurnoutFile {
132
133     my($FileName, $Function, $TurnoutData) = @_;
134     my($turnout, $rec);
135     my(@fileData) = ();
136
137     my(@keyList) = ('Pid', 'Addr', 'Port', 'Pos', 'Rate', 'Open', 'Middle', 'Close',
138                    'MinPos', 'MaxPos', 'Id');
139
140     &DisplayDebug(2, "ProcessTurnoutFile, Function: $Function    " .
141                  "keyList: '@keyList'");
142
143     if ($Function =~ m/^\^Read$/i) {
144         if (-e $FileName) {
145             if (&ReadFile($FileName, \@fileData)) {
146                 &DisplayWarning("ProcessTurnoutData, Using default " .
147                               "turnout data.");
148             }
149         } else {
150             %$TurnoutData = ();
151             foreach my $rec (@fileData) {
152                 next if ($rec =~ m/^\^s*$/ or $rec =~ m/^\^#/);
153                 if ($rec =~ m/Turnout:\s*(\d+)/i) {
154                     $turnout = sprintf("%2s", $1);
155                     $$TurnoutData{$turnout}{'Pid'} = 0;
156                     foreach my $key (@keyList) {
157                         if ($key eq 'Id') {
158                             if ($rec =~ m/$key:(.+)/) {
159                                 $$TurnoutData{$turnout}{$key} = &Trim($1);
160                             }
161                         } else {
162                             &DisplayWarning("ProcessTurnoutData, " .
163                                             "'$key' not found: '$rec'");
164                             next;
165                         }
166                     }
167                 } else {
168                     if ($rec =~ m/$key:\s*(\d+)/i) {
169                         $$TurnoutData{$turnout}{$key} = $1;
170                     }
171                     else {
172                         &DisplayWarning("ProcessTurnoutData, " .
173                                         "'$key' not found: '$rec'");
174                     }
175                 }
176             }
177             &DisplayDebug(2, "ProcessTurnoutFile, " .
178                          "Turnout: $turnout    key: $key    value: " .
179                          "$$TurnoutData{$turnout}{$key}");
180         }
181     }

```

```

181         }
182         else {
183             &DisplayWarning("ProcessTurnoutData, 'Turnout' key " .
184                 "not found: '$rec'");
185         }
186     }
187 }
188 $rec = scalar keys %$TurnoutData;
189 &DisplayDebug(1, "ProcessTurnoutFile, Function: $Function " .
190     "$rec turnout records.");
191 }
192 else {
193     &DisplayWarning("ProcessTurnoutData: File not found: $FileName.");
194     &DisplayWarning("ProcessTurnoutData: Using default turnout data.");
195 }
196 }
197 elsif ($Function =~ m/^\Write$/i) {
198     push (@fileData, "# =====");
199     push (@fileData, "# Turnout data file. Loaded during program start.");
200     push (@fileData, "# Edited values will be used upon next start. See");
201     push (@fileData, "# DnB.pl 'Turnout Related Data' section for more ");
202     push (@fileData, "# information.");
203     push (@fileData, "# =====");
204
205     $rec = scalar keys %$TurnoutData;
206     &DisplayDebug(1, "ProcessTurnoutFile, Function: $Function $rec " .
207         "turnout records.");
208
209     foreach my $turnout (sort keys %$TurnoutData) {
210         next if ($turnout =~ m/^\s*$/ or $turnout eq '00');
211         $rec = join(":", "Turnout", $turnout);
212         $$TurnoutData{$turnout}{'Pid'} = 0;
213         foreach my $key (@keyList) {
214             $rec = join(" ", $rec, join(":", $key,
215                 $$TurnoutData{$turnout}{$key}));
216         }
217         push (@fileData, $rec);
218         &DisplayDebug(2, "ProcessTurnoutFile, $Function: $rec");
219     }
220     &WriteFile($FileName, \@fileData);
221 }
222 else {
223     &DisplayWarning("ProcessTurnoutData, Unsupported function: $Function");
224 }
225 return 0;
226 }
227
228 # =====
229 # FUNCTION:  InitTurnouts
230 #
231 # DESCRIPTION:
232 #   Called once during DnB startup, this routine initializes all turnouts to
233 #   the PWM position specified in %TurnoutData. This ensures that all servo
234 #   driver board channels are synchronized to the %TurnoutData specified PWM
235 #   position.
236 #
237 #   A check of the %TurnoutData PWM values is performed since these values are
238 #   normally loaded from the user editable TurnoutDataFile. If an out-of-range
239 #   value is detected, initialization is aborted and an error is returned.
240 #

```

```

241 # If optional data is specified, the servo is set to the specified PWM
242 # position. This position is used for physical turnout point adjustment.
243 #
244 # CALLING SYNTAX:
245 # $result = &InitTurnouts(\%ServoBoardAddress, \%TurnoutData, $Turnout,
246 # $Position);
247 #
248 # ARGUMENTS:
249 # $ServoBoardAddress Pointer to %ServoBoardAddress hash.
250 # $TurnoutData Pointer to %TurnoutData hash.
251 # $Turnout Optional; turnout to position.
252 # $Position Optional; position to set.
253 #
254 # RETURNED VALUES:
255 # 0 = Success, 1 = Error.
256 #
257 # ACCESSED GLOBAL VARIABLES:
258 # None.
259 # =====
260 sub InitTurnouts {
261     my($ServoBoardAddress, $TurnoutData, $Turnout, $Position) = @_;
262     my($board, $pwm);
263     my($min,$max) = (300,900); # Absolute PWM values.
264     my($rmin,$rmax) = (1,850); # Absolute Rate values.
265     my($fail) = 0;
266
267     # Processing for -o, -m, and -c CLI options.
268     if ($Turnout ne '') {
269         $Turnout = "0${Turnout}" if (length($Turnout) == 1);
270         if ($Position ne 'Open' and $Position ne 'Close') {
271             $Position = 'Middle';
272         }
273     }
274
275     # Validate the %TurnoutData PWM values.
276     &DisplayMessage("Validate turnout PWM working values ...");
277     foreach my $tNmbr (sort keys %$TurnoutData) {
278         next if ($tNmbr eq '00'); # Skip temperature adjustment data.
279         foreach my $pos ('MinPos', 'MaxPos', 'Open', 'Middle', 'Close', 'Pos') {
280             $pwm = $$TurnoutData{$tNmbr}{$pos};
281             if ($pwm < $min or $pwm > $max) {
282                 &DisplayError("InitTurnouts, turnout $tNmbr $pos " .
283                     "value out of range: $pwm");
284                 $fail = 1;
285             }
286             elsif ($pwm < $$TurnoutData{$tNmbr}{'MinPos'} or
287                 $pwm > $$TurnoutData{$tNmbr}{'MaxPos'}) {
288                 &DisplayError("InitTurnouts, turnout $tNmbr $pos " .
289                     "value outside of min/max limit: $pwm");
290                 $fail = 1;
291             }
292         }
293         $pwm = $$TurnoutData{$tNmbr}{'Rate'};
294         if ($pwm < $rmin or $pwm > $rmax) {
295             &DisplayError("InitTurnouts, turnout $tNmbr Rate " .
296                 "value out of range: $pwm");
297             $fail = 1;
298         }
299     }
300     return 1 if ($fail == 1); # Error return if failure.

```

```

301 # Initialize servo channel on the driver boards.
302
303 for ($board = 1; $board <= scalar keys(%$ServoBoardAddress); $board++) {
304     if ($$ServoBoardAddress{$board} == 0) {
305         &DisplayDebug(1, "InitTurnouts, Skip board $board " .
306             "I2C_Address 0, code debug.");
307         next;
308     }
309     &DisplayMessage("Initializing turnout I2C board $board ...");
310     return 1 if (&I2C_InitServoDriver($board, $$ServoBoardAddress{$board}));
311
312     &DisplayMessage("Initializing turnout positions on board $board ...");
313
314     foreach my $tNmbr (sort keys %$TurnoutData) {
315         next if ($tNmbr eq '00'); # Skip temperature adjustment data.
316         if ($$TurnoutData{$tNmbr}{'Addr'} == $$ServoBoardAddress{$board}) {
317             if ($Turnout eq '00' or $Turnout eq $tNmbr) {
318                 $$TurnoutData{$tNmbr}{'Pos'} = $$TurnoutData{$tNmbr}{'Position'};
319             }
320
321             if (&SetTurnoutPosition($$TurnoutData{$tNmbr}{'Pos'}, $tNmbr,
322                 $TurnoutData)) {
323                 &DisplayWarning("InitTurnouts, Failed to set " .
324                     "turnout. board $board Turnout: $tNmbr" .
325                     "Position: $$TurnoutData{$tNmbr}{'Pos'}");
326                 $fail = 1;
327             }
328
329             $$TurnoutData{$tNmbr}{'Pid'} = 0; # Ensure the Pid value is 0.
330             sleep 0.1; # Delay so we don't overtax
331                     # the servo power supply.
332         }
333     }
334     &DisplayMessage("All board $board turnouts initialized.");
335 }
336 if ($Turnout ne '') {
337     if ($Turnout eq '00') {
338         &DisplayMessage("All turnouts set to $Position position.");
339     }
340     else {
341         &DisplayMessage("Turnout $Turnout set to $Position position.");
342     }
343 }
344 return 1 if ($fail == 1); # Error return if failure.
345 return 0;
346 }
347
348 # =====
349 # FUNCTION: MoveTurnout
350 #
351 # DESCRIPTION:
352 # This routine moves the turnout servo using the specified data. It is used
353 # to perform a slow motion position change. This is done by forking to a
354 # child process and calling SetTurnoutPosition 50 times a second until the
355 # move is complete. Each call positions the turnout servo toward the final
356 # position by a move step amount ('Rate'/50). Once the move is completed,
357 # the turnout position is updated in the TurnoutData hash and the child
358 # exits. A 'Rate' value of 450 positions the turnout from Open (350) to
359 # Close (850) in about 1.1 seconds.
360 #

```

```

361 # CALLING SYNTAX:
362 #   $result = &MoveTurnout($Function, $TurnoutNمبر, \%TurnoutData);
363 #
364 # ARGUMENTS:
365 #   $Function      'Open', 'Middle', or 'Close'.
366 #   $TurnoutNمبر    Turnout number; two digit hash index.
367 #   $TurnoutData    Pointer to TurnoutData hash.
368 #
369 # RETURNED VALUES:
370 #   0 = Success,  1 = Error, 2 = Already in position.
371 #
372 # ACCESSED GLOBAL VARIABLES:
373 #   None.
374 # =====
375 sub MoveTurnout {
376     my($Function, $TurnoutNمبر, $TurnoutData) = @_;
377     my($result, $pwmCurrent, $pwmFinal, $moveRate, $moveStep, $pid, $adjust);
378     my($noAdj);
379     my($timeout) = 40;    # Wait 10 seconds (40/.25) for move to complete.
380
381     &DisplayDebug(2, "MoveTurnout, Entry ...   $Function $TurnoutNمبر");
382
383     if ($TurnoutNمبر ne "") {
384         if ($Function =~ m/Open/i) {
385             $pwmFinal = $$TurnoutData{$TurnoutNمبر}{'Open'};
386         }
387         elsif ($Function =~ m/Middle/i) {
388             $pwmFinal = $$TurnoutData{$TurnoutNمبر}{'Middle'};
389         }
390         elsif ($Function =~ m/Close/i) {
391             $pwmFinal = $$TurnoutData{$TurnoutNمبر}{'Close'};
392         }
393         else {
394             &DisplayError("MoveTurnout, invalid function: '$Function'");
395             return 1;
396         }
397
398         # If gate or semaphore servo, adjust $pwmFinal for temperature.
399         if ($$TurnoutData{$TurnoutNمبر}{'Id'} =~ m/semaphore/i or
400             $$TurnoutData{$TurnoutNمبر}{'Id'} =~ m/gate/i) {
401             if ($$TurnoutData{'00'}{'Temperature'} > 0 and
402                 $$TurnoutData{'00'}{'Temperature'} < 38) {
403                 $noAdj = $pwmFinal;    # Used only for debug message.
404                 # 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37    degree C
405                 # -----
406                 # -8 -7 -6 -5 -4 -3 -2 -1 0 +1 +2 +3 +4 +5 +6 +7 +8    -2 divisor
407                 # -6 -5 -4 -4 -3 -2 -1 0 0 0 +1 +2 +3 +4 +4 +5 +6    -2.5 divisor
408                 # -5 -4 -4 -3 -2 -2 -1 0 0 0 +1 +2 +2 +3 +4 +4 +5    -3 divisor
409                 # -4 -3 -3 -2 -2 -1 -1 0 0 0 +1 +1 +2 +2 +3 +3 +4    -4 divisor
410                 # -----
411                 # Change divisor (-3) to increase/decrease adjustment value.
412                 # Change constant (21) to shift center point temperature; the
413                 # ambient temperature at time of physical position adjustment.
414                 # Note: TurnoutData MinPos and MaxPos will limit this code's
415                 # adjustment if set too close to Open/Close value.
416                 $adjust = int((21 - $$TurnoutData{'00'}{'Temperature'}) / -3);
417                 &DisplayDebug(1, "MoveTurnout, servo: $TurnoutNمبر    "
418                     "adjust: $adjust");
419
420                 # Application of adjustment is dependent on close direction.

```

```

421         if ($$TurnoutData{$TurnoutNmbr}{'Open'} >
422             $$TurnoutData{$TurnoutNmbr}{'Close'}) {
423             $pwmFinal += $adjust;
424         }
425         else {
426             $pwmFinal -= $adjust;
427         }
428         &DisplayDebug(1, "MoveTurnout, noAdj: $noAdj    adjusted: $pwmFinal");
429     }
430 }
431
432 # Make sure the requested move will not exceed a min/max limit.
433 $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'MinPos'}
434     if ($pwmFinal < $$TurnoutData{$TurnoutNmbr}{'MinPos'});
435 $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'MaxPos'}
436     if ($pwmFinal > $$TurnoutData{$TurnoutNmbr}{'MaxPos'});
437
438 # Check and wait for turnout to be idle.
439 while ($$TurnoutData{$TurnoutNmbr}{'Pid'} > 0 and $timeout > 0) {
440     if (($timeout % 4) == 0) {
441         &DisplayDebug(2, "MoveTurnout, waiting for previous move " .
442             "to complete. timeout: $timeout    Pid: " .
443             "$$TurnoutData{$TurnoutNmbr}{'Pid'}    Pos: " .
444             "$$TurnoutData{$TurnoutNmbr}{'Pos'}");
445     }
446     $timeout--;
447     sleep 0.25;          # Wait quarter sec.
448 }
449
450 # Abort turnout move if still active.
451 if ($$TurnoutData{$TurnoutNmbr}{'Pid'} > 0) {
452     &DisplayError("MoveTurnout, Turnout $TurnoutNmbr, Previous " .
453         "move still in progress, pid: " .
454         "$$TurnoutData{$TurnoutNmbr}{'Pid'}.");
455
456     # Check if the process is running, $result == 0. If so, kill it.
457     # Cleanup state data and continue new turnout move.
458     $result = waitpid($$TurnoutData{$TurnoutNmbr}{'Pid'}, WNOHANG);
459     system("kill -9 $$TurnoutData{$TurnoutNmbr}{'Pid'}") if ($result == 0);
460     $$TurnoutData{$TurnoutNmbr}{'Pid'} = 0;
461 }
462
463 $pwmCurrent = $$TurnoutData{$TurnoutNmbr}{'Pos'};
464 if ($pwmCurrent == $pwmFinal) {          # Done if already in position.
465     &DisplayDebug(2, "MoveTurnout, $TurnoutNmbr already in " .
466         "requested position: $pwmFinal");
467     return 2;
468 }
469
470 $moveRate = $$TurnoutData{$TurnoutNmbr}{'Rate'};
471
472 if ($moveRate > 0) {
473     # Fork program to complete the move. Use Forks::Super which is a go
474     # between the parent and child. It has a function for writing child
475     # data back to the main program using child STDOUT and STDERR. It is
476     # not necessary to 'reap' the child when using Forks::Super. Also,
477     # SIG{CHILD} should not be set by this program. It is set/used by
478     # Forks::Super. Do no other printing, including debug output.
479     #
480     # STDERR: move complete. $TurnoutData{<tNmbr>}{'Pid'} set to 0.

```



```

481     # STDOUT: new turnout position. $TurnoutData{<tNmbr>}{'Pos'}.
482
483     &DisplayDebug(2, "MoveTurnout, pre-fork: $Function " .
484         "$TurnoutNmbr    pwmCurrent: $pwmCurrent" .
485         "    pwmFinal: $pwmFinal    moveRate: $moveRate");
486
487     $pid = fork { os_priority => 1,
488         stdout => \$$TurnoutData{$TurnoutNmbr}{'Pos'},
489         stderr => \$$TurnoutData{$TurnoutNmbr}{'Pid'} };
490     if (!defined($pid)) {
491         &DisplayError("TurnoutChildProcess, Failed to create " .
492             "child process. $!");
493         return 1;
494     }
495     #-----
496     elsif ($pid == 0) {          # fork returned 0, so this is the child
497         $moveStep = $moveRate/50;          # Step increment
498         while ($pwmCurrent != $pwmFinal) {
499             if ($pwmCurrent < $pwmFinal) {      # Determine move direction
500                 $pwmCurrent += $moveStep;
501                 $pwmCurrent = $pwmFinal if ($pwmCurrent > $pwmFinal);
502             }
503             else {
504                 $pwmCurrent -= $moveStep;
505                 $pwmCurrent = $pwmFinal if ($pwmCurrent < $pwmFinal);
506             }
507
508             if (&SetTurnoutPosition($pwmCurrent, $TurnoutNmbr, $TurnoutData)) {
509                 # Retain previous pwmCurrent in Pos if error is returned.
510                 print STDERR 0;          # Clear Pid, move has completed.
511                 exit(1);                # Starting position is retained.
512             }
513             sleep 0.02;
514         }
515         print STDOUT $pwmCurrent;        # Store position of turnout
516         print STDERR 0;                  # Clear Pid, move has completed.
517         exit(0);
518     }
519     #-----
520     $$TurnoutData{$TurnoutNmbr}{'Pid'} = $pid; # Parent: Move in-progress.
521     &DisplayDebug(1, "MoveTurnout, $Function $TurnoutNmbr " .
522         "forked pid: $$TurnoutData{$TurnoutNmbr}{'Pid'}");
523 }
524 else {
525     &DisplayWarning("MoveTurnout, Rate value must be greater than 0.");
526     return 1;
527 }
528 }
529 else {
530     &DisplayError("MoveTurnout, invalid turnout number: $TurnoutNmbr");
531     return 1;
532 }
533 return 0;
534 }
535
536 # =====
537 # FUNCTION: SetTurnoutPosition
538 #
539 # DESCRIPTION:
540 #     This routine sets the turnout servo using the specified data. This

```

```

541 # routine writes the I2C interface with the needed command bytes.
542 #
543 # This routine checks the Position value to provide some servo protection
544 # due to a possible program runtime error.
545 #
546 # CALLING SYNTAX:
547 # $result = &SetTurnoutPosition($Position, $TurnoutNmbr, \%TurnoutData);
548 #
549 # ARGUMENTS:
550 # $Position      PWM position to set.
551 # $TurnoutNmbr   Turnout number.
552 # $TurnoutData   Pointer to TurnoutData hash.
553 #
554 # RETURNED VALUES:
555 # 0 = Success, 1 = Error.
556 #
557 # ACCESSED GLOBAL VARIABLES:
558 # None.
559 # =====
560 sub SetTurnoutPosition {
561     my($Position, $TurnoutNmbr, $TurnoutData) = @_;
562     my($driver, $reg_start, $reg_data_on, $reg_data_off);
563     my(@data) = ();
564
565     # The MoveTurnout subroutine uses STDOUT and STDERR to report final turnout
566     # position to the parent process. Debug messaging must be commented out if
567     # not doing code debug. Otherwise, TurnoutDataFile.txt will be corrupted
568     # when Ctrl+C is used.
569
570     # &DisplayDebug(2, "SetTurnoutPosition, $TurnoutNmbr - $Position");
571
572     if (exists($$TurnoutData{$TurnoutNmbr})) {
573         $Position = int($Position);
574         if ($Position < $$TurnoutData{$TurnoutNmbr}{'MinPos'}) {
575             $Position = $$TurnoutData{$TurnoutNmbr}{'MinPos'};
576             # &DisplayWarning("SetTurnoutPosition, Turnout $TurnoutNmbr " .
577             #                 "PWM value beyond MinPos limit. Set to " .
578             #                 "MinPos $Position");
579         }
580         if ($Position > $$TurnoutData{$TurnoutNmbr}{'MaxPos'}) {
581             $Position = $$TurnoutData{$TurnoutNmbr}{'MaxPos'};
582             # &DisplayWarning("SetTurnoutPosition, Turnout $TurnoutNmbr " .
583             #                 "PWM value beyond MaxPos limit. Set to " .
584             #                 "MaxPos $Position");
585         }
586     }
587
588     $reg_start = (($$TurnoutData{$TurnoutNmbr}{'Port'} % 16) * 4) + 6;
589
590     # Stagger pulse start (* 10) to minimize power drops.
591     $reg_data_on = $$TurnoutData{$TurnoutNmbr}{'Port'} * 10;
592     push (@data, ($reg_data_on & 0xFF));      # on_L
593     push (@data, (($reg_data_off >> 8) & 0x0F)); # on_H
594     $reg_data_off = $reg_data_on + $Position;
595     push (@data, ($reg_data_off & 0xFF));      # off_L
596     push (@data, (($reg_data_off >> 8) & 0x0F)); # off_H
597
598     $driver = RPi::I2C->new($$TurnoutData{$TurnoutNmbr}{'Addr'});
599     unless ($driver->check_device($$TurnoutData{$TurnoutNmbr}{'Addr'})) {
600         &DisplayError("SetTurnoutPosition, Failed to initialize " .
601                     "I2C address: " .

```

```

601         sprintf("%.2x", $$TurnoutData{$TurnoutNmbr}{'Addr'}));
602     return 1;
603 }
604 $driver->write_block(\@data, $reg_start);
605 undef($driver);
606 }
607 else {
608     &DisplayError("SetTurnoutPosition, invalid turnout number: $TurnoutNmbr");
609     return 1;
610 }
611 return 0;
612 }
613
614 # =====
615 # FUNCTION:  GetTemperature
616 #
617 # DESCRIPTION:
618 #   This routine gets the current temperature value in degrees Celsius from
619 #   the DS18B20 sensor attached to GPIO4. A timeout variable is also set to
620 #   facilitate future calls to this code.
621 #
622 #   The DS18B20 sensor is a 1-wire protocol device that is interfaced using
623 #   raspbian modprobe. The device must be configured external to this program.
624 #   Add the following.
625 #
626 #   sudo nano /boot/config.txt
627 #       dtoverlay=w1-gpio
628 #
629 #   sudo nano /etc/modules
630 #       w1-gpio
631 #       w1-therm
632 #
633 #   Reboot RPi.
634 #
635 #   Then use 'ls /sys/bus/w1/devices' to list the unique device ID and replace
636 #   <sensorId> in the $sensor variable below.
637 #
638 #   If a DS18B20 sensor is not present or misconfigured, safe values are set
639 #   in the TurnoutData hash.
640 #
641 #   Ambient temperature accuracy is affected by the sensor's proximity to the
642 #   warm circuit board electronics. The $calibration variable adjusts the
643 #   returned temperature value based on comparison with thermometer measurement.
644 #
645 #   Use a digital thermometer to measure the layout benchwork temperature and
646 #   compare it to the temperature value displayed on the console during DnB.pl
647 #   startup. Enter an appropriate adjustment value into $calibration.
648 #
649 # CALLING SYNTAX:
650 #   $result = &GetTemperature(\%TurnoutData);
651 #
652 # ARGUMENTS:
653 #   $TurnoutData    Pointer to TurnoutData hash.
654 #
655 # RETURNED VALUES:
656 #   0 = Error, non-zero = temperature.
657 #
658 # ACCESSED GLOBAL VARIABLES:
659 #   None.
660 # =====

```

```

661 sub GetTemperature {
662     my($TurnoutData) = @_ ;
663     my($temp);
664     # /sys/bus/w1/devices/<sensorId>/w1_slave
665     my($sensor) = '/sys/bus/w1/devices/28-030197944687/w1_slave';
666     my($calibration) = 1.837; # Centigrade value!
667     my($temperature) = 0;
668
669     if (-e $sensor) {
670         my $result = `cat $sensor`;
671         if ($result =~ m/t=(\d+)/) {
672             $temp = $1 / 1000;
673             if ($temp > 0 and $temp < 38) {
674                 $temperature = $temp - $calibration;
675             }
676             else {
677                 &DisplayError(1, "GetTemperature, Invalid temperature: $temperature");
678             }
679         }
680         else {
681             &DisplayDebug(1, "GetTemperature, Temperature value not parsed.");
682         }
683     }
684     else {
685         &DisplayDebug(1, "GetTemperature, DS18B20 sensor is not configured.");
686     }
687     $$TurnoutData{'00'}{'Temperature'} = $temperature;
688     $$TurnoutData{'00'}{'Timeout'} = time + 300;
689     return $temperature;
690 }
691
692 # =====
693 # FUNCTION: TestServoAdjust
694 #
695 # DESCRIPTION:
696 # This routine cycles the specified turnout range between the open and
697 # closed positions.
698 #
699 # CALLING SYNTAX:
700 # $result = &TestServoAdjust($Param, \%TurnoutData);
701 #
702 # ARGUMENTS:
703 # $Param Servo number and temperatures. -w Tx[p]:t1,t2,...
704 # $TurnoutData Pointer to TurnoutData hash.
705 #
706 # RETURNED VALUES:
707 # 0 = Success, 1 = Error.
708 #
709 # ACCESSED GLOBAL VARIABLES:
710 # $main::MainRun
711 # =====
712 sub TestServoAdjust {
713
714     my($Param, $TurnoutData) = @_ ;
715     my($servo, $position, $temp, $pos, $origPos, $sndFlag, $result);
716     my(@positions, @temperatures);
717
718     &DisplayDebug(1, "TestServoAdjust, Entry ... Param: '$Param'");
719     if ($Param =~ m/^(\\d+)(\\D*):(\\.+)/) {
720         $servo = $1;

```

```

721 $position = lc($2);
722 @temperatures = split(',', $3);
723
724 # Validate input parameters.
725 $servo = "0${servo}" if (length($servo) == 1);
726 unless (exists($$TurnoutData{$servo})) {
727     &DisplayError("TestServoAdjust, invalid servo number: $servo");
728     return 1;
729 }
730 if ($position eq '') {
731     @positions = ('Open', 'Middle', 'Close');
732 }
733 elseif ($position =~ m/o/) {
734     @positions = ('Open');
735 }
736 elseif ($position =~ m/m/) {
737     @positions = ('Middle');
738 }
739 elseif ($position =~ m/c/) {
740     @positions = ('Close');
741 }
742 else {
743     &DisplayError("TestServoAdjust, invalid position: $position");
744     return 1;
745 }
746 foreach my $temp (@temperatures) {
747     $temp = &Trim($temp);
748     unless ($temp > 0 and $temp < 38) {
749         &DisplayError("TestServoAdjust, invalid temperature: $temp");
750         return 1;
751     }
752 }
753
754 # Save current servo position for later restoration.
755 foreach my $pos ('Open', 'Middle', 'Close') {
756     if ($$TurnoutData{$servo}{$pos} eq $$TurnoutData{$servo}{'Pos'}) {
757         $origPos = $pos;
758         last;
759     }
760 }
761
762 # Start testing.
763 while ($main::MainRun) {
764     foreach my $pos (@positions) {
765         $sndFlag = 1;
766         foreach my $temp (@temperatures) {
767             $$TurnoutData{'00'}{'Temperature'} = $temp;
768             $result = &MoveTurnout($pos, $servo, $TurnoutData);
769             &DisplayDebug(1, "TestServoAdjust, pos: $pos servo: '$servo' (" .
770                 $$TurnoutData{$servo}{'Id'} . ") " .
771                 "temp: $temp result: $result");
772             # Sound tone.
773             if ($sndFlag eq 1) {
774                 &PlaySound("C.wav");
775                 $sndFlag = 0;
776             }
777             else {
778                 &PlaySound("E.wav");
779             }
780             # Wait for move to complete.

```

```

781         while ($$TurnoutData{$servo}{'Pid'}) {
782             sleep 0.25;
783         }
784         last if ($main::MainRun == 0);
785         sleep 2; # Intra-temperature delay
786     }
787     last if ($main::MainRun == 0);
788 }
789 }
790
791 # Restore original servo position.
792 $$TurnoutData{'00'}{'Temperature'} = 0;
793 $result = &MoveTurnout($origPos, $servo, $TurnoutData);
794 while ($$TurnoutData{$servo}{'Pid'}) {
795     sleep 0.25;
796 }
797 }
798 else {
799     &DisplayError("TestServoAdjust, invalid parameters: '$Param'");
800     return 1;
801 }
802 return 0;
803 }
804
805 # =====
806 # FUNCTION: TestTurnouts
807 #
808 # DESCRIPTION:
809 #   This routine cycles the specified turnout range between the open and
810 #   closed positions.
811 #
812 # CALLING SYNTAX:
813 #   $result = &TestTurnouts($Range, \%TurnoutData);
814 #
815 # ARGUMENTS:
816 #   $Range          Turnout number or range to use.
817 #   $TurnoutData    Pointer to TurnoutData hash.
818 #
819 # RETURNED VALUES:
820 #   0 = Success, 1 = Error.
821 #
822 # ACCESSED GLOBAL VARIABLES:
823 #   $main::MainRun
824 # =====
825 sub TestTurnouts {
826
827     my($Range, $TurnoutData) = @_;
828     my($moveResult, $turnout, $start, $end, $nbr, $oper, $pid, $cnt,
829        @turnoutNumbers, @inProgress, $position);
830     my($cntTurnout) = scalar keys %$TurnoutData;
831     my(%operation) = (1 => 'Open ', 2 => 'Close');
832     my(@turnoutList) = ();
833     my($random, $wait) = (0, 0);
834
835     &DisplayDebug(1, "TestTurnouts, Entry ... Range: '$Range'    " .
836                  "cntTurnout: $cntTurnout");
837
838     # =====
839     # Set specified position and exit.
840

```

```

841 if ($Range =~ m/^(Open):(\d+)/i or $Range =~ m/^(Close):(\d+)/i or
842     $Range =~ m/^(Middle):(\d+)/i) {
843     $position = ucfirst(lc $1);
844     $turnout = $2;
845     $turnout = "0${turnout}" if (length($turnout) == 1);
846
847     # The %TurnoutData Id string must contain the word turnout.
848     if ($$TurnoutData{$turnout}{'Id'} =~ m/turnout/) {
849         &MoveTurnout($position, $turnout, $TurnoutData);
850         &DisplayMessage("Turnout $turnout set to '$position'.");
851     }
852     else {
853         &DisplayError("TestTurnouts, invalid turnout number: $turnout");
854     }
855     exit(0);
856 }
857 elseif ($Range =~ m/^(Open)$/i or $Range =~ m/^(Close)$/i or
858     $Range =~ m/^(Middle)$/i) {
859     $position = ucfirst(lc $1);
860
861     # The %TurnoutData Id string must contain the word turnout.
862     foreach my $turnout (sort keys %$TurnoutData) {
863         if ($$TurnoutData{$turnout}{'Id'} =~ m/turnout/) {
864             &MoveTurnout($position, $turnout, $TurnoutData);
865             &DisplayDebug(1, "TestTurnouts, turnout: $turnout set " .
866                 "to $position");
867         }
868     }
869     &DisplayMessage("All turnouts set to '$position'.");
870     exit(0);
871 }
872
873 # =====
874 # Process special modifiers and then setup for looped testing.
875
876 if ($Range =~ m/r/i) {
877     $random = 1;
878     $Range =~ s/r//i;
879 }
880 if ($Range =~ m/w/i) {
881     $wait = 1;
882     $Range =~ s/w//i;
883 }
884
885 if ($Range =~ m/(\d+):(\d+)/) { # Range specified.
886     $start = $1;
887     $end = $2;
888     if ($start > $end or $start <= 0 or $start > $cntTurnout or $end <= 0 or
889         $end > $cntTurnout) {
890         &DisplayError("TestTurnouts, invalid turnout range: '$Range' .
891             " cntTurnout: $cntTurnout");
892         return 1;
893     }
894     for ($turnout = $start; $turnout <= $end; $turnout++) {
895         push (@turnoutList, $turnout);
896     }
897 }
898 else {
899     @turnoutList = split(",", $Range);
900 }

```

```

901 &DisplayDebug(1, "TestTurnouts, random: $random wait: $wait " .
902             "turnoutList: '@turnoutList'");
903
904 # Identify the servos being used for turnouts. The %TurnoutData Id string
905 # must contain the word turnout.
906 foreach my $key (sort keys %$TurnoutData) {
907     if ($$TurnoutData{$key}{'Id'} =~ m/turnout/) {
908         push (@turnoutNumbers, $key);
909     }
910 }
911
912 $oper = 'Open ';
913 while ($main::MainRun) {
914     # For random testing, we randomize the turnoutNumbers list and also the
915     # Open/Close operation. For non-random, Open and then Close the turnouts
916     # in the specified order.
917     &ShuffleArray(\@turnoutNumbers) if ($random == 1);
918
919     foreach my $turnout (@turnoutNumbers) {
920         return 0 unless ($main::MainRun);
921         $nbr = $turnout;
922         $nbr =~ s/^0//;
923         if (grep /^$nbr$/, @turnoutList) { # Move turnout if on the list.
924             $oper = $operation{(int(rand(2))+1)} if ($random == 1);
925             if ($#inProgress < 0) {
926                 &DisplayMessage("TestTurnouts, $oper $turnout Concurrent " .
927                               "moves: none");
928             }
929             else {
930                 &DisplayMessage("TestTurnouts, $oper $turnout Concurrent " .
931                               "moves: @inProgress");
932             }
933             $moveResult = &MoveTurnout($oper, $turnout, $TurnoutData);
934             return 1 if ($moveResult == 1);
935             if ($moveResult == 2) {
936                 &DisplayDebug(2, "TestTurnouts, MoveTurnout $turnout returned " .
937                               "already in position.");
938             }
939             elsif ($moveResult == 0) {
940                 if ($wait == 1) {
941                     $cnt = 20;
942                     while ($$TurnoutData{$turnout}{'Pid'}) {
943                         if ($cnt == 0) {
944                             &DisplayError("TestTurnouts, timeout waiting for " .
945                                           "turnout $turnout to complete positioning.");
946                             return 1;
947                         }
948                         &DisplayDebug(2, "TestTurnouts, waiting for " .
949                                       "pid: $$TurnoutData{$turnout}{'Pid'}");
950                         sleep 0.5;
951                         $cnt--;
952                     }
953                     &DisplayDebug(2, "TestTurnouts, Turnout $turnout new position: " .
954                                   "$$TurnoutData{$turnout}{'Pos'}");
955                 }
956             }
957             @inProgress = ();
958             foreach my $key (sort keys(%$TurnoutData)) {
959                 push (@inProgress, $key) if ($$TurnoutData{$key}{'Pid'} != 0);
960             }

```



```
961         sleep 0.05 unless ($moveResult == 2);
962     }
963 }
964
965 if ($random == 0) {    # Change if doing sequential testing.
966     if ($oper =~ m/Open/) {
967         $oper = 'Close ';
968     }
969     else {
970         $oper = 'Open  ';
971     }
972 }
973 sleep 2;
974 }
975 return 0;
976 }
977
978 return 1;
979
```