

```

1  # =====
2  # FILE: DnB_Turnout.pm                                     8/12/2020
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 for ($board = 1; $board <= scalar keys(%$ServoBoardAddress); $board++) {
303     if ($$ServoBoardAddress{$board} == 0) {
304         &DisplayDebug(1, "InitTurnouts, Skip board $board " .
305             "I2C_Address 0, code debug.");
306     }
307     next;
308 }
309 &DisplayMessage("Initializing turnout I2C board $board ...");
310 return 1 if (&I2C_InitServoDriver($board, $$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, $TurnoutNmbr, \%TurnoutData);
363 #
364 # ARGUMENTS:
365 #   $Function      'Open', 'Middle', or 'Close'.
366 #   $TurnoutNmbr   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, $TurnoutNmbr, $TurnoutData) = @_;
377     my($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 $TurnoutNmbr");
382
383     if ($TurnoutNmbr ne "") {
384         if ($Function =~ m/Open/i) {
385             $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'Open'};
386         }
387         elsif ($Function =~ m/Middle/i) {
388             $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'Middle'};
389         }
390         elsif ($Function =~ m/Close/i) {
391             $pwmFinal = $$TurnoutData{$TurnoutNmbr}{'Close'};
392         }
393         else {
394             &DisplayError("MoveTurnout, invalid function: '$Function'");
395             return 1;
396         }
397
398         # If gate or semaphore servo, adjust $pwmFinal for temperature.
399         if ($$TurnoutData{$TurnoutNmbr}{'Id'} =~ m/semaphore/i or
400             $$TurnoutData{$TurnoutNmbr}{'Id'} =~ m/gate/i) {
401             if ($$TurnoutData{'00'}{'Temperature'} > 0 and
402                 $$TurnoutData{'00'}{'Temperature'} < 38) {
403                 $noAdj = $pwmFinal;    # Used only for debug message.
404                 #  5  7  9 11 13 15 17 19 21 23 25 27 29 31 33 35 37  degree C
405                 # -8 -7 -6 -5 -4 -3 -2 -1  0 +1 +2 +3 +4 +5 +6 +7 +8  step adjust
406                 # Change divisor (-2) to increase/decrease overall step count.
407                 # Change constant (21) to shift center point temperature.
408                 # Note: TurnoutData MinPos and MaxPos will limit adjustment if
409                 #       set too close to Open/Close value.
410                 $adjust = int((21 - $$TurnoutData{'00'}{'Temperature'}) / -2);
411                 &DisplayDebug(1, "MoveTurnout, servo: $TurnoutNmbr    " .
412                     "adjust: $adjust");
413
414                 # Application of adjustment is dependent on close direction.
415                 if ($$TurnoutData{$TurnoutNmbr}{'Open'} >
416                     $$TurnoutData{$TurnoutNmbr}{'Close'}) {
417                     $pwmFinal += $adjust;
418                 }
419                 else {
420                     $pwmFinal -= $adjust;

```

```

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

```



```

481     $pid = fork { os_priority => 1,
482                 stdout => \$$TurnoutData{$TurnoutNmbr}{'Pos'},
483                 stderr => \$$TurnoutData{$TurnoutNmbr}{'Pid'} };
484     if (!defined($pid)) {
485         &DisplayError("TurnoutChildProcess, Failed to create " .
486                     "child process. $!");
487         return 1;
488     }
489     #-----
490     elsif ($pid == 0) {          # fork returned 0, so this is the child
491         $moveStep = $moveRate/50;      # Step increment
492         while ($pwmCurrent != $pwmFinal) {
493             if ($pwmCurrent < $pwmFinal) {      # Determine move direction
494                 $pwmCurrent += $moveStep;
495                 $pwmCurrent = $pwmFinal if ($pwmCurrent > $pwmFinal);
496             }
497             else {
498                 $pwmCurrent -= $moveStep;
499                 $pwmCurrent = $pwmFinal if ($pwmCurrent < $pwmFinal);
500             }
501
502             if (&SetTurnoutPosition($pwmCurrent, $TurnoutNmbr, $TurnoutData)) {
503                 # Retain previous pwmCurrent in Pos if error is returned.
504                 print STDERR 0;          # Clear Pid, move has completed.
505                 exit(1);                # Starting position is retained.
506             }
507             sleep 0.02;
508         }
509         print STDOUT $pwmCurrent;        # Store position of turnout
510         print STDERR 0;                  # Clear Pid, move has completed.
511         exit(0);
512     }
513     #-----
514     $$TurnoutData{$TurnoutNmbr}{'Pid'} = $pid; # Parent: Move in-progress.
515     &DisplayDebug(1, "MoveTurnout, $Function $TurnoutNmbr " .
516                 "forked pid: $$TurnoutData{$TurnoutNmbr}{'Pid'}");
517 }
518 else {
519     &DisplayWarning("MoveTurnout, Rate value must be greater than 0.");
520     return 1;
521 }
522 }
523 else {
524     &DisplayError("MoveTurnout, invalid turnout number: $TurnoutNmbr");
525     return 1;
526 }
527 return 0;
528 }
529
530 # =====
531 # FUNCTION:  SetTurnoutPosition
532 #
533 # DESCRIPTION:
534 #   This routine sets the turnout servo using the specified data. This
535 #   routine writes the I2C interface with the needed command bytes.
536 #
537 #   This routine checks the Position value to provide some servo protection
538 #   due to a possible program runtime error.
539 #
540 # CALLING SYNTAX:

```

```

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

```

```

601     else {
602         &DisplayError("SetTurnoutPosition, invalid turnout number: $TurnoutNmbr");
603         return 1;
604     }
605     return 0;
606 }
607
608 # =====
609 # FUNCTION:  GetTemperature
610 #
611 # DESCRIPTION:
612 #     This routine gets the current temperature value in degrees Celsius from
613 #     the DS18B20 sensor attached to GPIO4. A timeout variable is also set to
614 #     facilitate future calls to this code.
615 #
616 #     The DS18B20 sensor is a 1-wire protocol device that is interfaced using
617 #     raspbian modprobe. The device must be configured external to this program.
618 #     Add the following.
619 #
620 #     sudo nano /boot/config.txt
621 #         dtoverlay=w1-gpio
622 #
623 #     sudo nano /etc/modules
624 #         w1-gpio
625 #         w1-therm
626 #
627 #     Reboot RPi.
628 #
629 #     Then use 'ls /sys/bus/w1/devices' to list the unique device ID and replace
630 #     <sensorId> in the $sensor variable below.
631 #
632 #     If a DS18B20 sensor is not present or misconfigured, safe values are set
633 #     in the TurnoutData hash.
634 #
635 #     Ambient temperature accuracy is affected by the sensor's proximity to the
636 #     warm circuit board electronics. The $calibration variable adjusts the
637 #     returned temperature value based on comparison with thermometer measurement.
638 #
639 #     Use a digital thermometer to measure the layout benchwork temperature and
640 #     compare it to the temperature value displayed on the console during DnB.pl
641 #     startup. Enter an appropriate adjustment value into $calibration.
642 #
643 # CALLING SYNTAX:
644 #     $result = &GetTemperature(\%TurnoutData);
645 #
646 # ARGUMENTS:
647 #     $TurnoutData    Pointer to TurnoutData hash.
648 #
649 # RETURNED VALUES:
650 #     0 = Error, non-zero = temperature.
651 #
652 # ACCESSED GLOBAL VARIABLES:
653 #     None.
654 # =====
655 sub GetTemperature {
656     my($TurnoutData) = @_ ;
657     my($temp);
658     #           /sys/bus/w1/devices/<sensorId>/w1_slave
659     my($sensor) = '/sys/bus/w1/devices/28-030197944687/w1_slave';
660     my($calibration) = 1.837;    # Centigrade value!

```

```

661 my($temperature) = 0;
662
663 if (-e $sensor) {
664     my $result = `cat $sensor`;
665     if ($result =~ m/t=(\d+)/) {
666         $temp = $1 / 1000;
667         if ($temp > 0 and $temp < 38) {
668             $temperature = $temp - $calibration;
669         }
670         else {
671             &DisplayError(1, "GetTemperature, Invalid temperature: $temperature");
672         }
673     }
674     else {
675         &DisplayDebug(1, "GetTemperature, Temperature value not parsed.");
676     }
677 }
678 else {
679     &DisplayDebug(1, "GetTemperature, DS18B20 sensor is not configured.");
680 }
681 $$TurnoutData{'00'}{'Temperature'} = $temperature;
682 $$TurnoutData{'00'}{'Timeout'} = time + 300;
683 return $temperature;
684 }
685
686 # =====
687 # FUNCTION: TestServoAdjust
688 #
689 # DESCRIPTION:
690 #   This routine cycles the specified turnout range between the open and
691 #   closed positions.
692 #
693 # CALLING SYNTAX:
694 #   $result = &TestServoAdjust($Param, \%TurnoutData);
695 #
696 # ARGUMENTS:
697 #   $Param          Servo number and temperatures. -w Tx[p]:t1,t2,...
698 #   $TurnoutData    Pointer to TurnoutData hash.
699 #
700 # RETURNED VALUES:
701 #   0 = Success, 1 = Error.
702 #
703 # ACCESSED GLOBAL VARIABLES:
704 #   $main::MainRun
705 # =====
706 sub TestServoAdjust {
707
708     my($Param, $TurnoutData) = @_;
709     my($servo, $position, $temp, $pos, $origPos, $sndFlag, $result);
710     my(@positions, @temperatures);
711
712     &DisplayDebug(1, "TestServoAdjust, Entry ... Param: '$Param'");
713     if ($Param =~ m/^(\\d+)(\\D*):(\\.+)/) {
714         $servo = $1;
715         $position = lc($2);
716         @temperatures = split(',', $3);
717
718         # Validate input parameters.
719         $servo = "0${servo}" if (length($servo) == 1);
720         unless (exists($$TurnoutData{$servo})) {

```

```

721     &DisplayError("TestServoAdjust, invalid servo number: $servo");
722     return 1;
723 }
724 if ($position eq '') {
725     @positions = ('Open', 'Middle', 'Close');
726 }
727 elseif ($position =~ m/o/) {
728     @positions = ('Open');
729 }
730 elseif ($position =~ m/m/) {
731     @positions = ('Middle');
732 }
733 elseif ($position =~ m/c/) {
734     @positions = ('Close');
735 }
736 else {
737     &DisplayError("TestServoAdjust, invalid position: $position");
738     return 1;
739 }
740 foreach my $temp (@temperatures) {
741     $temp = &Trim($temp);
742     unless ($temp > 0 and $temp < 38) {
743         &DisplayError("TestServoAdjust, invalid temperature: $temp");
744         return 1;
745     }
746 }
747
748 # Save current servo position for later restoration.
749 foreach my $pos ('Open', 'Middle', 'Close') {
750     if ($$TurnoutData{$servo}{$pos} eq $$TurnoutData{$servo}{'Pos'}) {
751         $origPos = $pos;
752         last;
753     }
754 }
755
756 # Start testing.
757 while ($main::MainRun) {
758     foreach my $pos (@positions) {
759         $sndFlag = 1;
760         foreach my $temp (@temperatures) {
761             $$TurnoutData{'00'}{'Temperature'} = $temp;
762             $result = &MoveTurnout($pos, $servo, $TurnoutData);
763             &DisplayDebug(1, "TestServoAdjust, pos: $pos servo: '$servo' (" .
764                 $$TurnoutData{$servo}{'Id'} . ") " .
765                 "temp: $temp result: $result");
766
767             # Sound tone.
768             if ($sndFlag eq 1) {
769                 &PlaySound("C.wav");
770                 $sndFlag = 0;
771             }
772             else {
773                 &PlaySound("E.wav");
774             }
775             # Wait for move to complete.
776             while ($$TurnoutData{$servo}{'Pid'}) {
777                 sleep 0.25;
778             }
779             last if ($main::MainRun == 0);
780             sleep 2; # Intra-temperature delay
781         }
782     }
783 }

```

```

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

```

```

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

```

```

901     if ($$TurnoutData{$key}{'Id'} =~ m/turnout/) {
902         push (@turnoutNumbers, $key);
903     }
904 }
905
906 $oper = 'Open ';
907 while ($main::MainRun) {
908     # For random testing, we randomize the turnoutNumbers list and also the
909     # Open/Close operation. For non-random, Open and then Close the turnouts
910     # in the specified order.
911     &ShuffleArray(\@turnoutNumbers) if ($random == 1);
912
913     foreach my $turnout (@turnoutNumbers) {
914         return 0 unless ($main::MainRun);
915         $nbr = $turnout;
916         $nbr =~ s/^0//;
917         if (grep /^$nbr$/, @turnoutList) { # Move turnout if on the list.
918             $oper = $operation{($int(rand(2))+1)} if ($random == 1);
919             if ($#inProgress < 0) {
920                 &DisplayMessage("TestTurnouts, $oper $turnout Concurrent " .
921                     "moves: none");
922             }
923             else {
924                 &DisplayMessage("TestTurnouts, $oper $turnout Concurrent " .
925                     "moves: @inProgress");
926             }
927             $moveResult = &MoveTurnout($oper, $turnout, $TurnoutData);
928             return 1 if ($moveResult == 1);
929             if ($moveResult == 2) {
930                 &DisplayDebug(2, "TestTurnouts, MoveTurnout $turnout returned " .
931                     "already in position.");
932             }
933             elsif ($moveResult == 0) {
934                 if ($wait == 1) {
935                     $cnt = 20;
936                     while ($$TurnoutData{$turnout}{'Pid'}) {
937                         if ($cnt == 0) {
938                             &DisplayError("TestTurnouts, timeout waiting for " .
939                                 "turnout $turnout to complete positioning.");
940                             return 1;
941                         }
942                         &DisplayDebug(2, "TestTurnouts, waiting for " .
943                             "pid: $$TurnoutData{$turnout}{'Pid'}");
944                         sleep 0.5;
945                         $cnt--;
946                     }
947                     &DisplayDebug(2, "TestTurnouts, Turnout $turnout new position: " .
948                         "$$TurnoutData{$turnout}{'Pos'}");
949                 }
950             }
951             @inProgress = ();
952             foreach my $key (sort keys(%$TurnoutData)) {
953                 push (@inProgress, $key) if ($$TurnoutData{$key}{'Pid'} != 0);
954             }
955             sleep 0.05 unless ($moveResult == 2);
956         }
957     }
958
959     if ($random == 0) { # Change if doing sequential testing.
960         if ($oper =~ m/Open/) {

```



```
961         $oper = 'Close ';
962     }
963     else {
964         $oper = 'Open  ';
965     }
966 }
967 sleep 2;
968 }
969 return 0;
970 }
971
972 return 1;
973
```