

```

// 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
byte ritepress = 0; // Right switch operation state
byte operpress = 0; // Operate switch operation state
byte movedirn = 0;
byte sindex = 0;
byte flash = 0;
byte cvs_done = 0;
byte nrm_rev = 0;
byte numsrvpins = 4;
byte srvpins[] = {14,15,16,17}; // Nano pins for Servos
byte numledpins = 5;
byte ledpins[] = {3,4,5,6,7}; // Nano pins for LEDs

```

```

byte numswpins = 4;
byte swpins[] = {9,10,11,12}; // Nano pins for Switches
byte prog_actv = 0;
byte prog_done = 0;
int set_adr[] = {0,0,0,0}; // Temporary storage for Output Addresses

int t; // temp
int i;
int slctstep = 4; // Step in setup of each Servo - initialise to invalid value
byte busy_dcc = 0; // Set = 1 when any DCC command(s) being executed
byte actv_indx = 0; // Bit Set = 1 when DCC command for specific servo being executed
// Bit 0 = Servo 0, Bit 1 = Servo 1, Bit 2 = Servo 2, Bit 3 = Servo 3
byte actv_mask = 0; // Mask to set a specific bit in actv_indx

typedef struct
{
    byte active;
    byte curr_position;
    byte slow_rate;
    byte right_limit;
    byte left_limit;
    byte loop_count;
    byte nrm_dirn;
    byte end_value;
} serv_param; // Servo Parameters structure (block)
serv_param servoact[4]; // Array of Parameters for four Servos

typedef struct
{
    int cv_adr;
    byte cv_val;
} cv_pair;

byte cv_value;
int SET_CV_Address = 24; // This Address is for setting CV'S like a Loco using Ops Mode
int Accessory_Address = 1; // This Address is the Board Address - not necessarily a Servo Address
byte CV_DECODER_MASTER_RESET = 120; // This is the CV Address for Full Reset - load a value of 120
// to this location and then press the Nano Reset button
// Return to default CV values can also be achieved by loading any value
// other than 0xAD (173) to CV50 and then pressing the Nano Reset button
byte CV_To_Store_SET_CV_Address = 121; // The address used to change CVs using Ops Mode (set as 24 above) is
// stored in this location, and in the following CV if greater than 256
byte CV_Accessory_Address = CV_ACCESSORY_DECODER_ADDRESS_LSB; // CV01 - the Board Address

cv_pair FactoryDefaultCVs [] =
{
    // These two CVs define the Long Accessory Address
    {CV_ACCESSORY_DECODER_ADDRESS_LSB, Accessory_Address&0xFF},

```

```

{CV_ACCESSORY_DECODER_ADDRESS_MSB, (Accessory_Address>>8)&0x07},

{CV_MULTIFUNCTION_EXTENDED_ADDRESS_MSB, 0},
{CV_MULTIFUNCTION_EXTENDED_ADDRESS_LSB, 0},

// {CV_29_CONFIG,CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_F0_LOCATION}, // Accesory Decoder Short Address
{CV_29_CONFIG, CV29_ACCESSORY_DECODER|CV29_OUTPUT_ADDRESS_MODE|CV29_EXT_ADDRESSING | CV29_F0_LOCATION}, // Accesory Decoder Long Address

{CV_DECODER_MASTER_RESET, 0},
{CV_To_Store_SET_CV_Address, SET_CV_Address&0xFF }, // LSB Set CV Address
{CV_To_Store_SET_CV_Address+1,(SET_CV_Address>>8)&0x3F }, //MSB Set CV Address
{30, 0}, // Not Used
{31, 0}, // Not Used
{32, 0}, // Not Used
{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)
{71, 1}, // Direction - 1 = Normal Operation, 0 = Reverse Operation
{72, 70}, // Right Limit
{73, 110}, // Left Limit
{74, 70}, // Current Position
{109, 81}, // "Q" - Extended Decoder Version
{110, 83}, // "S"
{111, 68}, // "D"
{112, qsdd_ver}, // Software Version
};
uint8_t FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
void notifyCVResetFactoryDefault()
{
    // Make FactoryDefaultCVIndex non-zero and equal to number of CV's to be reset
    // to flag to the loop() function that a reset to Factory Defaults needs to be done
    FactoryDefaultCVIndex = sizeof(FactoryDefaultCVs)/sizeof(cv_pair);
};
void(* resetFunc) (void) = 0; // Declare reset function at address 0

//*****
void setup() {
    Serial.begin(115200); // Only required if debug messages are to be displayed by serial Monitor

    // Setup which External Interrupt to use for the DCC input, and the associated Pin (2)
    DCC.pin(0, 2, 0);
    // Call the main DCC Init function to enable the DCC Receiver
    DCC.init( MAN_ID_DIY, 601, FLAGS_OUTPUT_ADDRESS_MODE | FLAGS_DCC_ACCESSORY_DECODER, CV_To_Store_SET_CV_Address);
    // delay(150);

    pinMode(adr_prog, INPUT_PULLUP); // Set as Address Programming enable - when connected to GND
    // allows the Output Address of the selected Servo to be set
    // by issuing a DCC Accessory command to that address

    // Initialize the servo digital pins as outputs
    for (int i=0; i < numsrvpins; i++) {
        pinMode(srvpins[i], OUTPUT);
        digitalWrite(srvpins[i], LOW);
    }
    // Initialize the LED digital pins as outputs
    for (int i=0; i < numledpins; i++) {
        pinMode(ledpins[i], OUTPUT);
        digitalWrite(ledpins[i], LOW);
    }
    //for (int i=0; i < numledpins; i++) {
    //  digitalWrite(ledpins[i], HIGH); // Switch on all LEDs in sequence
    //  delay (30);

```

```

//}
//delay(250);
//for (int i=0; i < numledpins; i++) {
//  digitalWrite(ledpins[i], LOW); // .. then switch them off again
//  delay (30);
//}
// Initialize the digital pins connected to the pushbuttons as inputs
for (int i=0; i < numswpins; i++) {
  pinMode(swpins[i], INPUT);
}

if ((DCC.getCV(CV_DECODER_MASTER_RESET)== CV_DECODER_MASTER_RESET) || (DCC.getCV(50) != 0xAD))
{
  // Reset all defined CVs to default values if value of CV120 = 120 or CV50 is not equal to 173 (0xAD = "All Default")
  for (int j=0; j < sizeof(FactoryDefaultCVs)/sizeof(cv_pair); j++) {
    DCC.setCV( FactoryDefaultCVs[j].cv_adr, FactoryDefaultCVs[j].cv_val);
    digitalWrite(3, 1);
    delay (1000);
    digitalWrite(3, 0); // Flash Red LED when CVs loaded
    DCC.setCV(50, 0xAD);
#ifdef DEBUG
    Serial.print("Reset to Default CVs, CV50 = ");
    Serial.println(DCC.getCV(50), DEC) ;
#endif
  }

  // Check that current Software Version is stored in CV 112 in EEPROM
  if (qsdd_ver != byte (DCC.getCV(112))) {
    DCC.setCV(112, qsdd_ver); // Only save if Software Version has been updated
    delay(5);
  }

  // Initialise all attached servos
  for ( i=0; i < numsvpins; i++) {
    cv_value = DCC.getCV(41 + (i*2)) + (256 * DCC.getCV(42 + (i*2)));
#ifdef DEBUG
    Serial.print("CV_Number: ");
    Serial.print(41+(i*2), DEC) ;
    Serial.print(" Servo Address: ");
    Serial.println(cv_value, DEC) ;
#endif
    servoact[i].slow_rate = byte (DCC.getCV(55+(i*5)));
    if (servoact[i].slow_rate > 32) {
      servoact[i].slow_rate = 32; // Check for maximum allowed value
    }
    servoact[i].nrm_dirn = byte (DCC.getCV(56+(i*5)));
    servoact[i].right_limit = byte (DCC.getCV(57+(i*5)));
    if (servoact[i].right_limit > 180) {

```

```

    servoact[i].right_limit = 180;
}
servoact[i].left_limit = byte (DCC.getCV(58+(i*5)));
if (servoact[i].left_limit > 180) {
    servoact[i].left_limit = 180;
}
servoact[i].curr_position = byte (DCC.getCV(59+(i*5)));
if (servoact[i].curr_position > servoact[i].left_limit) {
    servoact[i].curr_position = servoact[i].left_limit; // Check that Current Postion is within Right & Left Limits
}
if (servoact[i].curr_position < servoact[i].right_limit) {
    servoact[i].curr_position = servoact[i].right_limit;
}
// Attaches servo on pin - enables servo to be driven to defined position
servo[i].attach(srvpins[i]);
servo[i].write(servoact[i].curr_position);
// delay(150);
// servo[i].detach(); // Removes drive from servo after position reached
// servoact[i].active = 0;

#ifdef DEBUG
    Serial.print("Initialised Servo ");
    Serial.println(i+1, DEC) ;
#endif
}
servdetch = 50; // Set timer to detach servos after 150msec - executed at end of main loop
digitalWrite(ledpins[0], HIGH); // Switch Red LED on
}

//*****
void loop()
{
    // The NmraDcc.process() method MUST be called frequently from this loop()
    // function for correct library operation and to process all DCC packets
    DCC.process();
    delay(3); // Sets normal execution time of loop() if no operations started

    // ===== Check for Address Programming Link fitted =====
    if (digitalRead(adr_prog) == LOW && prog_done == 0 && slctpress == 0 && operpress == 0) {
        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;
    if (flash == 100){
        digitalWrite(3, LOW); // Switch off Red LED
    }
    else{
        if (flash == 200){
            digitalWrite(3, HIGH); // Switch on Red LED
            flash = 0;
        }
    }
}
}

// ===== Fetch switch state to select Output Address for programming =====
if (digitalRead(sw_slct) == LOW && prog_actv == 1 && operpress == 0 && slctpress > 20) {
    delay(20);
    if (digitalRead(sw_slct) == LOW) {
        while (digitalRead(sw_slct) == LOW){

```

```

    //Select switch pressed
}    // Wait for Select switch to be released
slctpress = slctpress + 1; // Move to next setup state
// Setup states -
// 21 - Set Output Address 1, 22 - Output Address 2
// 23 - Set Output Address 3, 24 - Output Address 4
if (slctpress == 25){
    for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW);    // Switch all LEDs off
    }
    // Transfer entered Output Addresses (if any) to CV41-CV48
    for(i = 0; i < 4; i++){
        if (set_adr[i] != 0) {
            if ((DCC.getCV(41 + (i*2)) + (256 * DCC.getCV(42 + (i*2)))) != set_adr[i]) {
                DCC.setCV(41 + (i*2), byte(set_adr[i] % 256));
                delay(5);
                DCC.setCV(42 + (i*2), byte(set_adr[i] >> 8));
                delay(5);
                #ifdef DEBUG
                    Serial.print("Output Addr ");
                    Serial.print(i + 1, DEC);
                    Serial.print(" Programmed = ");
                    Serial.println(set_adr[i], DEC);
                #endif
            }
        }
    }
    for(i = 0; i < 4; i++){
        set_adr[i] = 0; // Clear any entered address data
    }
    prog_done = 1; // Prevent re-entry or setup operations until programming link removed
    slctpress = 0; // Programming complete
    #ifdef DEBUG
        Serial.println("Addr Programming Complete");
    #endif
}
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)
}
}
}

// ===== Fetch switch states to control Servo Left/Right Limits setup =====
if (digitalRead(sw_slct) == LOW && busy_dcc == 0 && prog_actv == 0 && operpress == 0 && slctpress < 18) {

```



```

delay(20);
if (digitalRead(sw_slct) == LOW) {
  digitalWrite(ledpins[0], HIGH);    // Setup started - Red LED on
  while (digitalRead(sw_slct) == LOW){
    delay(20);                      // Wait for Select switch to be released
  }
  slctpress = slctpress + 1; // Move to next setup state
  // Setup states -
  // 1 - Set Servo 1 Left, 2 - Set Servo 1 Right == Use Op to skip to next Servo (from state 1 to 5)
  // 3 - Set Servo 1 Rate
  // 4 - Store/Discard Servo 1 Data in CVs 55-59 - Red LED flashes
  // 5 - Set Servo 2 Left, 6 - Set Servo 2 Right == Use Op to skip to next Servo (from state 5 to 9)
  // 7 - Set Servo 2 Rate
  // 8 - Store/Discard Servo 2 Data in CVs 60-64 - Red LED flashes
  // 9 - Set Servo 3 Left, 10 - Set Servo 3 Right == Use Op to skip to next Servo (from state 9 to 13)
  // 11 - Set Servo 3 Rate
  // 12 - Store/Discard Servo 3 Data in CVs 65-69 - Red LED flashes
  // 13 - Set Servo 4 Left, 14 - Set Servo 4 Right == Use Op to skip to next Servo (from state 13 to 1)
  // 15 - Set Servo 4 Rate
  // 16 - Store/Discard Servo 4 Data in CVs 70-74 - Red LED flashes

  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 = (slctpress - 1) % 4;    // Determine current step in each Servo setup sequence
    if (slctstep == 0 || slctstep == 1){
      // Servo Left & Right Limit setup (slctpress = 1,2 or 5,6 or 9,10 or 13,14)
      for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW); // Ensure all LEDs off
      }
      for(i = 0; i < numsrvpins; i++){
        servo[i].detach();             // Remove drive from all servos
      }
      sindex = (slctpress - 1);
      sindex = sindex >> 2;             // Servo address (0 - 3)
      movedirn = slctpress % 2;        // 1 = Left, 0 = Right
      digitalWrite(ledpins[0], HIGH); // Switch Red LED on
    }
  }
}

```

```

servo[sindex].attach(srvpins[sindex]);    // Return drive to selected Servo
delay(50);
digitalWrite(ledpins[1 + sindex], HIGH); // Setup Servo(0-3) Limits - Green LED (1-4)
if (movedirn == 1) {
    servo[sindex].write(servoact[sindex].left_limit); // Position servo at current Left Limit
}
else {
    servo[sindex].write(servoact[sindex].right_limit); // Position servo at current Right Limit
}
delay(150);
servo[sindex].detach(); // Remove drive from servo
}
if (slctstep == 2){
    // Servo Rate setup (slctpress = 3, 7, 11 or 15)
    for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW);          // Ensure all LEDs off
    }
    sindex = (slctpress - 1);
    sindex = sindex >> 2;                        // Servo address (0 - 3)
    movedirn = slctpress % 2; // 1 = Left, 0 = Right
    digitalWrite(ledpins[0], HIGH);             // Switch Red LED on
    digitalWrite(ledpins[1 + sindex], HIGH);    // Setup Servo(0-3) Rates - Green LED (1-4)
    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 completed
        exec_servo_move ();
        delay(3);
    }
    delay(150);
    setup_servo_move (sindex, 0);               // Selected Servo to move right at set Slow Rate
    while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not completed
        exec_servo_move ();
        delay(3);
    }
    delay(150);
    busy_dcc = 0; // Allow configuration switch operations again
}
if (slctstep == 3){
    // Write Servo setup parameters to CVs (slctpress = 4, 8, 12 or 16)
    for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW);          // Ensure all LEDs off
    }
    digitalWrite(ledpins[0], HIGH); // Ready to write to CVs - Red LED flashing
    flash = 0;
    cvs_done = 0;                          // Prepare to save Servo parameters
    sindex = (slctpress - 1);
    sindex = sindex >> 2;                    // Servo address (0 - 3)
    digitalWrite(ledpins[0], HIGH);         // Switch Red LED on
    digitalWrite(ledpins[1 + sindex], HIGH); // Setup Servo(0-3) Write CVs - Green LED (1-4)
}

```

```

    }
  }
#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)
    for(i = 0; i < numledpins; i++){
      digitalWrite(ledpins[i], LOW); // Ensure all LEDs off
    }
    for(i = 0; i < numsrvpins; i++){
      servo[i].detach(); // Remove drive from all servos
    }
    sindex = (slctpress - 1);
    sindex = sindex >> 2; // Servo address (0 - 3)
    movedirn = 1; // 1 = Left
    digitalWrite(ledpins[0], HIGH); // Switch Red LED on
  }
}

```

```

servo[sindex].attach(srvpins[sindex]);    // Return drive to selected Servo
delay(150);
digitalWrite(ledpins[1 + sindex], HIGH); // Setup Servo(0-3) Limits - Green LED (1-4)
servo[sindex].write(servoact[sindex].left_limit); // Position servo at current Left Limit
delay(150);
servo[sindex].detach(); // Remove drive from servo
#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
}
while (digitalRead(sw_oper) == LOW){
    delay(20);           // Wait for Operate switch to be released
}
delay(300);
}
}

// ===== Fetch switch states to control Servo operation =====
if (digitalRead(sw_oper) == LOW && busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress < 5) {
    delay(20);
    if (digitalRead(sw_oper) == LOW) {
        for(i = 0; i < 5; i++){
            digitalWrite(ledpins[i], LOW); // Switch all LEDs off
        }
        operpress = operpress + 1; // Move to next operate state
        if (operpress == 5) {
            operpress = 0;           // Cease operations
        }
        if (operpress != 0){
            digitalWrite((ledpins[operpress]), HIGH); // Prep to operate Servo - Green LED on
        }
        #ifdef DEBUG
            Serial.print("Operate Press = ");
            Serial.println(operpress, DEC);
        #endif
        while (digitalRead(sw_oper) == LOW){
            delay(20);           // Wait for Operate switch to be released
        }
    }
}

// ===== Fetch switch states to centralise all Servos or perform Remote Reset =====
if (digitalRead(sw_left) == LOW && busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress == 0) {
    delay(20);

```

```

if (digitalRead(sw_left) == LOW) {
  //Left switch pressed
  while (digitalRead(sw_left) == LOW){
    if (digitalRead(sw_rite) == LOW) {
      delay(20);
      if (digitalRead(sw_rite) == LOW) {
        gotocentr = 1; // Right switch pressed also
      }
    }
  }
  if (digitalRead(sw_oper) == LOW) {
    delay(20);
    if (digitalRead(sw_oper) == LOW) {
      gotoreset = 1; // Operate switch pressed also
      if (digitalRead(sw_slct) == LOW) {
        delay(20);
        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
}
}

// ===== Fetch switch states to set Servo Left/Right Limits or Slow Rate =====
if (digitalRead(sw_left) == LOW && busy_dcc == 0 && prog_actv == 0 && ritepress == 0 && gotocentr != 1) {
  delay(20);
  if (digitalRead(sw_left) == LOW && digitalRead(sw_right) == HIGH) {
    leftpress = 1; // Left switch pressed (& not Right switch)
  }
}
if (digitalRead(sw_right) == LOW && leftpress == 0 && gotocentr != 1) {
  delay(20);
  if (digitalRead(sw_right) == LOW && digitalRead(sw_left) == HIGH) {
    ritepress = 1; // Right switch pressed (& not Left switch)
  }
}

// ===== Set all Servos to central position (90 degrees) =====
if (busy_dcc == 0 && prog_actv == 0 && slctpress == 0 && operpress == 0 && gotocentr == 1) {
  for(i = 0; i < 4; i++){
    servo[i].attach(srvpins[i]);
    servo[i].write(servcentr); // Initialise all servo positions
    delay(100);
    servo[i].write(servcentr);
    delay(100);
    servoact[i].curr_position = servcentr;
    servo[i].detach();
  }
  #ifdef DEBUG
    Serial.print("Servos 1-4 Position = ");
    Serial.println(servcentr, DEC);
  #endif
  leftpress = 0;
  ritepress = 0;
  gotocentr = 0;
  delay(200);
}

// ===== Set up Left and Right Limits for each Servo =====
if (busy_dcc == 0 && slctpress > 0 && slctpress < 17 && (slctstep == 0 || slctstep == 1)) {
  sindex = (slctpress - 1);
  sindex = sindex >> 2; // Servo address (0 - 3)
  movedirn = slctpress % 2; // 1 = Left, 0 = Right
  servo[sindex].attach(srvpins[sindex]);
  if (leftpress == 1) {
    if (movedirn == 1 && servoact[sindex].left_limit != 180) {
      servoact[sindex].left_limit = servoact[sindex].left_limit + 1; // Adjust servo left position up
    }
  }
}

```

```

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 != 180) {
  servoact[sindex].right_limit = servoact[sindex].right_limit + 1; // Adjust servo right position up
  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
}
}
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);
servo[sindex].detach();
}

// ===== 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)
      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);
      setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
      while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
        exec_servo_move ();
        delay(3);
      }
      delay(150);
      busy_dcc = 0; // Allow configuration switch operations again
#ifdef DEBUG
      Serial.print("Servo ");
      Serial.print(sindex + 1);
      Serial.print(" Slow Rate = ");
      Serial.println(servoact[sindex].slow_rate, DEC);
#endif
    }
  }
  if (ritepress == 1) {
    if (servoact[sindex].slow_rate > 1) {
      servoact[sindex].slow_rate = servoact[sindex].slow_rate - 1; //Adjust servo rate down (fast - Rapid)
      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);
      setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
      while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
        exec_servo_move ();
        delay(3);
      }
      delay(150);
      busy_dcc = 0; // Allow configuration switch operations again
#ifdef DEBUG
      Serial.print("Servo ");
      Serial.print(sindex + 1);
      Serial.print(" Slow Rate = ");
      Serial.println(servoact[sindex].slow_rate, DEC);

```



```

        #endif
    }
}
leftpress = 0;
ritepress = 0;
delay(150);
}

// ===== Prepare to Write Servo settings to CVs (EEPROM) =====
if (busy_dcc == 0 && slctpress > 3 && slctpress < 17 && slctstep == 3) {
    flash = flash + 1;
    if (flash == 100){
        digitalWrite(3, LOW); // Switch off Red LED
    }
    else{
        if (flash == 200){
            digitalWrite(3, HIGH); // Switch on Red LED
            flash = 0;
        }
    }
    sindex = (slctpress - 1);
    sindex = sindex >> 2; // Servo address (0 - 3)
    if (cvs_done == 0) {
        // Selected Servo identified by 'sindex'
        if (ritepress == 1) {
            // Save any changed parameters to relevant CV location in EEPROM
            if (servoact[sindex].slow_rate != byte (DCC.getCV(55+(sindex * 5)))) {
                DCC.setCV(55 + (sindex * 5), servoact[sindex].slow_rate); // Only save if value changed
                delay(5);
            }
            if (servoact[sindex].right_limit != byte (DCC.getCV(57+(sindex * 5)))) {
                DCC.setCV(57 + (sindex * 5), servoact[sindex].right_limit);
                delay(5);
            }
            if (servoact[sindex].left_limit != byte (DCC.getCV(58+(sindex * 5)))) {
                DCC.setCV(58 + (sindex * 5), servoact[sindex].left_limit);
                delay(5);
            }
            if (servoact[sindex].curr_position != byte (DCC.getCV(59+(sindex * 5)))) {
                DCC.setCV(59 + (sindex * 5), servoact[sindex].curr_position);
                delay(5);
            }
        }
        #ifdef DEBUG
            Serial.print("Servo ");
            Serial.print(sindex + 1, DEC);
            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);
            busy_dcc = 0; // Allow configuration switch operations again
            #ifdef DEBUG
                Serial.print("Operate Servo ");
                Serial.print(sindex + 1);
                Serial.print(" Left = ");
                Serial.println(servoact[sindex].left_limit, DEC);
            #endif
        }
        leftpress = 0;
    }
    if (ritepress == 1 && leftpress == 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].right_limit))

```

```

    || ((servoact[sindex].nrm_dirn == 0) && (servoact[sindex].curr_position != servoact[sindex].left_limit))) {
  setup_servo_move (sindex, 0); // Selected Servo to move right at set Slow Rate
  while (servoact[sindex].curr_position != servoact[sindex].end_value) { // Movement not yet completed
    exec_servo_move ();
    delay(3);
  }
  delay(150);
  busy_dcc = 0; // Allow configuration switch operations again
#ifdef DEBUG
  Serial.print("Operate Servo ");
  Serial.print(sindex + 1);
  Serial.print(" Right = ");
  Serial.println(servoact[sindex].right_limit, DEC);
#endif
}
ritepress = 0;
}
if (digitalRead(sw_slct) == LOW && slctpress == 0) {
  delay(20);
  if (digitalRead(sw_slct) == LOW) {
    nrm_rev = 1; // Note that Select switch has been pressed
  }
  while (digitalRead(sw_slct) == LOW){
    delay(20); // Wait for Select switch to be released
  }
  if (nrm_rev == 1) {
    if (servoact[sindex].nrm_dirn == 0) {
      servoact[sindex].nrm_dirn = 1; // Select pressed - so reverse Servo operate direction
    }
    else {
      servoact[sindex].nrm_dirn = 0;
    }
  }
  nrm_rev = 0;
  digitalWrite(ledpins[0], HIGH); // Switch Red LED on
  delay(600); // .. for 0.6 sec then
  digitalWrite(ledpins[0], LOW); // .. switch Red LED off again
  // Save changed Normal/Reverse Direction to relevant CV location in EEPROM
  if (servoact[sindex].nrm_dirn != byte (DCC.getCV(56+(sindex * 5)))) {
    DCC.setCV(56 + (sindex * 5), servoact[sindex].nrm_dirn);
    delay(5);
  }
}
#ifdef DEBUG
  Serial.print("Servo ");
  Serial.print(sindex + 1);
  if (servoact[sindex].nrm_dirn == 1) {
    Serial.println(" Set Normal");
  }
  else {

```

```

        Serial.println(" Set Reverse");
    }
    #endif
}
}
delay(150);
}

// ===== Operate Servos in response to received DCC Turnout Command =====
if (slctpress == 0 && operpress == 0) {
    // Call procedure to execute set up DCC Turnout Command(s) for all Servos
    exec_servo_move ();
    if (actv_indx == 0) {
        busy_dcc = 0; // Allow configuration switch operations again if no Servo moving
    }
} // End execute servo position commands

// ===== Complete Servo Initialisation after 150ms delay =====
if(servdetch != 0xFF) {
    servdetch--;
    if(servdetch == 0) {
        for ( i=0; i < numsrvpins; i++) {
            servo[i].detach(); // Removes drive from servo after position reached
            servoact[i].active = 0;
        }
        servdetch = 0xFF; // Mark servo initialisation complete
        digitalWrite(ledpins[0], LOW); // Switch Red LED off
        #ifdef DEBUG
            Serial.println("Servos - Positioned & Detached");
        #endif
    }
}

} // End main loop()

//*****

// This function is called whenever a normal DCC Turnout Packet is received in Output Addressing Mode
extern void notifyDccAccTurnoutOutput( uint16_t Addr, uint8_t Direction, uint8_t OutputPower ) {
    int Set_Addr;
    byte servo_sel;
    byte index;
    if (prog_actv == 1 && prog_done == 0) {
        // Address Programming is active -
        // Received Address is accepted as Output Address for the selected Servo
        index = slctpress - 21;
        if (Addr > 2043) {
            Addr = 0; // Only accept valid accessory addresses
        }
    }
}

```

```

}
set_adr[index] = Addr;
digitalWrite(ledpins[0], LOW); // Switch Red LED off
delay(600); // .. for 0.6 sec then
digitalWrite(ledpins[0], HIGH); // .. switch Red LED on again
#ifdef DEBUG
    Serial.print("Address = ");
    Serial.print(Addr, DEC);
    Serial.print(" Entered for Output ");
    Serial.println(index + 1, DEC);
#endif
slctpress = slctpress + 1; // Prepare for entry of next Address
// Setup states -
// 21 - Set Output Address 1, 22 - Output Address 2
// 23 - Set Output Address 3, 24 - Output Address 4
if (slctpress == 25){
    for(i = 0; i < numledpins; i++){
        digitalWrite(ledpins[i], LOW); // Switch all LEDs off
    }
    // Transfer entered Output Addresses (if any) to CV41-CV48
    for(i = 0; i < 4; i++){
        if (set_adr[i] != 0) {
            if ((DCC.getCV(41 + (i*2)) + (256 * DCC.getCV(42 + (i*2)))) != set_adr[i]) {
                DCC.setCV(41 + (i*2), byte(set_adr[i] % 256));
                delay(5);
                DCC.setCV(42 + (i*2), byte(set_adr[i] >> 8));
                delay(5);
                #ifdef DEBUG
                    Serial.print("Output Addr ");
                    Serial.print(i + 1, DEC);
                    Serial.print(" Programmed = ");
                    Serial.println(set_adr[i], DEC);
                #endif
            }
        }
    }
    for(i = 0; i < 4; i++){
        set_adr[i] = 0; // Clear any entered address data
    }
    prog_done = 1; // Prevent re-entry or setup operations until programming link removed
    slctpress = 0; // Programming complete
    flash = 0; // Prepare to blink Red LED
    #ifdef DEBUG
        Serial.println("Addr Programming Complete");
    #endif
}
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
  }
}
}

// This function is called whenever a DCC Turnout Command matches an Output Address
// to determine the direction to move the addressed Servo at the set rate
void setup_servo_move (int srv_num, int accsy_dirn) {
  byte pin_num;
  pin_num = srvpins[srv_num];
  if (servoact[srv_num].active == 0) {
    servoact[srv_num].active = 1; // Activate the addressed Servo
  }
}

```

```

servo[srv_num].attach(pin_num);
}
// Determine whether to move to the set right or left limit, depending on
// the commanded direction and whether the servo is set as reversed
if (accsy_dirn==1) {
  if (servoact[srv_num].nrm_dirn == 1) {
    servoact[srv_num].end_value = servoact[srv_num].left_limit;
  }
  else {
    servoact[srv_num].end_value = servoact[srv_num].right_limit;
  }
}
else {
  if (servoact[srv_num].nrm_dirn == 1) {
    servoact[srv_num].end_value = servoact[srv_num].right_limit;
  }
  else {
    servoact[srv_num].end_value = servoact[srv_num].left_limit;
  }
}
servoact[srv_num].loop_count = servoact[srv_num].slow_rate;
} // End setup_servo_move

// This function is called to execute a prepared DCC Turnout Command and
// move the selected Servo to the appropriate end point at the set rate
void exec_servo_move () {
  actv_mask = 1; // Set execute mask to bit 0 (Servo 1)
  for (int i=0; i < numsrvpins; i++) {
    if (servoact[i].active == 1) { // DCC command pending for this servo
      actv_indx = actv_indx | actv_mask; // Note that this servo is moving
      busy_dcc = 1; // Prevent configuration switch operations
      if (servoact[i].loop_count == 0) { // Act on pending DCC command
        if (servoact[i].curr_position < servoact[i].end_value) {
          // Move Left ie. increment Current Position
          servoact[i].curr_position = servoact[i].curr_position + 1;
        }
        else if (servoact[i].curr_position > servoact[i].end_value) {
          // Move Right ie. decrement Current Position
          servoact[i].curr_position = servoact[i].curr_position - 1;
        }
      }
      if (servoact[i].curr_position == servoact[i].end_value) { // DCC command completed
        actv_indx = actv_indx & (~actv_mask); // Note that this servo has stopped moving
        servoact[i].active = 0;
        servo[i].detach();
        if (servoact[i].curr_position != byte (DCC.getCV(59+(i * 5)))) {
          DCC.setCV(59 + (i * 5), servoact[i].curr_position); // Save new Current Position
          delay(5);
        }
      }
    }
  }
}

```

```

    }
    else {
        servo[i].write(servoact[i].curr_position);    // Move towards end position
        servoact[i].loop_count = servoact[i].slow_rate; // Prepare for next move
    }
}
else {
    servoact[i].loop_count = servoact[i].loop_count - 1; // Decrement Slow Rate count
}
} // End if this servo active
actv_mask = actv_mask << 1; // Shift execute mask to next Servo
} // End for all servos
} // End exec_servo_move

// This call-back function is called by the NmraDcc library when a DCC ACK needs to be sent in response
// to a Read/Verify or Program command when in Service Mode (connected to a programming track)
// Calling this function will flash all LEDs (ensure that the Keypad is connected to the Quad Servo Decoder)
// in order to increase the current drain on the power supply by approximately 68mA for 8 msec

void notifyCVAck(void) {
    for (int i=0; i < numledpins; i++) {
        digitalWrite(ledpins[i], HIGH); // Switch all LEDs on
    }
    delay(8); // .. wait for 8 msec
    for (int i=0; i < numledpins; i++) {
        digitalWrite(ledpins[i], LOW); // .. then switch all LEDs off
    }
}

```