```
// Nano is powered (+5V) from rectified input power (9-15V AC or DC) via 7805 voltage regulator
// DCC signal taken via 6N137 optoisolator to Nano pin D2
// Each of five LEDs (1xRed + 4xGrn) are connected as follows -
//    Cathode (short leg, flat side) to GND
//    Anode (long leg, round side) to 220ohm resistor
// Other ends of 220 ohm resistors to Nano pins D3 (Red), D4, D5, D6, D7 (Green)
// Nano pins D9, D10, D11, D12 are each connected to one end of a 10K resistor
// Other ends of 10K resistors are connected to +5V
// One side of each of four pushbuttons is connected to Nano pins D9, D10, D11, D12 respectively
// Other side of each pushbutton is connected to GND
// Servo signal connections are attached to Nano pins A0, A1, A2, A3
// Servo power connections are attached to +5V and GND
// Nano pin D8 has internal pull-up enabled - shorted to GND to enable input of servo DCC addresses

// To prevent all debug messages being compiled as part of the sketch (and hence not
// displayed by the Serial Monitor) the following line can, if you wish, be commented out -
#define DEBUG

#include <NmraDcc.h>
#include <Servo.h>

NmraDcc  DCC ;
Servo servo[4];

const byte version = 82;  // Software Version = 5.2 (0x52)
const byte adr_prog = 8;  // Set Output Addresses
const byte sw_slct = 9;   // Select
const byte sw_left = 10;  // Left
const byte sw_rite = 11;  // Right
const byte sw_oper = 12;  // Operate

byte  servcentr = 90;
byte  servdetch = 0;  // Time to detach all servos after initialisation
byte  gotocentr = 0;
byte  gotoreset = 0;
byte  slctpress = 0; // Select switch operation state
byte  leftpress = 0; // Left switch operation state
byte  ritepress = 0; // Right switch operation state
byte  operpress = 0; // Operate switch operation state
byte  movedirn = 0;
```

```
41  byte  sindex = 0;
42  byte  flash = 0;
43  byte  cvs_done = 0;
44  byte  nrm_rev = 0;
45  byte  numsrvpins = 4;
46  byte  srvpins[] = {14,15,16,17}; // Nano pins for Servos
47  byte  numledpins = 5;
48  byte  ledpins[] = {3,4,5,6,7};   // Nano pins for LEDs
49  byte  numswpins = 4;
50  byte  swpins[] = {9,10,11,12};   // Nano pins for Switches
51  byte  prog_actv = 0;
52  byte  prog_done = 0;
53  int   set_adr[] = {0,0,0,0};     // Temporary storage for Output Addresses
54
55  int t;              // temp
56  int i;
57  int slctstep = 4;   // Step in setup of each Servo - initialise to invalid value
58  byte busy_dcc = 0;  // Set = 1 when any DCC command(s) being executed
59  byte actv_indx = 0; // Bit Set = 1 when DCC command for specific servo being executed
60                      // Bit 0 = Servo 0, Bit 1 = Servo 1, Bit 2 = Servo 2, Bit 3 = Servo 3
61  byte actv_mask = 0; // Mask to set a specific bit in actv_indx
62
63  typedef struct
64  {
65    byte active;
66    byte curr_position;
67    byte slow_rate;
68    byte right_limit;
69    byte left_limit;
70    byte loop_count;
71    byte nrm_dirn;
72    byte end_value;
73  } serv_param;             // Servo Parameters structure (block)
74  serv_param servoact[4];   // Array of Parameters for four Servos
75
76  typedef struct
77  {
78    int  cv_adr;
79    byte cv_val;
80  } cv_pair;
```

```
81
82  byte cv_value;
83  int  SET_CV_Address = 24;              // This Address is for setting CV'S like a Loco using Ops Mode
84  int  Accessory_Address = 1;            // This Address is the Board Address - not necessarily a Servo Address
85  byte CV_DECODER_MASTER_RESET = 120;    // This is the CV Address for Full Reset - load a value of 120
86                                         // to this location and then press the Nano Reset button
87                                         // Return to default CV values can also be achieved by loading any value
88                                         // other than 0xAD (173) to CV50 and then pressing the Nano Reset button
89  byte CV_To_Store_SET_CV_Address = 121; // The address used to change CVs using Ops Mode (set as 24 above) is
90                                         // stored in this location, and in the following CV if greater than 256
91  byte CV_Accessory_Address = CV_ACCESSORY_DECODER_ADDRESS_LSB;  // CV01 - the Board Address
92
93  cv_pair FactoryDefaultCVs [] =
94  {
95    // These two CVs define the Long Accessory Address
96    {CV_ACCESSORY_DECODER_ADDRESS_LSB, Accessory_Address&0xFF},
97    {CV_ACCESSORY_DECODER_ADDRESS_MSB, (Accessory_Address>>8)&0x07},
98
99    {CV_MULTIFUNCTION_EXTENDED_ADDRESS_MSB, 0},
100   {CV_MULTIFUNCTION_EXTENDED_ADDRESS_LSB, 0},
101
102 //  {CV_29_CONFIG,CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_F0_LOCATION}, // Accesory Decoder Short Address
103   {CV_29_CONFIG, CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_EXT_ADDRESSING | CV29_F0_LOCATION},  // Accesory Decoder  Long  Address
104
105   {CV_DECODER_MASTER_RESET, 0},
106   {CV_To_Store_SET_CV_Address, SET_CV_Address&0xFF },   // LSB Set CV Address
107   {CV_To_Store_SET_CV_Address+1,(SET_CV_Address>>8)&0x3F },  //MSB Set CV Address
108   {30, 0},    //  Not Used
109   {31, 0},    //  Not Used
110   {32, 0},    //  Not Used
111   {33, 0},    //  Not Used
112   {34, 0},    //  Not Used
113   {35, 0},    //  Not Used
114   {36, 0},    //  Not Used
115   {37, 0},    //  Not Used
116   {38, 0},    //  Not Used
117   {39, 0},    //  Not Used
118   {40, 0},    //  Not Used
119   {41, 1},    //  Servo 1 Address LSB (01)
120   {42, 0},    //  Servo 1 Address MSB
```

```
121     {43, 2},    //  Servo 2 Address LSB (02)
122     {44, 0},    //  Servo 2 Address MSB
123     {45, 3},    //  Servo 3 Address LSB (03)
124     {46, 0},    //  Servo 3 Address MSB
125     {47, 4},    //  Servo 4 Address LSB (04)
126     {48, 0},    //  Servo 4 Address MSB
127     {49, 0},    //  Not Used
128     {50, 0},    //  Reset to default CVs if CV50 not equal to 173 (0xAD = "All Default")
129     {51, 0},    //  Not Used
130     {52, 0},    //  Not Used
131     {53, 0},    //  Not Used
132     {54, 0},    //  Not Used
133     {55, 6},    //  Servo 1 - Slow Rate (1 to 16)
134     {56, 1},    //  Direction - 1 = Normal Operation, 0 = Reverse Operation
135     {57, 70},   //  Right Limit
136     {58, 110},  //  Left Limit
137     {59, 70},   //  Current Position
138     {60, 6},    //  Servo 2 - Slow Rate (1 to 16)
139     {61, 1},    //  Direction - 1 = Normal Operation, 0 = Reverse Operation
140     {62, 70},   //  Right Limit
141     {63, 110},  //  Left Limit
142     {64, 70},   //  Current Position
143     {65, 6},    //  Servo 3 - Slow Rate (1 to 16)
144     {66, 1},    //  Direction - 1 = Normal Operation, 0 = Reverse Operation
145     {67, 70},   //  Right Limit
146     {68, 110},  //  Left Limit
147     {69, 70},   //  Current Position
148     {70, 6},    //  Servo 4 - Slow Rate (1 to 16)
149     {71, 1},    //  Direction - 1 = Normal Operation, 0 = Reverse Operation
150     {72, 70},   //  Right Limit
151     {73, 110},  //  Left Limit
152     {74, 70},   //  Current Position
153     {109, 81},  //  "Q" - Extended Decoder Version
154     {110, 83},  //  "S"
155     {111, 68},  //  "D"
156     {112, 82},  //  Software Version - 0x52
157   };
158   uint8_t FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
159   void notifyCVResetFactoryDefault()
160   {
```

```
161     // Make FactoryDefaultCVIndex non-zero and equal to number of CV's to be reset
162     // to flag to the loop() function that a reset to Factory Defaults needs to be done
163     FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
164   };
165   void(* resetFunc) (void) = 0; // Declare reset function at address 0
166
167   //*************************************************************************************************
168   void setup() {
169     Serial.begin(115200); // Only required if debug messages are to be displayed by serial Monitor
170
171     // Setup which External Interrupt to use for the DCC input, and the associated Pin (2)
172     DCC.pin(0, 2, 0);
173     // Call the main DCC Init function to enable the DCC Receiver
174     DCC.init( MAN_ID_DIY, 601, FLAGS_OUTPUT_ADDRESS_MODE | FLAGS_DCC_ACCESSORY_DECODER, CV_To_Store_SET_CV_Address);
175   //  delay(150);
176
177     pinMode(adr_prog, INPUT_PULLUP);  // Set as Address Programming enable - when connected to GND
178                                       // allows the Output Address of the selected Servo to be set
179                                       // by issuing a DCC Accessory command to that address
180     // Initialize the servo digital pins as outputs
181     for (int i=0; i < numsrvpins; i++) {
182       pinMode(srvpins[i], OUTPUT);
183       digitalWrite(srvpins[i], LOW);
184      }
185     // Initialize the LED digital pins as outputs
186     for (int i=0; i < numledpins; i++) {
187       pinMode(ledpins[i], OUTPUT);
188       digitalWrite(ledpins[i], LOW);
189      }
190     //for (int i=0; i < numledpins; i++) {
191     //  digitalWrite(ledpins[i], HIGH); // Switch on all LEDs in sequence
192     //  delay (30);
193     //}
194     //delay(250);
195     //for (int i=0; i < numledpins; i++) {
196     //  digitalWrite(ledpins[i], LOW);  // .. then switch them off again
197     //  delay (30);
198     //}
199     // Initialize the digital pins connected to the pushbuttons as inputs
200     for (int i=0; i < numswpins; i++) {
```

```
201       pinMode(swpins[i], INPUT);
202     }
203
204     if ((DCC.getCV(CV_DECODER_MASTER_RESET)== CV_DECODER_MASTER_RESET) || (DCC.getCV(50) != 0xAD))
205     {
206       // Reset all defined CVs to default values if value of CV120 = 120 or CV50 is not equal to 173 (0xAD = "All Default")
207       for (int j=0; j < sizeof(FactoryDefaultCVs)/sizeof(cv_pair); j++ )
208       DCC.setCV( FactoryDefaultCVs[j].cv_adr, FactoryDefaultCVs[j].cv_val);
209       digitalWrite(3, 1);
210       delay (1000);
211       digitalWrite(3, 0); // Flash Red LED when CVs loaded
212       DCC.setCV(50, 0xAD);
213       #ifdef DEBUG
214        Serial.print("Reset to Default CVs, CV50 = ");
215        Serial.println(DCC.getCV(50), DEC) ;
216       #endif
217     }
218
219     // Check that current Software Version is stored in CV 112 in EEPROM
220     if (version != byte (DCC.getCV(112))) {
221       DCC.setCV(112, version); // Only save if Software Version has been updated
222       delay(5);
223     }
224
225     // Initialise all attached servos
226     for ( i=0; i < numsrvpins; i++) {
227       cv_value = DCC.getCV(41 + (i*2)) + (256 * DCC.getCV(42 + (i*2)));
228       #ifdef DEBUG
229           Serial.print("CV_Number: ");
230           Serial.print(41+(i*2), DEC) ;
231           Serial.print(" Servo Address: ");
232           Serial.println(cv_value, DEC) ;
233       #endif
234       servoact[i].slow_rate = byte (DCC.getCV(55+(i*5)));
235       if (servoact[i].slow_rate > 32) {
236         servoact[i].slow_rate = 32;       // Check for maximum allowed value
237       }
238       servoact[i].nrm_dirn = byte (DCC.getCV(56+(i*5)));
239       servoact[i].right_limit = byte (DCC.getCV(57+(i*5)));
240       if (servoact[i].right_limit > 180) {
```

```
241         servoact[i].right_limit = 180;
242       }
243       servoact[i].left_limit = byte (DCC.getCV(58+(i*5)));
244       if (servoact[i].left_limit > 180) {
245         servoact[i].left_limit = 180;
246       }
247       servoact[i].curr_position = byte (DCC.getCV(59+(i*5)));
248       if (servoact[i].curr_position > servoact[i].left_limit) {
249         servoact[i].curr_position = servoact[i].left_limit; // Check that Current Postion is within Right & Left Limits
250       }
251       if (servoact[i].curr_position < servoact[i].right_limit) {
252         servoact[i].curr_position = servoact[i].right_limit;
253       }
254       // Attaches servo on pin - enables servo to be driven to defined position
255       servo[i].attach(srvpins[i]);
256       servo[i].write(servoact[i].curr_position);
257   //    delay(150);
258   //    servo[i].detach(); // Removes drive from servo after position reached
259   //    servoact[i].active = 0;
260
261       #ifdef DEBUG
262        Serial.print("Initialised Servo ");
263        Serial.println(i+1, DEC) ;
264       #endif
265     }
266     servdetch = 50; // Set timer to detach servos after 150msec - executed at end of main loop
267     digitalWrite(ledpins[0], HIGH);           // Switch Red LED on
268   }
269
270   //************************************************************************************************
271   void loop()
272   {
273     // The NmraDcc.process() method MUST be called frequently from this loop()
274     // function for correct library operation and to process all DCC packets
275     DCC.process();
276     delay(3); //  Sets normal execution time of loop() if no operations started
277
278   //  ==========  Check for Address Programming Link fitted  ============
279     if (digitalRead(adr_prog) == LOW && prog_done == 0 && slctpress == 0 && operpress == 0) {
280       delay(20);
```

```
    if (digitalRead(adr_prog) == LOW) {
      // Address Programming is active
      prog_actv = 1;
      busy_dcc = 1;
      slctpress = 21; // Prepare to select servo for received Output Address
      for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW); // Ensure all LEDs off
      }
      digitalWrite(ledpins[0], HIGH);           // Switch Red LED on
      digitalWrite(ledpins[slctpress - 20], HIGH);  // Show selected Servo - Green LED (1-4)
      #ifdef DEBUG
        Serial.println("Addr Programming Active");
      #endif
    }
  }
  if (digitalRead(adr_prog) == HIGH && prog_actv == 1) {
    delay(20);
    if (digitalRead(adr_prog) == HIGH) {
      // Programming Link removed - Address Programming will be abandoned
      for(i = 0; i < 4; i++){
        set_adr[i] = 0; // Clear any entered address data
      }
      for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW); // Ensure all LEDs off
      }
      prog_actv = 0;  // Disable Address Programming
      prog_done = 0;
//      slctpress = 0;  // Reset all switch states
//      operpress = 0;
//      leftpress = 0;
//      ritepress = 0;
      busy_dcc = 0;   // Allow normal switch actions
      #ifdef DEBUG
        Serial.println("Addr Programming Inactive");
      #endif
      resetFunc();  // Call system reset - execution stops here then returns to start
    }
  }
  if (prog_done == 1) {
    flash = flash + 1;
```

```
321       if (flash == 100){
322         digitalWrite(3, LOW);    // Switch off Red LED
323       }
324       else{
325         if (flash == 200){
326           digitalWrite(3, HIGH); // Switch on Red LED
327           flash = 0;
328         }
329       }
330     }
331
332   //  ==========  Fetch switch state to select Output Address for programming  ============
333     if (digitalRead(sw_slct) == LOW && prog_actv == 1 && operpress == 0 && slctpress > 20) {
334       delay(20);
335       if (digitalRead(sw_slct) == LOW) {
336         while (digitalRead(sw_slct) == LOW){
337           //Select switch pressed
338         }   // Wait for Select switch to be released
339         slctpress = slctpress + 1;  // Move to next setup state
340         // Setup states -
341         // 21 - Set Output Address 1,  22 - Output Address 2
342         // 23 - Set Output Address 3,  24 - Output Address 4
343         if (slctpress == 25){
344           for(i = 0; i < numledpins; i++){
345             digitalWrite(ledpins[i], LOW);     // Switch all LEDs off
346           }
347           // Transfer enterd Output Addresses (if any) to CV41-CV48
348           for(i = 0; i < 4; i++){
349             if (set_adr[i] != 0) {
350               if ((DCC.getCV(41 + (i*2)) + (256 * DCC.getCV(42 + (i*2)))) != set_adr[i]) {
351                 DCC.setCV(41 + (i*2), byte(set_adr[i] % 256));
352                 delay(5);
353                 DCC.setCV(42 + (i*2), byte(set_adr[i] >> 8));
354                 delay(5);
355                 #ifdef DEBUG
356                   Serial.print("Output Addr ");
357                   Serial.print(i + 1, DEC);
358                   Serial.print(" Programmed = ");
359                   Serial.println(set_adr[i], DEC);
360                 #endif
```

```
361             }
362           }
363         }
364         for(i = 0; i < 4; i++){
365           set_adr[i] = 0; // Clear any entered address data
366         }
367         prog_done = 1;  // Prevent re-entry or setup operations until programming link removed
368         slctpress = 0;  // Programming complete
369         #ifdef DEBUG
370           Serial.println("Addr Programming Complete");
371         #endif
372       }
373       if (slctpress > 20 && slctpress < 25){
374         for(i = 0; i < numledpins; i++){
375           digitalWrite(ledpins[i], LOW);     // Ensure all LEDs off
376         }
377         digitalWrite(ledpins[0], HIGH);      // Switch Red LED on
378         digitalWrite(ledpins[slctpress - 20], HIGH);  // Show Servo(0-3) ready for Address - Green LED (1-4)
379       }
380     }
381   }
382
383 //  ==========  Fetch switch states to control Servo Left/Right Limits setup  ============
384   if (digitalRead(sw_slct) == LOW && busy_dcc == 0 && prog_actv == 0 && operpress == 0 && slctpress < 18) {
385     delay(20);
386     if (digitalRead(sw_slct) == LOW) {
387       digitalWrite(ledpins[0], HIGH);    // Setup started - Red LED on
388       while (digitalRead(sw_slct) == LOW){
389         delay(20);              // Wait for Select switch to be released
390       }
391       slctpress = slctpress + 1;  // Move to next setup state
392       // Setup states -
393       // 1 - Set Servo 1 Left,  2 - Set Servo 1 Right == Use Op to skip to next Servo (from state 1 to 5)
394       // 3 - Set Servo 1 Rate
395       // 4 - Store/Discard Servo 1 Data in CVs 55-59 - Red LED flashes
396       // 5 - Set Servo 2 Left,  6 - Set Servo 2 Right == Use Op to skip to next Servo (from state 5 to 9)
397       // 7 - Set Servo 2 Rate
398       // 8 - Store/Discard Servo 2 Data in CVs 60-64 - Red LED flashes
399       // 9 - Set Servo 3 Left,  10 - Set Servo 3 Right == Use Op to skip to next Servo (from state 9 to 13)
400       // 11 - Set Servo 3 Rate
```

```
401         // 12 - Store/Discard Servo 3 Data in CVs 65-69 - Red LED flashes
402         // 13 - Set Servo 4 Left,  14 - Set Servo 4 Right == Use Op to skip to next Servo (from state 13 to 1)
403         // 15 - Set Servo 4 Rate
404         // 16 - Store/Discard Servo 4 Data in CVs 70-74 - Red LED flashes
405
406         if (slctpress == 17){
407           slctpress = 0;                    // Setup complete
408           slctstep = 4;                     // .. clear selection flags
409           for(i = 0; i < numledpins; i++){
410             digitalWrite(ledpins[i], LOW);  // Switch all LEDs off
411           }
412           for(sindex = 0; sindex < numsrvpins; sindex++){
413             servo[sindex].detach();         // Remove drive from all servos
414           }
415           sindex = 0;
416           movedirn = 0;
417           cvs_done = 1;
418         }
419         else {
420           slctstep = (slctpress - 1) % 4;     // Determine current step in each Servo setup sequence
421           if (slctstep == 0 || slctstep == 1){
422             // Servo Left & Right Limit setup (slctpress = 1,2 or 5,6 or 9,10 or 13,14)
423             for(i = 0; i < numledpins; i++){
424               digitalWrite(ledpins[i], LOW);  // Ensure all LEDs off
425             }
426             for(i = 0; i < numsrvpins; i++){
427               servo[i].detach();         // Remove drive from all servos
428             }
429             sindex = (slctpress - 1);
430             sindex = sindex >> 2;             // Servo address (0 - 3)
431             movedirn = slctpress % 2;         // 1 = Left, 0 = Right
432             digitalWrite(ledpins[0], HIGH);   // Switch Red LED on
433             servo[sindex].attach(srvpins[sindex]);    // Return drive to selected Servo
434             delay(50);
435             digitalWrite(ledpins[1 + sindex], HIGH);  // Setup Servo(0-3) Limits - Green LED (1-4)
436             if (movedirn == 1) {
437               servo[sindex].write(servoact[sindex].left_limit);  // Position servo at current Left Limit
438             }
439             else {
440               servo[sindex].write(servoact[sindex].right_limit); // Position servo at current Right Limit
```

```
441           }
442         }
443         if (slctstep == 2){
444           // Servo Rate setup (slctpress = 3, 7, 11 or 15)
445           for(i = 0; i < numledpins; i++){
446             digitalWrite(ledpins[i], LOW);          // Ensure all LEDs off
447           }
448           sindex = (slctpress - 1);
449           sindex = sindex >> 2;             // Servo address (0 - 3)
450           movedirn = slctpress % 2; // 1 = Left, 0 = Right
451           digitalWrite(ledpins[0], HIGH);           // Switch Red LED on
452           digitalWrite(ledpins[1 + sindex], HIGH);  // Setup Servo(0-3) Rates - Green LED (1-4)
453           setup_servo_move (sindex, 1);             // Selected Servo to move left at set Slow Rate
454           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not completed
455             exec_servo_move ();
456             delay(3);
457           }
458           delay(150);
459           setup_servo_move (sindex, 0);             // Selected Servo to move right at set Slow Rate
460           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not completed
461             exec_servo_move ();
462             delay(3);
463           }
464           delay(150);
465           busy_dcc = 0;   // Allow configuration switch operations again
466         }
467         if (slctstep == 3){
468           // Write Servo setup parameters to CVs (slctpress = 4, 8, 12 or 16)
469           for(i = 0; i < numledpins; i++){
470             digitalWrite(ledpins[i], LOW);          // Ensure all LEDs off
471           }
472           digitalWrite(ledpins[0], HIGH); // Ready to write to CVs - Red LED flashing
473           flash = 0;
474           cvs_done = 0;                   // Prepare to save Servo parameters
475           sindex = (slctpress - 1);
476           sindex = sindex >> 2;           // Servo address (0 - 3)
477           digitalWrite(ledpins[0], HIGH);           // Switch Red LED on
478           digitalWrite(ledpins[1 + sindex], HIGH);  // Setup Servo(0-3) Write CVs - Green LED (1-4)
479         }
480       }
```

```
481         #ifdef DEBUG
482           Serial.print("Select Press = ");
483           Serial.print(slctpress, DEC);
484           Serial.print(" : Servo = ");
485           Serial.print(sindex + 1, DEC);
486           Serial.print(" : Step = ");
487           Serial.println(slctstep + 1, DEC);
488         #endif
489       }
490     }
491   //  ==========  Fetch Operate switch state to skip to next Servo setup sequence  ============
492     if (digitalRead(sw_oper) == LOW && busy_dcc == 0 && prog_actv == 0 && operpress == 0
493                         && (slctpress == 1 || slctpress == 5 || slctpress == 9 || slctpress == 13)) {
494       delay(20);
495       if (digitalRead(sw_oper) == LOW) {
496         sindex = (slctpress - 1);
497         sindex = sindex >> 2;      // Servo address (0 - 3)
498         servo[sindex].write(servoact[sindex].curr_position);  // Position selected servo to its last saved position
499         delay(300);
500         slctpress = slctpress + 4; // Move to set up next Servo
501         if (slctpress == 17) {
502           slctpress = 0;                    // Setup complete
503           slctstep = 4;                     // .. clear selection flags
504           for(i = 0; i < numledpins; i++){
505             digitalWrite(ledpins[i], LOW);  // Switch all LEDs off
506           }
507           for(sindex = 0; sindex < numsrvpins; sindex++){
508             servo[sindex].detach();         // Remove drive from all servos
509           }
510           sindex = 0;
511           movedirn = 0;
512           cvs_done = 1;
513         }
514         else {
515           slctstep = 0;     // Set first step in each Servo setup sequence
516           // Servo Left Limit setup (slctpress = 1, 5, 9, or 13)
517           for(i = 0; i < numledpins; i++){
518             digitalWrite(ledpins[i], LOW);  // Ensure all LEDs off
519           }
520           for(i = 0; i < numsrvpins; i++){
```

```
521             servo[i].detach();        // Remove drive from all servos
522           }
523           sindex = (slctpress - 1);
524           sindex = sindex >> 2;       // Servo address (0 - 3)
525           movedirn = 1;               // 1 = Left
526           digitalWrite(ledpins[0], HIGH);   // Switch Red LED on
527           servo[sindex].attach(srvpins[sindex]);    // Return drive to selected Servo
528           delay(150);
529           digitalWrite(ledpins[1 + sindex], HIGH);  // Setup Servo(0-3) Limits - Green LED (1-4)
530           servo[sindex].write(servoact[sindex].left_limit);  // Position servo at current Left Limit
531           #ifdef DEBUG
532             Serial.print("Select Press = ");
533             Serial.print(slctpress, DEC);
534             Serial.print(" : Servo = ");
535             Serial.print(sindex + 1, DEC);
536             Serial.print(" : Step = ");
537             Serial.println(slctstep + 1, DEC);
538           #endif
539         }
540         while (digitalRead(sw_oper) == LOW){
541           delay(20);              // Wait for Operate switch to be released
542         }
543         delay(300);
544       }
545     }
546
547   //  ==========  Fetch switch states to control Servo operation  ============
548     if (digitalRead(sw_oper) == LOW && busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress < 5) {
549       delay(20);
550       if (digitalRead(sw_oper) == LOW) {
551         for(i = 0; i < 5; i++){
552           digitalWrite(ledpins[i], LOW); // Switch all LEDs off
553         }
554         operpress = operpress + 1; // Move to next operate state
555         if (operpress == 5) {
556           operpress = 0;           // Cease operations
557         }
558         if (operpress != 0){
559           digitalWrite((ledpins[operpress]), HIGH);  // Prep to operate Servo - Green LED on
560         }
```

```
561         #ifdef DEBUG
562           Serial.print("Operate Press = ");
563           Serial.println(operpress, DEC);
564         #endif
565         while (digitalRead(sw_oper) == LOW){
566           delay(20);              // Wait for Operate switch to be released
567         }
568       }
569     }
570   //  ==========  Fetch switch states to centralise all Servos or perform Remote Reset  ============
571     if (digitalRead(sw_left) == LOW && busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress == 0) {
572       delay(20);
573       if (digitalRead(sw_left) == LOW) {
574         //Left switch pressed
575         while (digitalRead(sw_left) == LOW){
576           if (digitalRead(sw_rite) == LOW) {
577              delay(20);
578             if (digitalRead(sw_rite) == LOW) {
579               gotocentr = 1;  // Right switch pressed also
580             }
581           }
582           if (digitalRead(sw_oper) == LOW) {
583              delay(20);
584             if (digitalRead(sw_oper) == LOW) {
585               gotoreset = 1;  // Operate switch pressed also
586               if (digitalRead(sw_slct) == LOW) {
587                  delay(20);
588                 if (digitalRead(sw_slct) == LOW) {
589                   // Select switch pressed while both Left and Operate switches pressed
590                   if (DCC.getCV(50) != 0) {
591                     DCC.setCV(50, 0); // Set CV50 to load default CV values after next reset (unless already = 0)
592                     delay(5);
593                     #ifdef DEBUG
594                       Serial.print("Set to load Default CVs, CV50 = ");
595                       Serial.println(DCC.getCV(50), DEC);
596                     #endif
597                   }
598                 }
599               }
600             }
```

```
601           }
602         }   // Wait for Left switch to be released
603       }
604     }
605     while (digitalRead(sw_oper) == LOW){
606        delay(20);
607       // Wait until Operate switch released
608     }
609     if (gotoreset == 1) {
610       resetFunc();  // Call system reset - execution stops here then returns to start
611     }
612     if (digitalRead(sw_rite) == LOW && slctpress == 0 && operpress == 0) {
613       delay(20);
614       if (digitalRead(sw_rite) == LOW) {
615         //Right switch pressed
616         while (digitalRead(sw_rite) == LOW){
617           if (digitalRead(sw_left) == LOW) {
618              delay(20);
619             if (digitalRead(sw_left) == LOW) {
620               gotocentr = 1;   // Left switch pressed also
621             }
622           }
623         }   // Wait for Right switch to be released
624       }
625     }
626   //  ==========  Fetch switch states to set Servo Left/Right Limits or Slow Rate  ============
627     if (digitalRead(sw_left) == LOW && busy_dcc == 0 && prog_actv == 0 && ritepress == 0 && gotocentr != 1) {
628       delay(20);
629       if (digitalRead(sw_left) == LOW && digitalRead(sw_rite) == HIGH) {
630         leftpress = 1;         // Left switch pressed (& not Right switch)
631       }
632     }
633     if (digitalRead(sw_rite) == LOW && leftpress == 0 && gotocentr != 1) {
634       delay(20);
635       if (digitalRead(sw_rite) == LOW && digitalRead(sw_left) == HIGH) {
636         ritepress = 1;         // Right switch pressed (& not Left switch)
637       }
638     }
639
640   //  ==========  Set all Servos to central position (90 degrees)  ==========
```

```
641     if (busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress == 0 && gotocentr == 1) {
642       for(i = 0; i < 4; i++){
643         servo[i].attach(srvpins[i]);
644         servo[i].write(servcentr); // Initialise all servo positions
645         delay(100);
646         servo[i].write(servcentr);
647         delay(100);
648         servoact[i].curr_position = servcentr;
649         servo[i].detach();
650       }
651       #ifdef DEBUG
652         Serial.print("Servos 1-4 Position = ");
653         Serial.println(servcentr, DEC);
654       #endif
655       leftpress = 0;
656       ritepress = 0;
657       gotocentr = 0;
658       delay(200);
659     }
660   //  ==========  Set up Left and Right Limits for each Servo  ==========
661     if (busy_dcc == 0 && slctpress > 0 && slctpress < 17 && (slctstep == 0 || slctstep == 1)) {
662       sindex = (slctpress - 1);
663       sindex = sindex >> 2;             // Servo address (0 - 3)
664       movedirn = slctpress % 2; // 1 = Left, 0 = Right
665       if (leftpress == 1) {
666         if (movedirn == 1 && servoact[sindex].left_limit != 180) {
667           servoact[sindex].left_limit = servoact[sindex].left_limit + 1; // Adjust servo left position up
668           servo[sindex].write(servoact[sindex].left_limit);
669           #ifdef DEBUG
670             Serial.print("Servo ");
671             Serial.print(sindex + 1);
672             Serial.print(" Left = ");
673             Serial.println(servoact[sindex].left_limit, DEC);
674           #endif
675         }
676         else if (movedirn == 0 && servoact[sindex].right_limit != 180) {
677           servoact[sindex].right_limit = servoact[sindex].right_limit + 1; // Adjust servo right position up
678           servo[sindex].write(servoact[sindex].right_limit);
679           #ifdef DEBUG
680             Serial.print("Servo ");
```

```
            Serial.print(sindex + 1);
            Serial.print(" Right = ");
            Serial.println(servoact[sindex].right_limit, DEC);
          #endif
        }
      }
      if (ritepress == 1) {
        if (movedirn == 1 && servoact[sindex].left_limit != 0) {
          servoact[sindex].left_limit = servoact[sindex].left_limit - 1; // Adjust servo left position down
          servo[sindex].write(servoact[sindex].left_limit);
          #ifdef DEBUG
            Serial.print("Servo ");
            Serial.print(sindex + 1);
            Serial.print(" Left = ");
            Serial.println(servoact[sindex].left_limit, DEC);
          #endif
        }
        else if (movedirn == 0 && servoact[sindex].right_limit != 0) {
          servoact[sindex].right_limit = servoact[sindex].right_limit - 1; // Adjust servo right position down
          servo[sindex].write(servoact[sindex].right_limit);
          #ifdef DEBUG
            Serial.print("Servo ");
            Serial.print(sindex + 1);
            Serial.print(" Right = ");
            Serial.println(servoact[sindex].right_limit, DEC);
          #endif
        }
      }
      leftpress = 0;
      ritepress = 0;
      delay(150);
    }

  //  ==========  Set up Slow Rate (transit time) for each Servo  ==========
    if (busy_dcc == 0 && slctpress > 2 && slctpress < 17 && slctstep == 2) {
      sindex = (slctpress - 1);
      sindex = sindex >> 2;             // Servo address (0 - 3)
      if (leftpress == 1) {
        if (servoact[sindex].slow_rate < 16) {
          servoact[sindex].slow_rate = servoact[sindex].slow_rate + 1; //Adjust servo rate up (slow - Leisurely)
```

```
721           setup_servo_move (sindex, 1); // Selected Servo to move left at set Slow Rate
722           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
723             exec_servo_move ();
724             delay(3);
725           }
726           delay(150);
727           setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
728           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yetcompleted
729             exec_servo_move ();
730             delay(3);
731           }
732           delay(150);
733           busy_dcc = 0; // Allow configuration switch operations again
734           #ifdef DEBUG
735             Serial.print("Servo ");
736             Serial.print(sindex + 1);
737             Serial.print(" Slow Rate = ");
738             Serial.println(servoact[sindex].slow_rate, DEC);
739           #endif
740         }
741       }
742       if (ritepress == 1) {
743         if (servoact[sindex].slow_rate > 1) {
744           servoact[sindex].slow_rate = servoact[sindex].slow_rate - 1; //Adjust servo rate down (fast - Rapid)
745           setup_servo_move (sindex, 1); // Selected Servo to move left at set Slow Rate
746           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
747             exec_servo_move ();
748             delay(3);
749           }
750           delay(150);
751           setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
752           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
753             exec_servo_move ();
754             delay(3);
755           }
756           delay(150);
757           busy_dcc = 0; // Allow configuration switch operations again
758           #ifdef DEBUG
759             Serial.print("Servo ");
760             Serial.print(sindex + 1);
```

```
761             Serial.print(" Slow Rate = ");
762             Serial.println(servoact[sindex].slow_rate, DEC);
763           #endif
764         }
765       }
766       leftpress = 0;
767       ritepress = 0;
768       delay(150);
769     }
770
771   //  ==========  Prepare to Write Servo settings to CVs (EEPROM)  ==========
772     if (busy_dcc == 0 && slctpress > 3 && slctpress < 17 && slctstep == 3) {
773       flash = flash + 1;
774       if (flash == 100){
775         digitalWrite(3, LOW); // Switch off Red LED
776       }
777       else{
778         if (flash == 200){
779           digitalWrite(3, HIGH); // Switch on Red LED
780           flash = 0;
781         }
782       }
783       sindex = (slctpress - 1);
784       sindex = sindex >> 2;              // Servo address (0 - 3)
785       if (cvs_done == 0) {
786         // Selected Servo identified by 'sindex'
787         if (ritepress == 1) {
788           // Save any changed parameters to relevant CV location in EEPROM
789           if (servoact[sindex].slow_rate != byte (DCC.getCV(55+(sindex * 5)))) {
790             DCC.setCV(55 + (sindex * 5), servoact[sindex].slow_rate); // Only save if value changed
791             delay(5);
792           }
793           if (servoact[sindex].right_limit != byte (DCC.getCV(57+(sindex * 5)))) {
794             DCC.setCV(57 + (sindex * 5), servoact[sindex].right_limit);
795             delay(5);
796           }
797           if (servoact[sindex].left_limit != byte (DCC.getCV(58+(sindex * 5)))) {
798             DCC.setCV(58 + (sindex * 5), servoact[sindex].left_limit);
799             delay(5);
800           }
```

```
801           if (servoact[sindex].curr_position != byte (DCC.getCV(59+(sindex * 5)))) {
802             DCC.setCV(59 + (sindex * 5), servoact[sindex].curr_position);
803             delay(5);
804           }
805           #ifdef DEBUG
806             Serial.print("Servo ");
807             Serial.print(sindex + 1, DEC);
808             Serial.println(" Parameters Saved to CVs");
809           #endif
810           digitalWrite(ledpins[sindex + 1], LOW);  // Selected Servo Parameters written - Green LED off
811           ritepress = 0;
812           cvs_done = 1;
813           while (digitalRead(sw_rite) == LOW){
814             delay(20);               // Wait for Right switch to be released
815           }
816         }
817         else if (leftpress == 1) {
818           #ifdef DEBUG
819             Serial.print("Servo ");
820             Serial.print(sindex + 1, DEC);
821             Serial.println(" Parameters Not Saved");
822           #endif
823           digitalWrite(ledpins[sindex + 1], LOW);  // Selected Servo - Green LED off
824           leftpress = 0;
825           cvs_done = 1;
826           while (digitalRead(sw_left) == LOW){
827             delay(20);               // Wait for Left switch to be released
828           }
829         }
830       } // End CV Writes
831     }
832
833   //  ==========  Operate Servos via switches and set Normal / Reverse ==========
834     if (busy_dcc == 0 && prog_actv == 0 && operpress > 0 && operpress < 5) {
835       sindex = (operpress - 1); // Servo address (0 - 3)
836       if (leftpress == 1 && ritepress == 0) {
837         // Check servo current position and only act on command if servo not in required position
838         if (((servoact[sindex].nrm_dirn == 1) && (servoact[sindex].curr_position != servoact[sindex].left_limit))
839               || ((servoact[sindex].nrm_dirn == 0) && (servoact[sindex].curr_position != servoact[sindex].right_limit))) {
840           setup_servo_move (sindex, 1); // Selected Servo to move left at set Slow Rate
```

```
841           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
842             exec_servo_move ();
843             delay(3);
844           }
845           delay(150);
846           busy_dcc = 0; // Allow configuration switch operations again
847           #ifdef DEBUG
848             Serial.print("Operate Servo ");
849             Serial.print(sindex + 1);
850             Serial.print(" Left = ");
851             Serial.println(servoact[sindex].left_limit, DEC);
852           #endif
853         }
854         leftpress = 0;
855       }
856       if (ritepress == 1 && leftpress == 0) {
857         // Check servo current position and only act on command if servo not in required position
858         if (((servoact[sindex].nrm_dirn == 1) && (servoact[sindex].curr_position != servoact[sindex].right_limit))
859               || ((servoact[sindex].nrm_dirn == 0) && (servoact[sindex].curr_position != servoact[sindex].left_limit))) {
860           setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
861           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
862             exec_servo_move ();
863             delay(3);
864           }
865           delay(150);
866           busy_dcc = 0; // Allow configuration switch operations again
867           #ifdef DEBUG
868             Serial.print("Operate Servo ");
869             Serial.print(sindex + 1);
870             Serial.print(" Right = ");
871             Serial.println(servoact[sindex].right_limit, DEC);
872           #endif
873         }
874         ritepress = 0;
875       }
876       if (digitalRead(sw_slct) == LOW && slctpress == 0) {
877         delay(20);
878         if (digitalRead(sw_slct) == LOW) {
879           nrm_rev = 1;            // Note that Select switch has been pressed
880         }
```

```
881         while (digitalRead(sw_slct) == LOW){
882           delay(20);              // Wait for Select switch to be released
883         }
884         if (nrm_rev == 1) {
885           if (servoact[sindex].nrm_dirn == 0) {
886             servoact[sindex].nrm_dirn = 1; // Select pressed  - so reverse Servo operate direction
887           }
888           else {
889             servoact[sindex].nrm_dirn = 0;
890           }
891           nrm_rev = 0;
892           digitalWrite(ledpins[0], HIGH); // Switch Red LED on
893           delay(600);                     // .. for 0.6 sec then
894           digitalWrite(ledpins[0], LOW);  // .. switch Red LED off again
895           // Save changed Normal/Reverse Direction to relevant CV location in EEPROM
896           if (servoact[sindex].nrm_dirn != byte (DCC.getCV(56+(sindex * 5)))) {
897             DCC.setCV(56 + (sindex * 5), servoact[sindex].nrm_dirn);
898             delay(5);
899           }
900           #ifdef DEBUG
901             Serial.print("Servo ");
902             Serial.print(sindex + 1);
903             if (servoact[sindex].nrm_dirn == 1) {
904               Serial.println(" Set Normal");
905             }
906             else {
907               Serial.println(" Set Reverse");
908             }
909           #endif
910         }
911       }
912       delay(150);
913     }
914
915   //  ==========  Operate Servos in response to received DCC Turnout Command  ==========
916     if (slctpress == 0 && operpress == 0) {
917       // Call procedure to execute set up DCC Turnout Command(s) for all Servos
918       exec_servo_move ();
919       if (actv_indx == 0) {
920         busy_dcc = 0; // Allow configuration switch operations again if no Servo moving
```

```
921       }
922     } // End execute servo position commands
923
924   //  ==========  Complete Servo Initialisation after 150ms delay  ==========
925     if(servdetch != 0xFF) {
926       servdetch--;
927       if(servdetch == 0) {
928         for ( i=0; i < numsrvpins; i++) {
929           servo[i].detach();  // Removes drive from servo after position reached
930           servoact[i].active = 0;
931         }
932         servdetch = 0xFF; // Mark servo initialisation complete
933         digitalWrite(ledpins[0], LOW);  // Switch Red LED off
934         #ifdef DEBUG
935           Serial.println("Servos - Positioned & Detached");
936         #endif
937       }
938     }
939
940   } // End main loop()
941
942   //*************************************************************************************************
943
944   // This function is called whenever a normal DCC Turnout Packet is received in Output Addressing Mode
945   extern void notifyDccAccTurnoutOutput( uint16_t Addr, uint8_t Direction, uint8_t OutputPower ) {
946     int Set_Addr;
947     byte servo_sel;
948     byte index;
949     if (prog_actv == 1 && prog_done == 0) {
950       // Address Programming is active -
951       // Received Address is accepted as Output Address for the selected Servo
952       index = slctpress - 21;
953       if (Addr > 2043) {
954         Addr = 0; // Only accept valid accessory addresses
955       }
956       set_adr[index] = Addr;
957       digitalWrite(ledpins[0], LOW);  // Switch Red LED off
958       delay(600);                     // .. for 0.6 sec then
959       digitalWrite(ledpins[0], HIGH); // .. switch Red LED on again
960       #ifdef DEBUG
```

```
961           Serial.print("Address = ");
962           Serial.print(Addr, DEC);
963           Serial.print(" Entered for Output ");
964           Serial.println(index + 1, DEC);
965         #endif
966         slctpress = slctpress + 1; // Prepare for entry of next Address
967         // Setup states -
968         // 21 - Set Output Address 1,  22 - Output Address 2
969         // 23 - Set Output Address 3,  24 - Output Address 4
970         if (slctpress == 25){
971           for(i = 0; i < numledpins; i++){
972             digitalWrite(ledpins[i], LOW);     // Switch all LEDs off
973           }
974           // Transfer entered Output Addresses (if any) to CV41-CV48
975           for(i = 0; i < 4; i++){
976             if (set_adr[i] != 0) {
977               if ((DCC.getCV(41 + (i*2)) + (256 * DCC.getCV(42 + (i*2)))) != set_adr[i]) {
978                 DCC.setCV(41 + (i*2), byte(set_adr[i] % 256));
979                 delay(5);
980                 DCC.setCV(42 + (i*2), byte(set_adr[i] >> 8));
981                 delay(5);
982                 #ifdef DEBUG
983                   Serial.print("Output Addr ");
984                   Serial.print(i + 1, DEC);
985                   Serial.print(" Programmed = ");
986                   Serial.println(set_adr[i], DEC);
987                 #endif
988               }
989             }
990           }
991           for(i = 0; i < 4; i++){
992             set_adr[i] = 0; // Clear any entered address data
993           }
994           prog_done = 1;  // Prevent re-entry or setup operations until programming link removed
995           slctpress = 0;  // Programming complete
996           flash = 0;      // Prepare to blink Red LED
997           #ifdef DEBUG
998             Serial.println("Addr Programming Complete");
999           #endif
1000        }
```

```
    if (slctpress > 20 && slctpress < 25){
      for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW);     // Ensure all LEDs off
      }
      digitalWrite(ledpins[0], HIGH);      // Switch Red LED on
      digitalWrite(ledpins[slctpress - 20], HIGH);  // Show Servo(0-3) ready for Address - Green LED (1-4)
    }
  }
  else if (prog_done == 0) {
    servo_sel = numsrvpins; // No servo selected initially
    for (index=0; index < numsrvpins; index++) {
      Set_Addr = DCC.getCV(41 + (index*2)) + (256 * DCC.getCV(42 + (index*2)));
      if (Set_Addr == Addr) {
        servo_sel = index; // Identify servo with address matching received DCC command
      }
    }
    if (servo_sel < numsrvpins) {
      #ifdef DEBUG
         Serial.print("Address = ");
         Serial.println(Addr);
         Serial.print("Direction = ");
         if (Direction == 1) {
           Serial.println("Normal (Straight)");
         }
         else {
           Serial.println("Route (Diverging)");
         }
         Serial.print("Matched Servo = ");
         Serial.println(servo_sel + 1); // Servos numbered 1 to 4 (index 0 to 3)
      #endif
      setup_servo_move(servo_sel, Direction );
      // Direction = 1 means go Left, unless Servo set Reverse
    }
    else {
      #ifdef DEBUG
         Serial.print("Addr = ");
         Serial.println(Addr);
         Serial.println("No Matched Servo");
      #endif
    }
```

```
1041    }
1042  }
1043
1044  // This function is called whenever a DCC Turnout Command matches an Output Address
1045  // to determine the direction to move the addressed Servo at the set rate
1046  void setup_servo_move (int srv_num, int accsy_dirn)  {
1047    byte pin_num;
1048    pin_num = srvpins[srv_num];
1049    if (servoact[srv_num].active == 0)  {
1050      servoact[srv_num].active = 1;   // Activate the addressed Servo
1051      servo[srv_num].attach(pin_num);
1052    }
1053    // Determine whether to move to the set  right or left limit, depending on
1054    // the commanded direction and whether the servo is set as reversed
1055    if (accsy_dirn==1) {
1056      if (servoact[srv_num].nrm_dirn == 1) {
1057        servoact[srv_num].end_value = servoact[srv_num].left_limit;
1058      }
1059      else {
1060        servoact[srv_num].end_value = servoact[srv_num].right_limit;
1061      }
1062    }
1063    else {
1064      if (servoact[srv_num].nrm_dirn == 1) {
1065        servoact[srv_num].end_value = servoact[srv_num].right_limit;
1066      }
1067      else {
1068        servoact[srv_num].end_value = servoact[srv_num].left_limit;
1069      }
1070    }
1071    servoact[srv_num].loop_count = servoact[srv_num].slow_rate;
1072  } // End setup_servo_move
1073
1074  // This function is called to execute a prepared DCC Turnout Command and
1075  // move the selected Servo to the appropriate end point at the set rate
1076  void exec_servo_move () {
1077    actv_mask = 1;  // Set execute mask to bit 0 (Servo 1)
1078    for (int i=0; i < numsrvpins; i++) {
1079      if (servoact[i].active == 1)  {                    // DCC command pending for this servo
1080        actv_indx = actv_indx | actv_mask;               // Note that this servo is moving
```

```
1081        busy_dcc = 1;                                   // Prevent configuration switch operations
1082        if (servoact[i].loop_count == 0) {              // Act on pending DCC command
1083          if (servoact[i].curr_position < servoact[i].end_value) {
1084            // Move Left ie. increment Current Position
1085            servoact[i].curr_position = servoact[i].curr_position + 1;
1086          }
1087          else if (servoact[i].curr_position > servoact[i].end_value) {
1088            // Move Right ie. decrement Current Position
1089            servoact[i].curr_position = servoact[i].curr_position - 1;
1090          }
1091          if (servoact[i].curr_position == servoact[i].end_value) { // DCC command completed
1092            actv_indx = actv_indx & (~actv_mask); // Note that this servo has stopped moving
1093            servoact[i].active = 0;
1094            servo[i].detach();
1095            if (servoact[i].curr_position != byte (DCC.getCV(59+(i * 5)))) {
1096              DCC.setCV(59 + (i * 5), servoact[i].curr_position); // Save new Current Position
1097              delay(5);
1098            }
1099          }
1100          else {
1101            servo[i].write(servoact[i].curr_position);      // Move towards end position
1102            servoact[i].loop_count = servoact[i].slow_rate; // Prepare for next move
1103          }
1104        }
1105        else {
1106          servoact[i].loop_count = servoact[i].loop_count - 1; // Decrement Slow Rate count
1107        }
1108      } // End if this servo active
1109      actv_mask = actv_mask << 1; // Shift execute mask to next Servo
1110    } // End for all servos
1111  } // End exec_servo_move
1112
1113  // This call-back function is called by the NmraDcc library when a DCC ACK needs to be sent in response
1114  // to a Read/Verify or Program command when in Service Mode (connected to a programming track)
1115  // Calling this function will flash all LEDs (ensure that the Keypad is connected to the Quad Servo Decoder)
1116  // in order to increase the current drain on the power supply by approximately 68mA for 8 msec
1117
1118  void notifyCVAck(void) {
1119    for (int i=0; i < numledpins; i++) {
1120      digitalWrite(ledpins[i], HIGH); // Switch all LEDs on
```

```
1121    }
1122    delay(8);                          // .. wait for 8 msec
1123    for (int i=0; i < numledpins; i++) {
1124      digitalWrite(ledpins[i], LOW);  // .. then switch all LEDs off
1125    }
1126  }
1127
```