```
// 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 qsdd_ver = 83; // Software Version = 5.3 (0x53)
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
```

```
38  byte  ritepress = 0; // Right switch operation state
39  byte  operpress = 0; // Operate switch operation state
40  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}, //Acc Decoder Long Addr
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
```

```
    {34, 0},    //  Not Used
    {35, 0},    //  Not Used
    {36, 0},    //  Not Used
    {37, 0},    //  Not Used
    {38, 0},    //  Not Used
    {39, 0},    //  Not Used
    {40, 0},    //  Not Used
    {41, 1},    //  Servo 1 Address LSB (01)
    {42, 0},    //  Servo 1 Address MSB
    {43, 2},    //  Servo 2 Address LSB (02)
    {44, 0},    //  Servo 2 Address MSB
    {45, 3},    //  Servo 3 Address LSB (03)
    {46, 0},    //  Servo 3 Address MSB
    {47, 4},    //  Servo 4 Address LSB (04)
    {48, 0},    //  Servo 4 Address MSB
    {49, 0},    //  Not Used
    {50, 0},    //  Reset to default CVs if CV50 not equal to 173 (0xAD = "All Default")
    {51, 0},    //  Not Used
    {52, 0},    //  Not Used
    {53, 0},    //  Not Used
    {54, 0},    //  Not Used
    {55, 6},    //  Servo 1 - Slow Rate (1 to 16)
    {56, 1},    //  Direction - 1 = Normal Operation, 0 = Reverse Operation
    {57, 70},   //  Right Limit
    {58, 110},  //  Left Limit
    {59, 70},   //  Current Position
    {60, 6},    //  Servo 2 - Slow Rate (1 to 16)
    {61, 1},    //  Direction - 1 = Normal Operation, 0 = Reverse Operation
    {62, 70},   //  Right Limit
    {63, 110},  //  Left Limit
    {64, 70},   //  Current Position
    {65, 6},    //  Servo 3 - Slow Rate (1 to 16)
    {66, 1},    //  Direction - 1 = Normal Operation, 0 = Reverse Operation
    {67, 70},   //  Right Limit
    {68, 110},  //  Left Limit
    {69, 70},   //  Current Position
    {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, qsdd_ver},  //  Software Version
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 (qsdd_ver != byte (DCC.getCV(112))) {
221       DCC.setCV(112, qsdd_ver); // 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);
281       if (digitalRead(adr_prog) == LOW) {
282         // Address Programming is active
283         prog_actv = 1;
284         busy_dcc = 1;
285         slctpress = 21; // Prepare to select servo for received Output Address
286         for(i = 0; i < numledpins; i++){
287           digitalWrite(ledpins[i], LOW); // Ensure all LEDs off
288         }
289         digitalWrite(ledpins[0], HIGH);           // Switch Red LED on
290         digitalWrite(ledpins[slctpress - 20], HIGH);  // Show selected Servo - Green LED (1-4)
291         #ifdef DEBUG
292           Serial.println("Addr Programming Active");
293         #endif
294       }
295     }
296     if (digitalRead(adr_prog) == HIGH && prog_actv == 1) {
```

```
297       delay(20);
298       if (digitalRead(adr_prog) == HIGH) {
299         // Programming Link removed - Address Programming will be abandoned
300         for(i = 0; i < 4; i++){
301           set_adr[i] = 0; // Clear any entered address data
302         }
303         for(i = 0; i < numledpins; i++){
304           digitalWrite(ledpins[i], LOW); // Ensure all LEDs off
305         }
306         prog_actv = 0;  // Disable Address Programming
307         prog_done = 0;
308   //      slctpress = 0;  // Reset all switch states
309   //      operpress = 0;
310   //      leftpress = 0;
311   //      ritepress = 0;
312         busy_dcc = 0;   // Allow normal switch actions
313         #ifdef DEBUG
314           Serial.println("Addr Programming Inactive");
315         #endif
316         resetFunc();  // Call system reset - execution stops here then returns to start
317       }
318     }
319     if (prog_done == 1) {
320       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             delay(150);
443             servo[sindex].detach(); // Remove drive from servo
444           }
```

```
445         if (slctstep == 2){
446           // Servo Rate setup (slctpress = 3, 7, 11 or 15)
447           for(i = 0; i < numledpins; i++){
448             digitalWrite(ledpins[i], LOW);          // Ensure all LEDs off
449           }
450           sindex = (slctpress - 1);
451           sindex = sindex >> 2;             // Servo address (0 - 3)
452           movedirn = slctpress % 2; // 1 = Left, 0 = Right
453           digitalWrite(ledpins[0], HIGH);           // Switch Red LED on
454           digitalWrite(ledpins[1 + sindex], HIGH);  // Setup Servo(0-3) Rates - Green LED (1-4)
455           setup_servo_move (sindex, 1);             // Selected Servo to move left at set Slow Rate
456           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not completed
457             exec_servo_move ();
458             delay(3);
459           }
460           delay(150);
461           setup_servo_move (sindex, 0);             // Selected Servo to move right at set Slow Rate
462           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not completed
463             exec_servo_move ();
464             delay(3);
465           }
466           delay(150);
467           busy_dcc = 0;   // Allow configuration switch operations again
468         }
469         if (slctstep == 3){
470           // Write Servo setup parameters to CVs (slctpress = 4, 8, 12 or 16)
471           for(i = 0; i < numledpins; i++){
472             digitalWrite(ledpins[i], LOW);          // Ensure all LEDs off
473           }
474           digitalWrite(ledpins[0], HIGH); // Ready to write to CVs - Red LED flashing
475           flash = 0;
476           cvs_done = 0;                   // Prepare to save Servo parameters
477           sindex = (slctpress - 1);
478           sindex = sindex >> 2;           // Servo address (0 - 3)
479           digitalWrite(ledpins[0], HIGH);           // Switch Red LED on
480           digitalWrite(ledpins[1 + sindex], HIGH);  // Setup Servo(0-3) Write CVs - Green LED (1-4)
481         }
```

```
      }
      #ifdef DEBUG
        Serial.print("Select Press = ");
        Serial.print(slctpress, DEC);
        Serial.print(" : Servo = ");
        Serial.print(sindex + 1, DEC);
        Serial.print(" : Step = ");
        Serial.println(slctstep + 1, DEC);
      #endif
    }
  }
//  ==========  Fetch Operate switch state to skip to next Servo setup sequence  ============
  if (digitalRead(sw_oper) == LOW && busy_dcc == 0 && prog_actv == 0 && operpress == 0
                      && (slctpress == 1 || slctpress == 5 || slctpress == 9 || slctpress == 13)) {
    delay(20);
    if (digitalRead(sw_oper) == LOW) {
      sindex = (slctpress - 1);
      sindex = sindex >> 2;      // Servo address (0 - 3)
      servo[sindex].write(servoact[sindex].curr_position);  // Position selected servo to its last saved position
      delay(300);
      slctpress = slctpress + 4; // Move to set up next Servo
      if (slctpress == 17) {
        slctpress = 0;                    // Setup complete
        slctstep = 4;                     // .. clear selection flags
        for(i = 0; i < numledpins; i++){
          digitalWrite(ledpins[i], LOW);  // Switch all LEDs off
        }
        for(sindex = 0; sindex < numsrvpins; sindex++){
          servo[sindex].detach();         // Remove drive from all servos
        }
        sindex = 0;
        movedirn = 0;
        cvs_done = 1;
      }
      else {
        slctstep = 0;     // Set first step in each Servo setup sequence
        // Servo Left Limit setup (slctpress = 1, 5, 9, or 13)
```

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

```
556           digitalWrite(ledpins[i], LOW); // Switch all LEDs off
557         }
558         operpress = operpress + 1; // Move to next operate state
559         if (operpress == 5) {
560           operpress = 0;           // Cease operations
561         }
562         if (operpress != 0){
563           digitalWrite((ledpins[operpress]), HIGH);  // Prep to operate Servo - Green LED on
564         }
565         #ifdef DEBUG
566           Serial.print("Operate Press = ");
567           Serial.println(operpress, DEC);
568         #endif
569         while (digitalRead(sw_oper) == LOW){
570           delay(20);              // Wait for Operate switch to be released
571         }
572       }
573     }
574   //  ==========  Fetch switch states to centralise all Servos or perform Remote Reset  ============
575     if (digitalRead(sw_left) == LOW && busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress == 0) {
576       delay(20);
577       if (digitalRead(sw_left) == LOW) {
578         //Left switch pressed
579         while (digitalRead(sw_left) == LOW){
580           if (digitalRead(sw_rite) == LOW) {
581              delay(20);
582             if (digitalRead(sw_rite) == LOW) {
583               gotocentr = 1;  // Right switch pressed also
584             }
585           }
586           if (digitalRead(sw_oper) == LOW) {
587              delay(20);
588             if (digitalRead(sw_oper) == LOW) {
589               gotoreset = 1;  // Operate switch pressed also
590               if (digitalRead(sw_slct) == LOW) {
591                  delay(20);
592                 if (digitalRead(sw_slct) == LOW) {
```

```
                // Select switch pressed while both Left and Operate switches pressed
                if (DCC.getCV(50) != 0) {
                  DCC.setCV(50, 0); // Set CV50 to load default CV values after next reset (unless already = 0)
                  delay(5);
                  #ifdef DEBUG
                    Serial.print("Set to load Default CVs, CV50 = ");
                    Serial.println(DCC.getCV(50), DEC);
                  #endif
                }
              }
            }
          }
        }
      }   // Wait for Left switch to be released
    }
  }
  while (digitalRead(sw_oper) == LOW){
     delay(20);
    // Wait until Operate switch released
  }
  if (gotoreset == 1) {
    resetFunc();  // Call system reset - execution stops here then returns to start
  }
  if (digitalRead(sw_rite) == LOW && slctpress == 0 && operpress == 0) {
    delay(20);
    if (digitalRead(sw_rite) == LOW) {
      //Right switch pressed
      while (digitalRead(sw_rite) == LOW){
        if (digitalRead(sw_left) == LOW) {
           delay(20);
          if (digitalRead(sw_left) == LOW) {
            gotocentr = 1;   // Left switch pressed also
          }
        }
      }   // Wait for Right switch to be released
    }
  }
```

```
630  //  ==========  Fetch switch states to set Servo Left/Right Limits or Slow Rate  ============
631    if (digitalRead(sw_left) == LOW && busy_dcc == 0 && prog_actv == 0 && ritepress == 0 && gotocentr != 1) {
632      delay(20);
633      if (digitalRead(sw_left) == LOW && digitalRead(sw_rite) == HIGH) {
634        leftpress = 1;          // Left switch pressed (& not Right switch)
635      }
636    }
637    if (digitalRead(sw_rite) == LOW && leftpress == 0 && gotocentr != 1) {
638      delay(20);
639      if (digitalRead(sw_rite) == LOW && digitalRead(sw_left) == HIGH) {
640        ritepress = 1;          // Right switch pressed (& not Left switch)
641      }
642    }
643
644  //  ==========  Set all Servos to central position (90 degrees)  ==========
645    if (busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress == 0 && gotocentr == 1) {
646      for(i = 0; i < 4; i++){
647        servo[i].attach(srvpins[i]);
648        servo[i].write(servcentr); // Initialise all servo positions
649        delay(100);
650        servo[i].write(servcentr);
651        delay(100);
652        servoact[i].curr_position = servcentr;
653        servo[i].detach();
654      }
655      #ifdef DEBUG
656        Serial.print("Servos 1-4 Position = ");
657        Serial.println(servcentr, DEC);
658      #endif
659      leftpress = 0;
660      ritepress = 0;
661      gotocentr = 0;
662      delay(200);
663    }
664  //  ==========  Set up Left and Right Limits for each Servo  ==========
665    if (busy_dcc == 0 && slctpress > 0 && slctpress < 17 && (slctstep == 0 || slctstep == 1)) {
666      sindex = (slctpress - 1);
```

```
667       sindex = sindex >> 2;             // Servo address (0 - 3)
668       movedirn = slctpress % 2; // 1 = Left, 0 = Right
669       servo[sindex].attach(srvpins[sindex]);
670       if (leftpress == 1) {
671         if (movedirn == 1 && servoact[sindex].left_limit != 180) {
672           servoact[sindex].left_limit = servoact[sindex].left_limit + 1; // Adjust servo left position up
673           servo[sindex].write(servoact[sindex].left_limit);
674           #ifdef DEBUG
675             Serial.print("Servo ");
676             Serial.print(sindex + 1);
677             Serial.print(" Left = ");
678             Serial.println(servoact[sindex].left_limit, DEC);
679           #endif
680         }
681         else if (movedirn == 0 && servoact[sindex].right_limit != 180) {
682           servoact[sindex].right_limit = servoact[sindex].right_limit + 1; // Adjust servo right position up
683           servo[sindex].write(servoact[sindex].right_limit);
684           #ifdef DEBUG
685             Serial.print("Servo ");
686             Serial.print(sindex + 1);
687             Serial.print(" Right = ");
688             Serial.println(servoact[sindex].right_limit, DEC);
689           #endif
690         }
691       }
692       if (ritepress == 1) {
693         if (movedirn == 1 && servoact[sindex].left_limit != 0) {
694           servoact[sindex].left_limit = servoact[sindex].left_limit - 1; // Adjust servo left position down
695           servo[sindex].write(servoact[sindex].left_limit);
696           #ifdef DEBUG
697             Serial.print("Servo ");
698             Serial.print(sindex + 1);
699             Serial.print(" Left = ");
700             Serial.println(servoact[sindex].left_limit, DEC);
701           #endif
702         }
703         else if (movedirn == 0 && servoact[sindex].right_limit != 0) {
```

```
704           servoact[sindex].right_limit = servoact[sindex].right_limit - 1; // Adjust servo right position down
705           servo[sindex].write(servoact[sindex].right_limit);
706           #ifdef DEBUG
707             Serial.print("Servo ");
708             Serial.print(sindex + 1);
709             Serial.print(" Right = ");
710             Serial.println(servoact[sindex].right_limit, DEC);
711           #endif
712         }
713       }
714       leftpress = 0;
715       ritepress = 0;
716       delay(150);
717       servo[sindex].detach();
718     }
719
720   //  ==========  Set up Slow Rate (transit time) for each Servo  ==========
721     if (busy_dcc == 0 && slctpress > 2 && slctpress < 17 && slctstep == 2) {
722       sindex = (slctpress - 1);
723       sindex = sindex >> 2;             // Servo address (0 - 3)
724       if (leftpress == 1) {
725         if (servoact[sindex].slow_rate < 16) {
726           servoact[sindex].slow_rate = servoact[sindex].slow_rate + 1; //Adjust servo rate up (slow - Leisurely)
727           setup_servo_move (sindex, 1); // Selected Servo to move left at set Slow Rate
728           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
729             exec_servo_move ();
730             delay(3);
731           }
732           delay(150);
733           setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
734           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yetcompleted
735             exec_servo_move ();
736             delay(3);
737           }
738           delay(150);
739           busy_dcc = 0; // Allow configuration switch operations again
740           #ifdef DEBUG
```

```
741             Serial.print("Servo ");
742             Serial.print(sindex + 1);
743             Serial.print(" Slow Rate = ");
744             Serial.println(servoact[sindex].slow_rate, DEC);
745           #endif
746         }
747       }
748       if (ritepress == 1) {
749         if (servoact[sindex].slow_rate > 1) {
750           servoact[sindex].slow_rate = servoact[sindex].slow_rate - 1; //Adjust servo rate down (fast - Rapid)
751           setup_servo_move (sindex, 1); // Selected Servo to move left 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           setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
758           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
759             exec_servo_move ();
760             delay(3);
761           }
762           delay(150);
763           busy_dcc = 0; // Allow configuration switch operations again
764           #ifdef DEBUG
765             Serial.print("Servo ");
766             Serial.print(sindex + 1);
767             Serial.print(" Slow Rate = ");
768             Serial.println(servoact[sindex].slow_rate, DEC);
769           #endif
770         }
771       }
772       leftpress = 0;
773       ritepress = 0;
774       delay(150);
775     }
776
777   //  ==========  Prepare to Write Servo settings to CVs (EEPROM)  ==========
```

```
778     if (busy_dcc == 0 && slctpress > 3 && slctpress < 17 && slctstep == 3) {
779       flash = flash + 1;
780       if (flash == 100){
781         digitalWrite(3, LOW); // Switch off Red LED
782       }
783       else{
784         if (flash == 200){
785           digitalWrite(3, HIGH); // Switch on Red LED
786           flash = 0;
787         }
788       }
789       sindex = (slctpress - 1);
790       sindex = sindex >> 2;             // Servo address (0 - 3)
791       if (cvs_done == 0) {
792         // Selected Servo identified by 'sindex'
793         if (ritepress == 1) {
794           // Save any changed parameters to relevant CV location in EEPROM
795           if (servoact[sindex].slow_rate != byte (DCC.getCV(55+(sindex * 5)))) {
796             DCC.setCV(55 + (sindex * 5), servoact[sindex].slow_rate); // Only save if value changed
797             delay(5);
798           }
799           if (servoact[sindex].right_limit != byte (DCC.getCV(57+(sindex * 5)))) {
800             DCC.setCV(57 + (sindex * 5), servoact[sindex].right_limit);
801             delay(5);
802           }
803           if (servoact[sindex].left_limit != byte (DCC.getCV(58+(sindex * 5)))) {
804             DCC.setCV(58 + (sindex * 5), servoact[sindex].left_limit);
805             delay(5);
806           }
807           if (servoact[sindex].curr_position != byte (DCC.getCV(59+(sindex * 5)))) {
808             DCC.setCV(59 + (sindex * 5), servoact[sindex].curr_position);
809             delay(5);
810           }
811           #ifdef DEBUG
812             Serial.print("Servo ");
813             Serial.print(sindex + 1, DEC);
814             Serial.println(" Parameters Saved to CVs");
```

```
          #endif
          digitalWrite(ledpins[sindex + 1], LOW);  // Selected Servo Parameters written - Green LED off
          ritepress = 0;
          cvs_done = 1;
          while (digitalRead(sw_rite) == LOW){
            delay(20);                // Wait for Right switch to be released
          }
        }
        else if (leftpress == 1) {
          #ifdef DEBUG
            Serial.print("Servo ");
            Serial.print(sindex + 1, DEC);
            Serial.println(" Parameters Not Saved");
          #endif
          digitalWrite(ledpins[sindex + 1], LOW);  // Selected Servo - Green LED off
          leftpress = 0;
          cvs_done = 1;
          while (digitalRead(sw_left) == LOW){
            delay(20);                // Wait for Left switch to be released
          }
        }
      } // End CV Writes
    }

//  ==========  Operate Servos via switches and set Normal / Reverse ==========
    if (busy_dcc == 0 && prog_actv == 0 && operpress > 0 && operpress < 5) {
      sindex = (operpress - 1); // Servo address (0 - 3)
      if (leftpress == 1 && ritepress == 0) {
        // Check servo current position and only act on command if servo not in required position
        if (((servoact[sindex].nrm_dirn == 1) && (servoact[sindex].curr_position != servoact[sindex].left_limit))
              || ((servoact[sindex].nrm_dirn == 0) && (servoact[sindex].curr_position != servoact[sindex].right_limit))) {
          setup_servo_move (sindex, 1); // Selected Servo to move left at set Slow Rate
          while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
            exec_servo_move ();
            delay(3);
          }
          delay(150);
```

```
852           busy_dcc = 0; // Allow configuration switch operations again
853           #ifdef DEBUG
854             Serial.print("Operate Servo ");
855             Serial.print(sindex + 1);
856             Serial.print(" Left = ");
857             Serial.println(servoact[sindex].left_limit, DEC);
858           #endif
859         }
860         leftpress = 0;
861       }
862       if (ritepress == 1 && leftpress == 0) {
863         // Check servo current position and only act on command if servo not in required position
864         if (((servoact[sindex].nrm_dirn == 1) && (servoact[sindex].curr_position != servoact[sindex].right_limit))
865               || ((servoact[sindex].nrm_dirn == 0) && (servoact[sindex].curr_position != servoact[sindex].left_limit))) {
866           setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
867           while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
868             exec_servo_move ();
869             delay(3);
870           }
871           delay(150);
872           busy_dcc = 0; // Allow configuration switch operations again
873           #ifdef DEBUG
874             Serial.print("Operate Servo ");
875             Serial.print(sindex + 1);
876             Serial.print(" Right = ");
877             Serial.println(servoact[sindex].right_limit, DEC);
878           #endif
879         }
880         ritepress = 0;
881       }
882       if (digitalRead(sw_slct) == LOW && slctpress == 0) {
883         delay(20);
884         if (digitalRead(sw_slct) == LOW) {
885           nrm_rev = 1;            // Note that Select switch has been pressed
886         }
887         while (digitalRead(sw_slct) == LOW){
888           delay(20);              // Wait for Select switch to be released
```

```
889         }
890         if (nrm_rev == 1) {
891           if (servoact[sindex].nrm_dirn == 0) {
892             servoact[sindex].nrm_dirn = 1; // Select pressed  - so reverse Servo operate direction
893           }
894           else {
895             servoact[sindex].nrm_dirn = 0;
896           }
897           nrm_rev = 0;
898           digitalWrite(ledpins[0], HIGH); // Switch Red LED on
899           delay(600);                     // .. for 0.6 sec then
900           digitalWrite(ledpins[0], LOW);  // .. switch Red LED off again
901           // Save changed Normal/Reverse Direction to relevant CV location in EEPROM
902           if (servoact[sindex].nrm_dirn != byte (DCC.getCV(56+(sindex * 5)))) {
903             DCC.setCV(56 + (sindex * 5), servoact[sindex].nrm_dirn);
904             delay(5);
905           }
906           #ifdef DEBUG
907             Serial.print("Servo ");
908             Serial.print(sindex + 1);
909             if (servoact[sindex].nrm_dirn == 1) {
910               Serial.println(" Set Normal");
911             }
912             else {
913               Serial.println(" Set Reverse");
914             }
915           #endif
916         }
917       }
918       delay(150);
919     }
920
921   //  ==========  Operate Servos in response to received DCC Turnout Command  ==========
922     if (slctpress == 0 && operpress == 0) {
923       // Call procedure to execute set up DCC Turnout Command(s) for all Servos
924       exec_servo_move ();
925       if (actv_indx == 0) {
```

```
926         busy_dcc = 0; // Allow configuration switch operations again if no Servo moving
927       }
928     } // End execute servo position commands
929
930   //  ==========  Complete Servo Initialisation after 150ms delay  ==========
931     if(servdetch != 0xFF) {
932       servdetch--;
933       if(servdetch == 0) {
934         for ( i=0; i < numsrvpins; i++) {
935           servo[i].detach();  // Removes drive from servo after position reached
936           servoact[i].active = 0;
937         }
938         servdetch = 0xFF; // Mark servo initialisation complete
939         digitalWrite(ledpins[0], LOW);  // Switch Red LED off
940         #ifdef DEBUG
941           Serial.println("Servos - Positioned & Detached");
942         #endif
943       }
944     }
945
946   } // End main loop()
947
948   //*************************************************************************************************
949
950   // This function is called whenever a normal DCC Turnout Packet is received in Output Addressing Mode
951   extern void notifyDccAccTurnoutOutput( uint16_t Addr, uint8_t Direction, uint8_t OutputPower ) {
952     int Set_Addr;
953     byte servo_sel;
954     byte index;
955     if (prog_actv == 1 && prog_done == 0) {
956       // Address Programming is active -
957       // Received Address is accepted as Output Address for the selected Servo
958       index = slctpress - 21;
959       if (Addr > 2043) {
960         Addr = 0; // Only accept valid accessory addresses
961       }
962       set_adr[index] = Addr;
```

```
963       digitalWrite(ledpins[0], LOW);  // Switch Red LED off
964       delay(600);                     // .. for 0.6 sec then
965       digitalWrite(ledpins[0], HIGH); // .. switch Red LED on again
966       #ifdef DEBUG
967          Serial.print("Address = ");
968          Serial.print(Addr, DEC);
969          Serial.print(" Entered for Output ");
970          Serial.println(index + 1, DEC);
971       #endif
972       slctpress = slctpress + 1; // Prepare for entry of next Address
973       // Setup states -
974       // 21 - Set Output Address 1,  22 - Output Address 2
975       // 23 - Set Output Address 3,  24 - Output Address 4
976       if (slctpress == 25){
977         for(i = 0; i < numledpins; i++){
978           digitalWrite(ledpins[i], LOW);     // Switch all LEDs off
979         }
980         // Transfer entered Output Addresses (if any) to CV41-CV48
981         for(i = 0; i < 4; i++){
982           if (set_adr[i] != 0) {
983             if ((DCC.getCV(41 + (i*2)) + (256 * DCC.getCV(42 + (i*2)))) != set_adr[i]) {
984               DCC.setCV(41 + (i*2), byte(set_adr[i] % 256));
985               delay(5);
986               DCC.setCV(42 + (i*2), byte(set_adr[i] >> 8));
987               delay(5);
988               #ifdef DEBUG
989                 Serial.print("Output Addr ");
990                 Serial.print(i + 1, DEC);
991                 Serial.print(" Programmed = ");
992                 Serial.println(set_adr[i], DEC);
993               #endif
994             }
995           }
996         }
997         for(i = 0; i < 4; i++){
998           set_adr[i] = 0; // Clear any entered address data
999         }
```

```
1000        prog_done = 1;  // Prevent re-entry or setup operations until programming link removed
1001        slctpress = 0;  // Programming complete
1002        flash = 0;      // Prepare to blink Red LED
1003        #ifdef DEBUG
1004          Serial.println("Addr Programming Complete");
1005        #endif
1006      }
1007      if (slctpress > 20 && slctpress < 25){
1008        for(i = 0; i < numledpins; i++){
1009          digitalWrite(ledpins[i], LOW);     // Ensure all LEDs off
1010        }
1011        digitalWrite(ledpins[0], HIGH);      // Switch Red LED on
1012        digitalWrite(ledpins[slctpress - 20], HIGH);  // Show Servo(0-3) ready for Address - Green LED (1-4)
1013      }
1014    }
1015    else if (prog_done == 0) {
1016      servo_sel = numsrvpins; // No servo selected initially
1017      for (index=0; index < numsrvpins; index++) {
1018        Set_Addr = DCC.getCV(41 + (index*2)) + (256 * DCC.getCV(42 + (index*2)));
1019        if (Set_Addr == Addr) {
1020          servo_sel = index; // Identify servo with address matching received DCC command
1021        }
1022      }
1023      if (servo_sel < numsrvpins) {
1024        #ifdef DEBUG
1025           Serial.print("Address = ");
1026           Serial.println(Addr);
1027           Serial.print("Direction = ");
1028           if (Direction == 1) {
1029             Serial.println("Normal (Straight)");
1030           }
1031           else {
1032             Serial.println("Route (Diverging)");
1033           }
1034           Serial.print("Matched Servo = ");
1035           Serial.println(servo_sel + 1); // Servos numbered 1 to 4 (index 0 to 3)
1036        #endif
```

```
1037        setup_servo_move(servo_sel, Direction );
1038        // Direction = 1 means go Left, unless Servo set Reverse
1039      }
1040      else {
1041        #ifdef DEBUG
1042           Serial.print("Addr = ");
1043           Serial.println(Addr);
1044           Serial.println("No Matched Servo");
1045        #endif
1046      }
1047    }
1048  }
1049
1050  // This function is called whenever a DCC Turnout Command matches an Output Address
1051  // to determine the direction to move the addressed Servo at the set rate
1052  void setup_servo_move (int srv_num, int accsy_dirn)  {
1053    byte pin_num;
1054    pin_num = srvpins[srv_num];
1055    if (servoact[srv_num].active == 0)  {
1056      servoact[srv_num].active = 1;   // Activate the addressed Servo
1057      servo[srv_num].attach(pin_num);
1058    }
1059    // Determine whether to move to the set  right or left limit, depending on
1060    // the commanded direction and whether the servo is set as reversed
1061    if (accsy_dirn==1) {
1062      if (servoact[srv_num].nrm_dirn == 1) {
1063        servoact[srv_num].end_value = servoact[srv_num].left_limit;
1064      }
1065      else {
1066        servoact[srv_num].end_value = servoact[srv_num].right_limit;
1067      }
1068    }
1069    else {
1070      if (servoact[srv_num].nrm_dirn == 1) {
1071        servoact[srv_num].end_value = servoact[srv_num].right_limit;
1072      }
1073      else {
```

```
1074        servoact[srv_num].end_value = servoact[srv_num].left_limit;
1075      }
1076    }
1077    servoact[srv_num].loop_count = servoact[srv_num].slow_rate;
1078  } // End setup_servo_move
1079
1080  // This function is called to execute a prepared DCC Turnout Command and
1081  // move the selected Servo to the appropriate end point at the set rate
1082  void exec_servo_move () {
1083    actv_mask = 1;  // Set execute mask to bit 0 (Servo 1)
1084    for (int i=0; i < numsrvpins; i++) {
1085      if (servoact[i].active == 1)  {                       // DCC command pending for this servo
1086        actv_indx = actv_indx | actv_mask;                  // Note that this servo is moving
1087        busy_dcc = 1;                                       // Prevent configuration switch operations
1088        if (servoact[i].loop_count == 0) {                  // Act on pending DCC command
1089          if (servoact[i].curr_position < servoact[i].end_value) {
1090            // Move Left ie. increment Current Position
1091            servoact[i].curr_position = servoact[i].curr_position + 1;
1092          }
1093          else if (servoact[i].curr_position > servoact[i].end_value) {
1094            // Move Right ie. decrement Current Position
1095            servoact[i].curr_position = servoact[i].curr_position - 1;
1096          }
1097          if (servoact[i].curr_position == servoact[i].end_value) { // DCC command completed
1098            actv_indx = actv_indx & (~actv_mask); // Note that this servo has stopped moving
1099            servoact[i].active = 0;
1100            servo[i].detach();
1101            if (servoact[i].curr_position != byte (DCC.getCV(59+(i * 5)))) {
1102              DCC.setCV(59 + (i * 5), servoact[i].curr_position); // Save new Current Position
1103              delay(5);
1104            }
1105          }
1106          else {
1107            servo[i].write(servoact[i].curr_position);      // Move towards end position
1108            servoact[i].loop_count = servoact[i].slow_rate; // Prepare for next move
1109          }
1110        }
```

```
1111        else {
1112          servoact[i].loop_count = servoact[i].loop_count - 1; // Decrement Slow Rate count
1113        }
1114      } // End if this servo active
1115      actv_mask = actv_mask << 1; // Shift execute mask to next Servo
1116    } // End for all servos
1117  } // End exec_servo_move
1118
1119  // This call-back function is called by the NmraDcc library when a DCC ACK needs to be sent in response
1120  // to a Read/Verify or Program command when in Service Mode (connected to a programming track)
1121  // Calling this function will flash all LEDs (ensure that the Keypad is connected to the Quad Servo Decoder)
1122  // in order to increase the current drain on the power supply by approximately 68mA for 8 msec
1123
1124  void notifyCVAck(void) {
1125    for (int i=0; i < numledpins; i++) {
1126      digitalWrite(ledpins[i], HIGH); // Switch all LEDs on
1127    }
1128    delay(8);                         // .. wait for 8 msec
1129    for (int i=0; i < numledpins; i++) {
1130      digitalWrite(ledpins[i], LOW);  // .. then switch all LEDs off
1131    }
1132  }
1133
```