```
#include <avr/pgmspace.h>
#include <EEPROM.h>

// *** THIS IS NOW DEVELOPED ON ARDUINO 1.0 ***

// Arduino Yaesu GS-232A/B Rotator Interface Unit Emulator
// Anthony Good
// K3NG
// anthony.good@gmail.com

// Modified by John Eigenbrode
// W3SA
// w3sa@arrl.net
// Contributions: AZ/EL testing and debugging, AZ/EL LCD Enhancements, original
North center code, Az/El Rotator Control Connector Pins

    /*

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

    */

#define CODE_VERSION 2012050501

// All copyrights are the property of their respective owners

//  ASCII Art Schematic
//
//                             +----------------Yaesu Pin 1
//                             |
//                             N
//         D11---{100 ohms}---P      2N2222
//                             N    or similar
//                             |
//                            GND
//
//                             +----------------Yaesu Pin 2
//                             |
//                             N
//         D12---{100 ohms}---P      2N2222
//                             N    or similar
//                             |
//                            GND
//
//         A0-------------+---------------------Yaesu Pin 4aq
//                        |
//                     [0.01uF]
//                        |
//                       GND
//
//         D10---{4.7K}---+---------------------Yaesu Pin 3
//                        |
//                      [10uF]
```

```
//                        |
//                       GND
//
//
//         Not Connected-----------------------Yaesu Pin 6
//
//         GND---------------------------------Yaesu Pin 5

// Alternatively Yaesu Pin 3 can be connected to Pin 6 if variable speed
functionality (X commands) are not desired.  This will feed +5V directly
// into the speed voltage pin and set the unit for maximum speed all the time

// Yaesu Azimuth Only Rotator Controller Connector Pins
//
//    6 || 5
//   4      3
//     2  1
// 1 - ground to rotate L
// 2 - ground to rotate R
// 3 - speed voltage (input); 4.5V = max speed
// 4 - analog azimuth voltage (output); 0V = full CCW, ~4.9V = full CW
// 5 - ground
// 6 - +5V or so

// Yaesu Az/El Rotator Control Connector Pins
//
//    7  | | 6
//   3    8   1
//    5      4
//        2
//
// 1 - 2 - 4.5 VDC corresponding to 0 to 180 degrees elevation
// 2 - Connect to Pin 8 to rotate right (clockwise)
// 3 - Connect to Pin 8 to rotate Down
// 4 - Connect to Pin 8 to rotate left (counterclockwise)
// 5 - Connect to Pin 8 to rotate Up
// 6 - 2 - 4.5 VDC corresponding to 0 to 450 degrees rotation
// 7 - 13 - 6 VDC at up to 200 mA
// 8 - Common ground

//  ASCII Art Schematic
//
//                             +-----------------Yaesu Pin 4
//                             |
//                             N
//         D6---{100 ohms}---P      2N2222
//                             N    or similar
//                             |
//                            GND
//
//                             +-----------------Yaesu Pin 2
//                             |
//                             N
//         D7---{100 ohms}---P      2N2222
//                             N    or similar
//                             |
//                            GND
//                             +-----------------Yaesu Pin 5
//                             |
//                             N
//         D8---{100 ohms}---P      2N2222
//                             N    or similar
//                             |
//                            GND
```

```
//
//                            +----------------Yaesu Pin 3
//                            |
//                            N
//         D9---{100 ohms}---P      2N2222
//                            N    or similar
//                            |
//                           GND
//
//         A0-----------------------------------Yaesu Pin 6
//
//         A1-----------------------------------Yaesu Pin 1
//
//         NC-----------------------------------Yaesu Pin 7
//
//         GND----------------------------------Yaesu Pin 8

// Quick Start
//
// In order to test and calibrate your unit, connect the Serial Monitor to the
COM port set for 9600 and carriage return
// All command letters must be uppercase.
// The backslash (\) command toggles debug mode which will periodically display
key parameters.
//
// To test basic operation, send commands using Serial Monitor:
// Rotate left(CCW): L
// Rotate right(CW): R
// Stop rotation: A or S commands
// Read the current azimuth: C
// Go to an azimuth automatically: M command (examples: M180 = go to 180
degrees, M010 = go to 10 degrees
//
// To calibrate the unit, send the O command and rotate to 180 degrees / full
CCW and send a carriage return, then
// send the F command and rotate to 270 degrees / full CW and send a carriage
return (assuming a 450 degrees rotation rotator).
// If you are operating a 360 degree rotation rotator, for the F command rotate
to 180 degrees full CW, not 270.
//
// ( CW means clockwise (or LEFT on Yaesu rotators) and CCW means counter
clockwise (or RIGHT on Yaesy rotators))
//
// To use this code with AZ/EL rotators, uncomment the FEATURE_ELEVATION_CONTROL
line below
//
// It does properly handle the 450 degree rotation capability of the Yaesu
rotators.
//
// This code has been successfully interfaced with non-Yaesu rotators. Email me
if you have a rotator you would like to interface this to.
//
// With the addition of a reasonable capacity DC power supply and two relays,
this unit could entirely replace a control unit if desired.

// 9/12/11 W3SA JJE added code to correct elevation display which was not
following A1 input (map function was not working using the variables)
// Added code to keep azimuth and elevation updated if changed from the rotor
control unit.
// Added code to handle flipped azimuth of antenna when elevation passes 90
degrees.
// Changed LCD display to display Direction, Azimuth and Elevation of antenna(s)
on first line of display and actual Rotor azimuth and elevation on the second
line
```

```
// Then when the elevation has passed 90 degrees you would get:
//              NNE A 15 E 75
//          RTR A 195 E 115
// Otherwise it would be
//          NNE A 15 E 75
//          RTR A 15 E 75
//

//#define FEATURE_ELEVATION_CONTROL            // uncomment this for AZ/EL
rotators
#define FEATURE_GS_232B_EMULATION           // uncomment this for GS-232B
emulation (instead of GS-232A)
//#define OPTION_C_COMMAND_SENDS_AZ_AND_EL  // uncomment this when using with
AZ/EL rotators with Ham Radio Deluxe
#define OPTION_DELAY_C_CMD_OUTPUT           // uncomment this when using with
Ham Radio Deluxe
#define OPTION_SLOW_DOWN_BEFORE_TARGET_AZ   // this option slows down azimuthal
rotation shortly before reaching target azimuth
#define OPTION_AZ_SLOW_START_UP             // start rotation at a slower speed

// rotation settings
#define AZIMUTH_STARTING_POINT_DEFAULT 180      // the starting point in degrees
of the azimuthal rotator
                                                // (the GS-232B Emulation Z
command will override this and write the setting to eeprom)
#define AZIMUTH_ROTATION_CAPABILITY_DEFAULT 450 // the default rotation
capability of the rotator in degrees
                                                // (the P36 and P45 commands
will override this and write the setting to eeprom)
#define ELEVATION_MAXIMUM_DEGREES 180           // change this to set the
maximum elevation in degrees


// Ham Radio Deluxe expects AZ and EL in output for C command in AZ/EL mode.
I'm not sure if this is default behavior for
// the Yaesu interface since the C2 command is supposed to be for AZ and EL.  If
you have problems with other software with this code in AZ/EL mode,
// uncomment #define OPTION_C_COMMAND_SENDS_AZ_AND_EL.

#define DEBUG_MEMORY
#define INITIALIZE_EEPROM_AZ_SETTINGS

#define FEATURE_LCD_DISPLAY             // Comment this line out to compile
without LCD display support
#include <LiquidCrystal.h>              // Comment this line out to compile
without LCD display support
LiquidCrystal lcd(12, 11, 5, 4, 3, 2);   // Comment this line out to compile
without LCD display support

// azimuth pins                         // use just the azimuth pins for an
azimuth-only rotator
#define rotate_cw 6                     // goes high to activate rotator R (CW)
rotation - pin 1 on Yaesu connector
#define rotate_ccw 7                    // goes high to activate rotator L
(CCW) rotation - pin 2 on Yaesu connector
#define button_cw A2                    // normally open button to ground for
manual CW rotation
#define button_ccw A3                   // normally open button to ground for
manual CCW rotation
#define serial_led 0                     // LED blinks when command is received
on serial port (set to 0 to disable)
#define rotator_analog_az A0            // reads analog azimuth voltage from
rotator - pin 4 on Yaesu connector
#define azimuth_speed_voltage 10        // PWM output for speed control voltage
```

```
feed into rotator (optional)
#define overlap_led 13                    // line goes high when azimuth rotator
is in overlap (> 360 rotators)
#define brake_az 0                        // goes high to disengage azimuth
brake (set to 0 to disable)


// alternate pinouts for K3NG's personal setup
//#define rotate_cw A2
//#define rotate_ccw A1
//#define button_cw 0
//#define button_ccw 0
//#define button_up 0
//#define button_down 0

// elevation pins
#ifdef FEATURE_ELEVATION_CONTROL
#define rotate_up 8                      // goes high to activate rotator
elevation up
#define rotate_down 9                    // goes high to activate rotator
elevation down
#define rotator_analog_el A1             // reads analog elevation voltage from
rotator
#define button_up A4                     // normally open button to ground for
manual up elevation
#define button_down A5                   // normally open button to ground for
manual down rotation
#define brake_el 0                        // goes high to disengage elevation
brake (set to 0 to disable)
#endif


// macros
#define AZ 1
#define EL 2

#define DISENGAGE 0
#define ENGAGE 1

#define IDLE 0
#define ROTATING_CW 1
#define ROTATING_CCW 2
#define ROTATING_CW_AUTO 3
#define ROTATING_CCW_AUTO 4

#define ROTATING_UP 1
#define ROTATING_DOWN 2
#define ROTATING_UP_AUTO 3
#define ROTATING_DOWN_AUTO 4

#define DEACTIVATE 0
#define ACTIVATE 1

#define ROTATION_CW 1
#define ROTATION_CCW 2
#define ROTATION_UP 3
#define ROTATION_DOWN 4

#define STOP 0
#define LEFT 1
#define RIGHT 2

#define UP 1
#define DOWN 2
```

```
#define EMPTY 0
#define LOADED_AZIMUTHS 1
#define RUNNING_AZIMUTHS 2
#ifdef FEATURE_ELEVATION_CONTROL
#define LOADED_AZIMUTHS_ELEVATIONS 3

#define RUNNING_AZIMUTHS_ELEVATIONS 4
#endif

// analog voltage calibration - these are default values; you can either tweak
these or set via the O and F commands (and O2 and F2)....
#define ANALOG_AZ_FULL_CCW 4
#define ANALOG_AZ_FULL_CW 1009
#define ANALOG_EL_0_DEGREES 2
#define ANALOG_EL_MAX_ELEVATION 1018  // maximum elevation is normally 180
degrees unless change below for ELEVATION_MAXIMUM_DEGREES

#define ANALOG_AZ_OVERLAP_DEGREES  180        // if overlap_led above is
enabled, turn on overlap led line if azimuth is greater than this setting

// PWM speed voltage settings
#define PWM_SPEED_VOLTAGE_X1  64
#define PWM_SPEED_VOLTAGE_X2  128
#define PWM_SPEED_VOLTAGE_X3  191
#define PWM_SPEED_VOLTAGE_X4  255

#define SLOW_DOWN_BEFORE_TARGET_AZ 10  // if OPTION_SLOW_DOWN_BEFORE_TARGET_AZ
is enabled, slowdown will be activated within this many degrees of target
azimuth
#define SLOW_START_UP_TIME 2000        // if OPTION_AZ_SLOW_START_UP is enabled,
the unit will start rotation at a lower speed for this many milliseconds
#define PWM_SPEED_VOLTAGE_SLOW 64      // PWM value for slow start up and slow
down

// various code settings
#define AZIMUTH_TOLERANCE 1            // rotator will stop within X degrees
when doing autorotation
#define ELEVATION_TOLERANCE 1
#define OPERATION_TIMEOUT 60000        // timeout for any rotation operation in
mS ; 60 seconds is usually enough unless you have the speed turned down
#define TIMED_INTERVAL_ARRAY_SIZE 20
#define SERIAL_BAUD_RATE 9600          // 9600
#define LCD_UPDATE_TIME 2000           // LCD update time in milliseconds
#define AZ_BRAKE_DELAY 3000            // in milliseconds
#define EL_BRAKE_DELAY 3000            // in milliseconds



//eeprom memory locations
#define EEPROM_ID_BYTE 1
#define EEPROM_ANALOG_AZ_FULL_CCW 2
#define EEPROM_ANALOG_AZ_FULL_CW 4
#define EEPROM_ANALOG_EL_0_DEGREES 6
#define EEPROM_ANALOG_EL_MAX_ELEVATION 8
#define EEPROM_AZIMUTH_STARTING_POINT 10
#define EEPROM_AZIMUTH_ROTATION_CAPABILITY 12

// variables
int azimuth = 0;
int display_azimuth = 0;   /// Variable added to handle flipped azimuth ///W3SA
int raw_azimuth = 0;
byte incoming_serial_byte = 0;
byte command_buffer[50];
```

```
int command_buffer_index = 0;
byte az_rotation_status = IDLE;
byte debug_mode = 0;
int analog_az = 0;
unsigned long last_debug_output_time = 0;
int target_azimuth = 0;
unsigned long az_last_rotate_initiation = 0;
byte azimuth_button_was_pushed = 0;
int azimuth_starting_point = AZIMUTH_STARTING_POINT_DEFAULT;
int azimuth_rotation_capability = AZIMUTH_ROTATION_CAPABILITY_DEFAULT;
byte brake_az_engaged = 0;
byte brake_el_engaged = 0;


int analog_az_full_ccw = ANALOG_AZ_FULL_CCW;
int analog_az_full_cw = ANALOG_AZ_FULL_CW;
int analog_el_0_degrees = ANALOG_EL_0_DEGREES;
int analog_el_max_elevation = ANALOG_EL_MAX_ELEVATION;

int timed_buffer_azimuths[TIMED_INTERVAL_ARRAY_SIZE];
int timed_buffer_number_entries_loaded = 0;
int timed_buffer_entry_pointer = 0;
int timed_buffer_interval_value_seconds = 0;
unsigned long last_timed_buffer_action_time = 0;
byte timed_buffer_status = 0;
byte current_azimuth_speed_voltage = 0;

#ifdef FEATURE_ELEVATION_CONTROL
int elevation = 0;
int display_elevation = 0;   // Variable added to handle elevation beyond 90
degrees.  /// W3SA
int el_rotation_status = IDLE;
int analog_el = 0;
int target_elevation = 0;
unsigned long el_last_rotate_initiation = 0;
int timed_buffer_elevations[TIMED_INTERVAL_ARRAY_SIZE];
byte elevation_button_was_pushed = 0;
#endif

#ifdef FEATURE_LCD_DISPLAY
unsigned long last_lcd_update = 0;
byte display_status_dirty = 0;
#define LCD_COLUMNS 16
//#define LCD_COLUMNS 20
#endif

prog_uchar running_azimuths_string[] __attribute__((section(".progmem.data"))) =
{"initiate_timed_buffer: changing state to RUNNING_AZIMUTHS\n"};
prog_uchar running_azimuths_elevations_string[]
__attribute__((section(".progmem.data"))) = {"initiate_timed_buffer: changing
state to RUNNING_AZIMUTHS_ELEVATIONS\n"};
prog_uchar az_abort_rotation_string[] __attribute__((section(".progmem.data")))
= {"az_check_operation_timeout: timeout reached, aborting rotation\r\n"};
prog_uchar req_azimuth_with_tol_string[]
__attribute__((section(".progmem.data"))) = {"initial_auto_rotation: requested
azimuth within tolerance\r\n"};
prog_uchar exec_next_time_az_string[] __attribute__((section(".progmem.data")))
= {"check_timed_interval: executing next timed interval step - azimuths\n"};
prog_uchar complete_time_buff_string[] __attribute__((section(".progmem.data")))
= {"check_timed_interval: completed timed buffer; changing state to EMPTY\n"};
prog_uchar exec_next_time_az_el_string[]
__attribute__((section(".progmem.data"))) = {"check_timed_interval: executing
next timed interval step - az and el\n"};
prog_uchar target_az_reach_stop_string[]
```

```
__attribute__((section(".progmem.data"))) = {"az_check_rotation: target azimuth
reached, stopping azimuth rotation\r\n"};
prog_uchar target_el_reached_string[] __attribute__((section(".progmem.data")))
= {"el_check_rotation: target elevation reached, stopping elevation rotation;
elevation: "};
prog_uchar el_w_cmd_erro_string[] __attribute__((section(".progmem.data"))) =
{"el_initial_auto_rotation: W command elevation error\r\n"};
prog_uchar az_offset_set_string[] __attribute__((section(".progmem.data"))) =
{"Rotate to full CCW and send keystroke...\r\n"};
prog_uchar az_fullscale_set_string[] __attribute__((section(".progmem.data"))) =
{"Rotate to full CW and send keystroke...\r\n"};
prog_uchar el_offset_set_string[] __attribute__((section(".progmem.data"))) =
{"Elevate to 0 degrees and send keystroke...\r\n"};
prog_uchar el_fullscale_set_string[] __attribute__((section(".progmem.data"))) =
{"Elevate to 180 (or maximum elevation) and send keystroke...\r\n"};
prog_uchar wrote_to_memory_string[] __attribute__((section(".progmem.data"))) =
{"Wrote to memory.\r\n"};

// debug dump text
prog_uchar Analog_Azimuth_string[] __attribute__((section(".progmem.data"))) =
{"Analog Azimuth: "};
prog_uchar Azimuth_string[] __attribute__((section(".progmem.data"))) =
{"Azimuth: "};
prog_uchar Raw_Azimuth_string[] __attribute__((section(".progmem.data"))) =
{"Raw Azimuth: "};
prog_uchar Azimuth_Starting_Point_string[]
__attribute__((section(".progmem.data"))) = {"Azimuth Starting Point: "};
prog_uchar azimuth_rotation_capability_string[]
__attribute__((section(".progmem.data"))) = {"Azimuth Rotation Capability: "};
prog_uchar Timed_interval_buffer_status_string[]
__attribute__((section(".progmem.data"))) = {"Timed interval buffer status: "};
prog_uchar Timed_interval_buffer_time_interval_seconds_string[]
__attribute__((section(".progmem.data"))) = {"Timed interval buffer time
interval (seconds): "};


prog_uchar serial_help_string[] __attribute__((section(".progmem.data"))) = {
  "R Rotate Azimuth Clockwise\r\n"
  "L Rotate Azimuth Counter Clockwise\r\n"
  "A Stop\r\n"
  "C Report Azimuth in Degrees\r\n"
  "M### Rotate to ### degrees \r\n"
  "MTTT XXX XXX XXX ... Timed Interval Direction Setting  (TTT = Step value in
seconds, XXX = Azimuth in degrees)\r\n"
  "T Start Timed Interval Tracking\r\n"
  "N Report Total Number of M Timed Interval Azimuths\r\n"
  "X1 Horizontal Rotation Low Speed\r\n"
  "X2 Horizontal Rotation Middle 1 Speed\r\n"
  "X3 Horizontal Rotation Middle 2 Speed\r\n"
  "X4 Horizontal Rotation High Speed\r\n"
  "S Stop\r\n"
  "O Offset Calibration\r\n"
  "F Full Scale Calibration\r\n"
  #ifdef FEATURE_ELEVATION_CONTROL
  "U Rotate Elevation Up\r\n"
  "D Rotate Elevation Down\r\n"
  "E Stop Elevation Rotation\r\n"
  "B Report Elevation in Degrees\r\n"
  "Wxxx yyy Rotate Azimuth to xxx Degrees and Elevation to yyy Degrees\r\r"
  "O2 Elevation Offset Calibration (0 degrees)\r\n"
  "F2 Elevation Full Scale Calibration (180 degrees (or maximum))\r\n"
  #endif
  };
```

```
void setup() {

  Serial.begin(SERIAL_BAUD_RATE);

  #ifdef INITIALIZE_EEPROM_AZ_SETTINGS
  EEPROM.write(EEPROM_AZIMUTH_STARTING_POINT,highByte(azimuth_starting_point));
  EEPROM.write(EEPROM_AZIMUTH_STARTING_POINT+1,lowByte(azimuth_starting_point));

EEPROM.write(EEPROM_AZIMUTH_ROTATION_CAPABILITY,highByte(azimuth_rotation_capabi
lity));

EEPROM.write(EEPROM_AZIMUTH_ROTATION_CAPABILITY+1,lowByte(azimuth_rotation_capab
ility));
  #endif //INITIALIZE_EEPROM_AZ_SETTINGS

  read_settings_from_eeprom();

  if (serial_led) {
    pinMode(serial_led, OUTPUT);
  }

  if (overlap_led) {
    pinMode(overlap_led, OUTPUT);
  }

  if (brake_az) {
    pinMode(brake_az, OUTPUT);
    digitalWrite(brake_az, LOW);
  }

  #ifdef FEATURE_ELEVATION_CONTROL
  if (brake_el) {
    pinMode(brake_el, OUTPUT);
    digitalWrite(brake_el, LOW);
  }
  #endif //FEATURE_ELEVATION_CONTROL

  pinMode(rotate_cw, OUTPUT);
  pinMode(rotate_ccw, OUTPUT);
  rotator(DEACTIVATE,ROTATION_CW);
  rotator(DEACTIVATE,ROTATION_CCW);
  pinMode(rotator_analog_az, INPUT);
  if (button_cw) {
    pinMode(button_cw, INPUT);
    digitalWrite(button_cw, HIGH);
  }
  if (button_ccw) {
    pinMode(button_ccw, INPUT);
    digitalWrite(button_ccw, HIGH);
  }

  if (azimuth_speed_voltage) {                  // if azimuth_speed_voltage pin
is configured, set it up for PWM output
    analogWrite(azimuth_speed_voltage, PWM_SPEED_VOLTAGE_X4);
    current_azimuth_speed_voltage = PWM_SPEED_VOLTAGE_X4;
  }

  #ifdef FEATURE_ELEVATION_CONTROL
  pinMode(rotate_up, OUTPUT);
  pinMode(rotate_down, OUTPUT);
  rotator(DEACTIVATE,ROTATION_UP);
  rotator(DEACTIVATE,ROTATION_DOWN);
  pinMode(rotator_analog_el, INPUT);
  if (button_up) {
```

```
    pinMode(button_up, INPUT);
    digitalWrite(button_up, HIGH);
  }
  if (button_down) {
    pinMode(button_down, INPUT);
    digitalWrite(button_down, HIGH);
  }
  read_elevation();
  #endif

  read_azimuth();
  report_current_azimuth();      // report the azimuth right off the bat without
a C command; the Arduino doesn't wake up quick enough
                                 // to get first C command from HRD and if HRD
doesn't see anything it doesn't connect

  timed_buffer_status = EMPTY;

  #ifdef FEATURE_LCD_DISPLAY
  lcd.begin(LCD_COLUMNS, 2);
  lcd.setCursor(((LCD_COLUMNS-4)/2),0);
  lcd.print("K3NG");
  if (LCD_COLUMNS < 20) {
    lcd.setCursor(((LCD_COLUMNS-15)/2),1);  // W3SA
  } else {
    lcd.setCursor(((LCD_COLUMNS-18)/2),1);
  }
  lcd.print("Rotor Controller");
  last_lcd_update = millis();
  #endif

}

void loop() {

  check_serial();
  read_azimuth();
  az_check_rotation();
  az_check_operation_timeout();
  check_timed_interval();
  check_buttons();
  check_overlap();
  check_brake_release();

  #ifdef FEATURE_ELEVATION_CONTROL
  read_elevation();
  el_check_rotation();
  el_check_operation_timeout();
  #endif

  #ifdef FEATURE_LCD_DISPLAY
  update_display();
  #endif

  #ifdef OPTION_SLOW_DOWN_BEFORE_TARGET_AZ
  check_rotation_slow_down();
  #endif

  #ifdef OPTION_AZ_SLOW_START_UP
  check_rotation_slow_startup();
  #endif

  if (debug_mode) {
    output_debug();
```

```
  }
}

//--------------------------------------------------------------
void check_brake_release() {


 static byte in_az_brake_release_delay = 0;
 static unsigned long az_brake_delay_start_time = 0;
 #ifdef FEATURE_ELEVATION_CONTROL
 static byte in_el_brake_release_delay = 0;
 static unsigned long el_brake_delay_start_time = 0;
 #endif //FEATURE_ELEVATION_CONTROL

 if ((az_rotation_status == IDLE) && (brake_az_engaged)) {
   if (in_az_brake_release_delay) {
     if ((millis() - az_brake_delay_start_time) > AZ_BRAKE_DELAY) {
       brake_release(AZ, DISENGAGE);
       in_az_brake_release_delay = 0;
     }
   } else {
     az_brake_delay_start_time = millis();
     in_az_brake_release_delay = 1;
   }
 }

 #ifdef FEATURE_ELEVATION_CONTROL
 if ((el_rotation_status == IDLE) && (brake_el_engaged)) {
   if (in_el_brake_release_delay) {
     if ((millis() - el_brake_delay_start_time) > EL_BRAKE_DELAY) {
       brake_release(EL, DISENGAGE);
       in_el_brake_release_delay = 0;
     }
   } else {
     el_brake_delay_start_time = millis();
     in_el_brake_release_delay = 1;
   }
 }
 #endif //FEATURE_ELEVATION_CONTROL

}

//--------------------------------------------------------------
void brake_release(byte az_or_el, byte operation){

 if (az_or_el == AZ) {
   if (brake_az) {
     if (operation == ENGAGE) {
       digitalWrite(brake_az,HIGH);
       brake_az_engaged = 1;
       if (debug_mode) {
         Serial.println(F("brake_release: brake_az ENGAGE"));
       }
     } else {
       digitalWrite(brake_az,LOW);
       brake_az_engaged = 0;
       if (debug_mode) {
         Serial.println(F("brake_release: brake_az DISENGAGE"));
       }

     }
   }
 } else {
   #ifdef FEATURE_ELEVATION_CONTROL
```

```
   if (brake_el) {
       digitalWrite(brake_el,HIGH);
       brake_el_engaged = 1;
       if (debug_mode) {
         Serial.println(F("brake_release: brake_el ENGAGE"));
       }
     } else {
       digitalWrite(brake_el,LOW);
       brake_el_engaged = 0;
       if (debug_mode) {
         Serial.println(F("brake_release: brake_el DISENGAGE"));
       }
   }
   #endif //FEATURE_ELEVATION_CONTROL
 }

}


//-------------------------------------------------------------

#ifdef OPTION_AZ_SLOW_START_UP
void check_rotation_slow_startup()
{
   static byte in_slowstart = 0;
   static byte slowstart_completed = 0;
   static unsigned long slowstart_start_time = 0;

   if (az_rotation_status != IDLE){
     if ((!in_slowstart) && (!slowstart_completed) && ((az_rotation_status ==
ROTATING_CW_AUTO) || (az_rotation_status == ROTATING_CCW_AUTO)) ) {
       if (debug_mode) {
        Serial.print(F("check_rotation_slow_startup: activating slow
start\r\n"));
       }
       analogWrite(azimuth_speed_voltage, PWM_SPEED_VOLTAGE_SLOW);
       in_slowstart = 1;
       slowstart_start_time = millis();
     }

     if ((in_slowstart) && ((millis() - slowstart_start_time) >
SLOW_START_UP_TIME)) {
       if (debug_mode) {
         Serial.print(F("check_rotation_slow_startup: deactivating slow
start\r\n"));
       }
       analogWrite(azimuth_speed_voltage, current_azimuth_speed_voltage);
       in_slowstart = 0;
       slowstart_completed = 1;
     }
   } else {
     slowstart_completed = 0;  // we're back in idle mode
   }


}
#endif

//-------------------------------------------------------------
#ifdef OPTION_SLOW_DOWN_BEFORE_TARGET_AZ
void check_rotation_slow_down()
{
   static byte in_slowdown = 0;
```

```
   if (az_rotation_status != IDLE){
     if ((!in_slowdown) && (abs(target_azimuth - azimuth) <
SLOW_DOWN_BEFORE_TARGET_AZ) && ((az_rotation_status == ROTATING_CW_AUTO) ||
(az_rotation_status == ROTATING_CCW_AUTO)) ) {
       if (debug_mode) {
        Serial.print(F("check_rotation_slow_down: activating slowdown\r\n"));
       }
       analogWrite(azimuth_speed_voltage, PWM_SPEED_VOLTAGE_SLOW);
       in_slowdown = 1;
     }

   } else {
     if (in_slowdown) {
       if (debug_mode) {
         Serial.print(F("check_rotation_slow_down: deactivating slowdown\r\n"));
       }
       analogWrite(azimuth_speed_voltage, current_azimuth_speed_voltage);
       in_slowdown = 0;
     }

   }


}
#endif
//--------------------------------------------------------------------
void check_overlap(){

  static byte overlap_led_status = 0;
  static unsigned long last_check_time;

  if ((overlap_led) && ((millis() - last_check_time) > 500)) {
     if ((analog_az > 500) && (azimuth > ANALOG_AZ_OVERLAP_DEGREES) && (!
overlap_led_status)) {
       digitalWrite(overlap_led, HIGH);
       overlap_led_status = 1;
       if (debug_mode) {
         Serial.print(F("check_overlap: in overlap\r\n"));
       }
     } else {
       if (((analog_az < 500) || (azimuth < ANALOG_AZ_OVERLAP_DEGREES)) &&
(overlap_led_status)) {
         digitalWrite(overlap_led, LOW);
         overlap_led_status = 0;
         if (debug_mode) {
           Serial.print(F("check_overlap: overlap off\r\n")
           );
         }
       }
     }
     last_check_time = millis();

  }

}
//--------------------------------------------------------------------
void check_serial(){

  if (Serial.available() > 0) {
    if (serial_led) {
      digitalWrite(serial_led, HIGH);                    // blink the LED just
to say we got something
    }
    incoming_serial_byte = Serial.read();
```

```
    if (incoming_serial_byte != 13) {               // if it's not a carriage
return, add it to the buffer
      command_buffer[command_buffer_index] = incoming_serial_byte;
      command_buffer_index++;
    } else {                        // we got a carriage return, time to get to
work on the command
      if ((command_buffer[0] > 96) && (command_buffer[0] < 123)) {
        command_buffer[0] = command_buffer[0] - 32;
      }
      switch (command_buffer[0]) {                  // look at the first
character of the command
        case 67:                                    // C - return current azimuth
          if (debug_mode) {Serial.println(F("check_serial: C cmd"));}
          #ifdef OPTION_DELAY_C_CMD_OUTPUT
          delay(400);
          #endif
          report_current_azimuth();
          break;
        case 70: if (debug_mode) {Serial.println(F("check_serial: F cmd"));}
f_command(); break;                         // F - full scale calibration
        case 92: if (debug_mode) {debug_mode = 0;} else {debug_mode = 1;} break;
// backslash toggles debug mode (not a standard Yaesu GS-232A command
        case 72: print_help(); break;                       // H - print help
(simulated Yaesu GS-232A help - not all commands are supported
        case 76: if (debug_mode) {Serial.println(F("check_serial: L cmd"));}
az_manual_rotation(LEFT); break;            // L - manual left (CCW) rotation
        case 79: if (debug_mode) {Serial.println(F("check_serial: O cmd"));}
o_command(); break;                         // O - offset calibration
        case 82: if (debug_mode) {Serial.println(F("check_serial: R cmd"));}
az_manual_rotation(RIGHT); break;           // R - manual right (CW) rotation
        case 65: if (debug_mode) {Serial.println(F("check_serial: A cmd"));}
az_manual_rotation(STOP); break;            // A - CW/CCW rotation stop
        case 83:          // S - all stop
          if (debug_mode) {Serial.println(F("check_serial: S cmd"));}
          az_manual_rotation(STOP);
          #ifdef FEATURE_ELEVATION_CONTROL
          el_manual_rotation(STOP);
          #endif
          clear_timed_buffer();
          break;
        case 77: if (debug_mode) {Serial.println(F("check_serial: M cmd"));}
az_initiate_auto_rotation(-1); break;       // M - auto azimuth rotation
        case 78: if (debug_mode) {Serial.println(F("check_serial: N cmd"));}
Serial.print(timed_buffer_number_entries_loaded); Serial.write("\r\n"); break;
// N - number of loaded timed interval entries
        case 84: initiate_timed_buffer(); break;            // T - initiate timed
tracking
        case 88: if (debug_mode) {Serial.println(F("check_serial: X cmd"));}
x_command(); break;                         // X - azimuth speed change
        #ifdef FEATURE_ELEVATION_CONTROL
        case 85: if (debug_mode) {Serial.println(F("check_serial: U cmd"));}
el_manual_rotation(UP); break;              // U - manual up rotation
        case 68: if (debug_mode) {Serial.println(F("check_serial: D cmd"));}
el_manual_rotation(DOWN); break;            // D - manual down rotation
        case 69: if (debug_mode) {Serial.println(F("check_serial: E cmd"));}
el_manual_rotation(STOP); break;            // E - stop elevation rotation
        case 66: report_current_elevation(); break;         // B - return current
elevation
        case 87: if (debug_mode) {Serial.println(F("check_serial: W cmd"));}
az_el_initiate_auto_rotation(); break;      // W - auto elevation rotation
        #endif
        #ifdef FEATURE_GS_232B_EMULATION
        case 80: p_command(); break;                        // P - switch between
360 and 450 degree mode
```

```
        case 90:                                                // Z - Starting point
toggle
          if (azimuth_starting_point == 180) {
            azimuth_starting_point = 0;
            Serial.print(F("N Center\r\n"));
          } else {
            azimuth_starting_point = 180;
            Serial.print(F("S Center\r\n"));
          }
          write_settings_to_eeprom();
          break;
        #endif
        default: Serial.print(F("?>\r\n"));
      }
      command_buffer_index = 0;
      command_buffer[0] = 0;
    }
    if (serial_led) {
      digitalWrite(serial_led, LOW);
    }
  }

}

//--------------------------------------------------------------
void check_buttons(){

  if (button_cw && (digitalRead(button_cw) == LOW)) {
    if (azimuth_button_was_pushed == 0) {
      az_manual_rotation(RIGHT);
      azimuth_button_was_pushed = 1;
      if (debug_mode) {
        Serial.println(F("check_buttons: button_cw pushed"));
      }
    }

  } else {
    if (button_ccw && (digitalRead(button_ccw) == LOW)) {
      if (azimuth_button_was_pushed == 0) {
        az_manual_rotation(LEFT);
        azimuth_button_was_pushed = 1;
        if (debug_mode) {
          Serial.println(F("check_buttons: button_ccw pushed"));
        }
      }
    }
  }

  if ((azimuth_button_was_pushed) && (digitalRead(button_ccw) == HIGH) &&
(digitalRead(button_cw) == HIGH)) {
    delay(200);
    if ((digitalRead(button_ccw) == HIGH) && (digitalRead(button_cw) == HIGH)) {
      az_manual_rotation(STOP);
      azimuth_button_was_pushed = 0;
    }
  }

  #ifdef FEATURE_ELEVATION_CONTROL
  if (button_up && (digitalRead(button_up) == LOW)) {
    if (elevation_button_was_pushed == 0) {
      el_manual_rotation(UP);
      elevation_button_was_pushed = 1;
      if (debug_mode) {
        Serial.println(F("check_buttons: button_up pushed"));
```

```
      }
    }

  } else {
    if (button_down && (digitalRead(button_down) == LOW)) {
      if (elevation_button_was_pushed == 0) {
        el_manual_rotation(DOWN);
        elevation_button_was_pushed = 1;
        if (debug_mode) {
          Serial.println(F("check_buttons: button_down pushed"));
        }
      }
    }
  }

  if ((elevation_button_was_pushed) && (digitalRead(button_up) == HIGH) &&
(digitalRead(button_down) == HIGH)) {
    delay(200);
    if ((digitalRead(button_up) == HIGH) && (digitalRead(button_down) == HIGH))
{
      el_manual_rotation(STOP);
      elevation_button_was_pushed = 0;
    }
  }

  #endif

}
//--------------------------------------------------------------------
#ifdef FEATURE_LCD_DISPLAY
void display_direction() {

  String direction_string = "N";

  if (display_azimuth < 348) {
    if (display_azimuth > 11) {direction_string = "NNE";}
    if (display_azimuth > 33) {direction_string = "NE";}
    if (display_azimuth > 56) {direction_string = "ENE";}
    if (display_azimuth > 78) {direction_string = "E";}
    if (display_azimuth > 101) {direction_string = "ESE";}
    if (display_azimuth > 123) {direction_string = "SE";}
    if (display_azimuth > 146) {direction_string = "SSE";}
    if (display_azimuth > 168) {direction_string = "S";}
    if (display_azimuth > 191) {direction_string = "SSW";}
    if (display_azimuth > 213) {direction_string = "SW";}
    if (display_azimuth > 236) {direction_string = "WSW";}
    if (display_azimuth > 258) {direction_string = "W";}
    if (display_azimuth > 281) {direction_string = "WNW";}
    if (display_azimuth > 303) {direction_string = "NW";}
    if (display_azimuth > 326) {direction_string = "NNW";}
  }
  lcd.setCursor(((LCD_COLUMNS - direction_string.length())/2),0);
  lcd.print(direction_string);

  if (debug_mode) {
    Serial.print(F("display_direction: disp_az: "));
    Serial.print(display_azimuth);
    Serial.print(F("  string: "));
    Serial.println(direction_string);
  }

}
#endif
//--------------------------------------------------------------------
```

```
#ifdef FEATURE_LCD_DISPLAY
void update_display()
{

  // update the LCD display
  // yea, I know I could optimize this code and break out some of the repetitive
tasks into functions
  // but this is a hobby not a profession

  byte number_columns = 0;
  byte larger_lcd = 0;

  static int last_azimuth;

  if (LCD_COLUMNS > 16) {
    larger_lcd = 1;
  }

  if ((millis() - last_lcd_update) > LCD_UPDATE_TIME) {
    if (millis() < 3000) {    // display direction for first time
      lcd.clear();
      display_azimuth = azimuth;
      display_direction();
    }

    if (target_azimuth > 9) { number_columns++; }
    if (target_azimuth > 99) { number_columns++; }

    #ifndef FEATURE_ELEVATION_CONTROL
    if (az_rotation_status != IDLE) {
      if ((az_rotation_status == ROTATING_CCW_AUTO) || (az_rotation_status ==
ROTATING_CW_AUTO)) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 13 - number_columns)/2),0);
        lcd.print("Rotating to ");
        lcd.print(target_azimuth);
        lcd.write(223);
        display_status_dirty = 1;
      }
      if (az_rotation_status == ROTATING_CW) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 11)/2),0);
        lcd.print("Rotating CW");
        display_status_dirty = 1;
      }
      if (az_rotation_status == ROTATING_CCW) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 12)/2),0);
        lcd.print("Rotating CCW");
        display_status_dirty = 1;
      }
    } else {
      if ((display_status_dirty) || (last_azimuth != azimuth)) {
        lcd.clear();
        display_azimuth = azimuth;
        display_direction();
        display_status_dirty = 0;
        last_azimuth = azimuth;
      }
    }
    #endif

    #ifdef FEATURE_ELEVATION_CONTROL
    if ((az_rotation_status != IDLE) && (el_rotation_status == IDLE)) {
```

```
      if ((az_rotation_status == ROTATING_CCW_AUTO) || (az_rotation_status ==
ROTATING_CW_AUTO)) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 13 - number_columns)/2),0);
        lcd.print("Rotating to ");
        lcd.print(target_azimuth);
        lcd.write(223);
        display_status_dirty = 1;
      }
      if (az_rotation_status == ROTATING_CW) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 11)/2),0);
        lcd.print("Rotating CW");
        display_status_dirty = 1;
      }
      if (az_rotation_status == ROTATING_CCW) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 12)/2),0);
        lcd.print("Rotating CCW");
        display_status_dirty = 1;
      }

    }
    if ((az_rotation_status == IDLE) && (el_rotation_status != IDLE)) {
      if ((el_rotation_status == ROTATING_UP_AUTO) || (el_rotation_status ==
ROTATING_DOWN_AUTO)) {
        number_columns = 0;
        if (target_elevation > 9) { number_columns++; }
        if (target_elevation > 99) { number_columns++; }
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 15 - number_columns)/2),0);
        lcd.print("Elevating to ");
        lcd.print(target_elevation);
        lcd.write(223);
        display_status_dirty = 1;
      }
      if (el_rotation_status == ROTATING_UP) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 12)/2),0);
        lcd.print("Elevating up");
        display_status_dirty = 1;
      }
      if (el_rotation_status == ROTATING_DOWN) {
        lcd.clear();
        lcd.setCursor(((LCD_COLUMNS - 14)/2),0);
        lcd.print("Elevating down");
        display_status_dirty = 1;
      }
    }
    if ((az_rotation_status != IDLE) && (el_rotation_status != IDLE)) {
      if (target_elevation > 9) { number_columns++; }
      if (target_elevation > 99) { number_columns++; }
      lcd.clear();
      lcd.setCursor(((LCD_COLUMNS - 11 - number_columns - (2 *
larger_lcd))/2),0);
      lcd.print("Rotating ");
      lcd.print(target_azimuth);
      if (larger_lcd) {lcd.write(223);}
      lcd.print(" ");
      lcd.print(target_elevation);
      if (larger_lcd) {lcd.write(223);}
      display_status_dirty = 1;
    }
```

```
    if ((az_rotation_status == IDLE) && (el_rotation_status == IDLE) &&
((display_status_dirty) || (abs(last_azimuth-azimuth) > 2))) {
        lcd.clear();
        display_azimuth = azimuth;
        last_azimuth = azimuth;
        display_direction();
        display_status_dirty = 0;
    }
    #endif //FEATURE_ELEVATION_CONTROL

    number_columns = 0;
    if (azimuth > 9) { number_columns++; }
    if (azimuth > 99) { number_columns++; }

    #ifndef FEATURE_ELEVATION_CONTROL
    clear_display_row(1);
    lcd.setCursor(((LCD_COLUMNS - 10 - number_columns)/2),1);
    lcd.print("Azimuth ");
    lcd.print(azimuth);
    lcd.write(223);
    #endif //FEATURE_ELEVATION_CONTROL

    #ifdef FEATURE_ELEVATION_CONTROL
    if (elevation > 9) { number_columns++; }
    if (elevation > 99) { number_columns++; }
    clear_display_row(1);
    lcd.setCursor(((LCD_COLUMNS - 9 - number_columns - (2 * larger_lcd))/2),1);
    lcd.print("Az ");
    lcd.print(azimuth);
    if (larger_lcd) {lcd.write(223);}
    lcd.print(" El ");
    lcd.print(elevation);
    if (larger_lcd) {lcd.write(223);}
    #endif //FEATURE_ELEVATION_CONTROL

    last_lcd_update = millis();
  }
}
#endif
//--------------------------------------------------------------
#ifdef FEATURE_LCD_DISPLAY
void clear_display_row(byte row_number)
{
  for (byte x = 0; x < LCD_COLUMNS; x++) {
    lcd.setCursor(x,row_number);
    lcd.print(" ");
  }
}
#endif

//--------------------------------------------------------------
void get_keystroke()
{
    while (Serial.available() == 0) {}
    while (Serial.available() > 0) {
      incoming_serial_byte = Serial.read();
    }
}

//--------------------------------------------------------------

void x_command()
{
```

```
  if (command_buffer_index > 1) {
    if (azimuth_speed_voltage) {
      switch (command_buffer[1]) {
        case '4':
          analogWrite(azimuth_speed_voltage, PWM_SPEED_VOLTAGE_X4);
          current_azimuth_speed_voltage = PWM_SPEED_VOLTAGE_X4;
          Serial.print(F("Speed X4\r\n"));
          break;
        case '3':
          analogWrite(azimuth_speed_voltage, PWM_SPEED_VOLTAGE_X3);
          current_azimuth_speed_voltage = PWM_SPEED_VOLTAGE_X3;
          Serial.print(F("Speed X3\r\n"));
          break;
        case '2':
          analogWrite(azimuth_speed_voltage, PWM_SPEED_VOLTAGE_X2);
          current_azimuth_speed_voltage = PWM_SPEED_VOLTAGE_X2;
          Serial.print(F("Speed X2\r\n"));
          break;
        case '1':
          analogWrite(azimuth_speed_voltage, PWM_SPEED_VOLTAGE_X1);
          current_azimuth_speed_voltage = PWM_SPEED_VOLTAGE_X1;
          Serial.print(F("Speed X1\r\n"));
          break;
        default: Serial.print(F("?>\r\n")); break;
      }
    } else {
      Serial.print(F("error: azimuth_speed_voltage pin not enabled\r\n"));
    }
  } else {
      Serial.print(F("?>\r\n"));
  }
}

//---------------------------------------------------------------
#ifdef FEATURE_GS_232B_EMULATION
void p_command()
{
  if ((command_buffer[1] == '3') && (command_buffer_index > 2)) {  // P36
command
    azimuth_rotation_capability = 360;
    Serial.print(F("Mode 360 degree\r\n"));
    write_settings_to_eeprom();
  } else {
    if ((command_buffer[1] == '4') && (command_buffer_index > 2)) { // P45
command
      azimuth_rotation_capability = 450;
      Serial.print(F("Mode 450 degree\r\n"));
      write_settings_to_eeprom();
    } else {
      Serial.print(F("?>\r\n"));
    }
  }

}


#endif //FEATURE_GS_232B_EMULATION
//---------------------------------------------------------------


void o_command()
{
```

```
  #ifdef FEATURE_ELEVATION_CONTROL
  if ((command_buffer[1] == '2') && (command_buffer_index > 1)) {     // did we
get the O2 command?
    o2_command();
    return;
  }
  #endif

  serial_print(az_offset_set_string);
  get_keystroke();
  read_azimuth();
  analog_az_full_ccw = analog_az;
  write_settings_to_eeprom();
  serial_print(wrote_to_memory_string);
}

//--------------------------------------------------------------

void f_command()
{

  #ifdef FEATURE_ELEVATION_CONTROL
  if ((command_buffer[1] == '2') && (command_buffer_index > 1)) {     // did we
get the F2 command?
    f2_command();
    return;
  }
  #endif

  serial_print(az_fullscale_set_string);
  get_keystroke();
  read_azimuth();
  analog_az_full_cw = analog_az;
  write_settings_to_eeprom();
  serial_print(wrote_to_memory_string);
}

//--------------------------------------------------------------

#ifdef FEATURE_ELEVATION_CONTROL
void o2_command()
{
  serial_print(el_offset_set_string);
  get_keystroke();
  read_elevation();
  analog_el_0_degrees = analog_el;
  write_settings_to_eeprom();
  serial_print(wrote_to_memory_string);
}
#endif

//--------------------------------------------------------------

#ifdef FEATURE_ELEVATION_CONTROL
void f2_command()
{
  serial_print(el_fullscale_set_string);
  get_keystroke();
  read_elevation();
  analog_el_max_elevation = analog_el;
  write_settings_to_eeprom();
  serial_print(wrote_to_memory_string);
}
#endif
```

```
//--------------------------------------------------------------

void read_settings_from_eeprom()
{

  if ( EEPROM.read(EEPROM_ID_BYTE) == 74) {
    if (debug_mode) {
      Serial.print(F("read_settings_from_eeprom: reading settings from
eeprom\n"));
      Serial.println(EEPROM.read(EEPROM_ANALOG_AZ_FULL_CCW),DEC);
      Serial.println(EEPROM.read(EEPROM_ANALOG_AZ_FULL_CCW+1),DEC);
      Serial.println(EEPROM.read(EEPROM_ANALOG_AZ_FULL_CW),DEC);
      Serial.println(EEPROM.read(EEPROM_ANALOG_AZ_FULL_CW+1),DEC);
    }
    analog_az_full_ccw = (EEPROM.read(EEPROM_ANALOG_AZ_FULL_CCW)*256) +
EEPROM.read(EEPROM_ANALOG_AZ_FULL_CCW+1);
    analog_az_full_cw = (EEPROM.read(EEPROM_ANALOG_AZ_FULL_CW)*256) +
EEPROM.read(EEPROM_ANALOG_AZ_FULL_CW+1);
    analog_el_0_degrees = (EEPROM.read(EEPROM_ANALOG_EL_0_DEGREES)*256) +
EEPROM.read(EEPROM_ANALOG_EL_0_DEGREES+1);
    analog_el_max_elevation = (EEPROM.read(EEPROM_ANALOG_EL_MAX_ELEVATION)*256)
+ EEPROM.read(EEPROM_ANALOG_EL_MAX_ELEVATION+1);

    if (EEPROM.read(EEPROM_AZIMUTH_ROTATION_CAPABILITY) == 255) {  // hack to
initialize pre 2011092701 units
      if (debug_mode) {
        Serial.print(F("read_settings_from_eeprom: pre 2011092701 eeprom init
hack\n"));
      }
      write_settings_to_eeprom();
    } else {
      azimuth_starting_point = (EEPROM.read(EEPROM_AZIMUTH_STARTING_POINT)*256)
+ EEPROM.read(EEPROM_AZIMUTH_STARTING_POINT+1);
      azimuth_rotation_capability =
(EEPROM.read(EEPROM_AZIMUTH_ROTATION_CAPABILITY)*256) +
EEPROM.read(EEPROM_AZIMUTH_ROTATION_CAPABILITY+1);
    }


  } else {  // initialize eeprom with default values
    if (debug_mode) {
      Serial.print(F("read_settings_from_eeprom: uninitialized eeprom, using
defaults\n"));
    }
    analog_az_full_ccw = ANALOG_AZ_FULL_CCW;
    analog_az_full_cw = ANALOG_AZ_FULL_CW;
    analog_el_0_degrees = ANALOG_EL_0_DEGREES;
    analog_el_max_elevation = ANALOG_EL_MAX_ELEVATION;
    write_settings_to_eeprom();
  }
}
//--------------------------------------------------------------
void write_settings_to_eeprom()
{
  if (debug_mode) {
    Serial.print(F("write_settings_to_eeprom: writing settings to eeprom\n"));
  }

  EEPROM.write(EEPROM_ID_BYTE,74);
  EEPROM.write(EEPROM_ANALOG_AZ_FULL_CCW,highByte(analog_az_full_ccw));
  EEPROM.write(EEPROM_ANALOG_AZ_FULL_CCW+1,lowByte(analog_az_full_ccw));
  EEPROM.write(EEPROM_ANALOG_AZ_FULL_CW,highByte(analog_az_full_cw));
  EEPROM.write(EEPROM_ANALOG_AZ_FULL_CW+1,lowByte(analog_az_full_cw));
```

```
  EEPROM.write(EEPROM_ANALOG_EL_0_DEGREES,highByte(analog_el_0_degrees));
  EEPROM.write(EEPROM_ANALOG_EL_0_DEGREES+1,lowByte(analog_el_0_degrees));

EEPROM.write(EEPROM_ANALOG_EL_MAX_ELEVATION,highByte(analog_el_max_elevation));

EEPROM.write(EEPROM_ANALOG_EL_MAX_ELEVATION+1,lowByte(analog_el_max_elevation));
  EEPROM.write(EEPROM_AZIMUTH_STARTING_POINT,highByte(azimuth_starting_point));
  EEPROM.write(EEPROM_AZIMUTH_STARTING_POINT+1,lowByte(azimuth_starting_point));

EEPROM.write(EEPROM_AZIMUTH_ROTATION_CAPABILITY,highByte(azimuth_rotation_capabi
lity));

EEPROM.write(EEPROM_AZIMUTH_ROTATION_CAPABILITY+1,lowByte(azimuth_rotation_capab
ility));
}

//--------------------------------------------------------------

void serial_print(prog_uchar str[])
{
  char c;
  while((c = pgm_read_byte(str++))) {
    Serial.write(c);
  }
}

//--------------------------------------------------------------

void az_check_operation_timeout()
{

  // check if the last executed rotation operation has been going on too long

  if (((millis() - az_last_rotate_initiation) > OPERATION_TIMEOUT) &&
(az_rotation_status != IDLE)) {
    az_manual_rotation(STOP);
    if (debug_mode) {
      serial_print(az_abort_rotation_string);
    }
  }
}

//--------------------------------------------------------------
void az_manual_rotation (int rotation)
{
  switch(rotation) {
    case STOP:
      rotator(DEACTIVATE,ROTATION_CW);
      rotator(DEACTIVATE,ROTATION_CCW);
      /*digitalWrite(rotate_cw,LOW);digitalWrite(rotate_ccw,LOW);*/
      az_rotation_status = IDLE;
      if (debug_mode) {
        Serial.println(F("az_manual_rotation: az_rotation_status = IDLE"));
      }
      break;
    case LEFT:
      rotator(ACTIVATE,ROTATION_CCW);
      /*digitalWrite(rotate_cw,LOW);digitalWrite(rotate_ccw,HIGH);*/
      az_rotation_status = ROTATING_CCW;
      az_last_rotate_initiation = millis();
      if (debug_mode) {
        Serial.println(F("az_manual_rotation: az_rotation_status =
ROTATING_CCW"));
      }
```

```
      break;
    case RIGHT:
      rotator(ACTIVATE,ROTATION_CW);
      /*digitalWrite(rotate_cw,HIGH);digitalWrite(rotate_ccw,LOW);*/
      az_rotation_status = ROTATING_CW;
      az_last_rotate_initiation = millis();
      if (debug_mode) {
        Serial.println(F("az_manual_rotation: az_rotation_status =
ROTATING_CW"));
      }
      break;
  }
}

//--------------------------------------------------------------

void az_autorotate_cw()
{
  rotator(ACTIVATE,ROTATION_CW);
  //digitalWrite(rotate_cw,HIGH);
  //digitalWrite(rotate_ccw,LOW);
  az_rotation_status = ROTATING_CW_AUTO;
  az_last_rotate_initiation = millis();
}

//--------------------------------------------------------------

void az_autorotate_ccw()
{
  rotator(ACTIVATE,ROTATION_CCW);
  //digitalWrite(rotate_cw,LOW);
  //digitalWrite(rotate_ccw,HIGH);
  az_rotation_status = ROTATING_CCW_AUTO;
  az_last_rotate_initiation = millis();
}

//--------------------------------------------------------------
void clear_timed_buffer()
{
  timed_buffer_status = EMPTY;
  timed_buffer_number_entries_loaded = 0;
  timed_buffer_entry_pointer = 0;
}

//--------------------------------------------------------------
//AAAAA
void az_initiate_auto_rotation (int target_azimuth_in)
{
  // start autorotation, selecting the right direction

  int parsed_azimuth;

  // parse out M command
  if (target_azimuth_in < 0) {       // if we were not passed a target azimuth,
parse the command buffer
    if (command_buffer_index > 4) {  // if there are more than 4 characters in
the command buffer, we got a timed interval command
      az_load_timed_intervals();
      return;
    } else {                         // if there were less than four characters,
this is just a single direction setting
      parsed_azimuth = ((int(command_buffer[1])-48)*100) +
((int(command_buffer[2])-48)*10) + (int(command_buffer[3])-48);
      clear_timed_buffer();
```

```
    }
  } else {                            // we were passed a specific target
azimuth, use that
    parsed_azimuth = target_azimuth_in;
  }
  if ((parsed_azimuth > -1) && (parsed_azimuth < 361)) {
    target_azimuth = parsed_azimuth;
    if (target_azimuth == 360) {
      target_azimuth = 0;
    }

    if ((target_azimuth > (azimuth - AZIMUTH_TOLERANCE)) && (target_azimuth <
(azimuth + AZIMUTH_TOLERANCE))) {
      if (debug_mode) {
        serial_print(req_azimuth_with_tol_string);
      }
    } else {  // target azimuth is not within tolerance, we need to rotate
      int target_raw_azimuth = target_azimuth;

      if (target_raw_azimuth < azimuth_starting_point) {
        target_raw_azimuth = target_raw_azimuth + 360;
      }
      if ((target_raw_azimuth + 360) < (azimuth_starting_point +
azimuth_rotation_capability)) { // is there a second possible heading in
overlap?
        if (abs(raw_azimuth - target_raw_azimuth) < abs((target_raw_azimuth+360)
- raw_azimuth)) { // is second possible heading closer?
          if (target_raw_azimuth  > raw_azimuth) { // not closer, use position
in non-overlap
            az_autorotate_cw();
          } else {
            az_autorotate_ccw();
          }
        } else {
          if ((target_raw_azimuth + 360)  > raw_azimuth) { // go to position in
overlap
            az_autorotate_cw();
          } else {
            az_autorotate_ccw();
          }
        }
      } else {  // no possible second heading in overlap
        if (target_raw_azimuth  > raw_azimuth) {
          az_autorotate_cw();
        } else {
          az_autorotate_ccw();
        }
      }
    }
  } else {
    Serial.print(F("?>"));      // bogus azimuth - return and error and don't do
anything
  }
  Serial.println();
}

//--------------------------------------------------------------
void initiate_timed_buffer()
{
  if (timed_buffer_status == LOADED_AZIMUTHS) {
    timed_buffer_status = RUNNING_AZIMUTHS;
    az_initiate_auto_rotation(timed_buffer_azimuths[1]);
    last_timed_buffer_action_time = millis();
    timed_buffer_entry_pointer = 2;
```

```
    if (debug_mode) {
      serial_print(running_azimuths_string);
    }
  } else {
    #ifdef FEATURE_ELEVATION_CONTROL
    if (timed_buffer_status == LOADED_AZIMUTHS_ELEVATIONS) {
      timed_buffer_status = RUNNING_AZIMUTHS_ELEVATIONS;
      az_initiate_auto_rotation(timed_buffer_azimuths[1]);
      el_initiate_auto_rotation(timed_buffer_elevations[1]);
      last_timed_buffer_action_time = millis();
      timed_buffer_entry_pointer = 2;
      if (debug_mode) {
        serial_print(running_azimuths_elevations_string);
      }
    } else {
      Serial.write(">\r\n");  // error
    }
    #endif
  }

}

//------------------------------------------------------------
void check_timed_interval()
{

  if ((timed_buffer_status == RUNNING_AZIMUTHS) && (((millis() -
last_timed_buffer_action_time)/1000) > timed_buffer_interval_value_seconds)) {
    timed_buffer_entry_pointer++;
    serial_print(exec_next_time_az_string);
    az_initiate_auto_rotation(timed_buffer_azimuths[timed_buffer_entry_pointer-
1]);
    last_timed_buffer_action_time = millis();
    if (timed_buffer_entry_pointer == timed_buffer_number_entries_loaded) {
      clear_timed_buffer();
      if (debug_mode) {
        serial_print(complete_time_buff_string);
      }
    }
  }
  #ifdef FEATURE_ELEVATION_CONTROL
  if ((timed_buffer_status == RUNNING_AZIMUTHS_ELEVATIONS) && (((millis() -
last_timed_buffer_action_time)/1000) > timed_buffer_interval_value_seconds)) {
    timed_buffer_entry_pointer++;
    serial_print(exec_next_time_az_el_string);
    az_initiate_auto_rotation(timed_buffer_azimuths[timed_buffer_entry_pointer-
1]);

el_initiate_auto_rotation(timed_buffer_elevations[timed_buffer_entry_pointer-
1]);
    last_timed_buffer_action_time = millis();
    if (timed_buffer_entry_pointer == timed_buffer_number_entries_loaded) {
      clear_timed_buffer();
      if (debug_mode) {
        serial_print(complete_time_buff_string);
      }
    }
  }
  #endif
}

//------------------------------------------------------------

void az_load_timed_intervals()
```

```
{
  int parsed_value = 0;

  clear_timed_buffer();

  parsed_value = ((int(command_buffer[1])-48)*100) + ((int(command_buffer[2])-
48)*10) + (int(command_buffer[3])-48);
  if ((parsed_value > 0) && (parsed_value < 1000)) {
    timed_buffer_interval_value_seconds = parsed_value;
    for (int x = 5; x < command_buffer_index; x = x + 4) {
      parsed_value = ((int(command_buffer[x])-48)*100) +
((int(command_buffer[x+1])-48)*10) + (int(command_buffer[x+2])-48);
      if ((parsed_value > -1) && (parsed_value < 361)) {  // is it a valid
azimuth?
        timed_buffer_azimuths[timed_buffer_number_entries_loaded] =
parsed_value;
        timed_buffer_number_entries_loaded++;
        timed_buffer_status = LOADED_AZIMUTHS;
        if (timed_buffer_number_entries_loaded > TIMED_INTERVAL_ARRAY_SIZE)
{   // is the array full?
          az_initiate_auto_rotation(timed_buffer_azimuths[0]);  // array is
full, go to the first azimuth
          timed_buffer_entry_pointer = 1;
          return;
        }
      } else {   // we hit an invalid bearing
        timed_buffer_status = EMPTY;
        timed_buffer_number_entries_loaded = 0;
        Serial.write("?>\r\n");  // error
        return;
      }
    }
    az_initiate_auto_rotation(timed_buffer_azimuths[0]);   // go to the first
azimuth
    timed_buffer_entry_pointer = 1;

  } else {
    Serial.print(F("?>\r\n"));  // error
  }

}

//-------------------------------------------------------------

void read_azimuth()
{
  // read analog input and convert it to degrees; this gets funky because of 450
degree rotation
  // Bearings:  180-------359-0--------270
  // Voltage:    0----------------------5

  analog_az = analogRead(rotator_analog_az);
  raw_azimuth = map(analog_az, analog_az_full_ccw, analog_az_full_cw,
azimuth_starting_point, (azimuth_starting_point + azimuth_rotation_capability));
  if (raw_azimuth > 359) {
    azimuth = raw_azimuth - 360;
      if (azimuth > 359) {
        azimuth = azimuth - 360;
      }
  } else {
    if (raw_azimuth < 0) {
      azimuth = raw_azimuth + 360;
    } else {
      azimuth = raw_azimuth;
```

```
    }
  }
}

//-----------------------------------------------------------------

void output_debug()
{


  if ((millis() - last_debug_output_time) > 3000) {
    Serial.write("--------debug---------\n");
    Serial.println(CODE_VERSION);
    serial_print(Analog_Azimuth_string);
    Serial.println(analog_az);
    serial_print(Azimuth_string);
    Serial.println(azimuth);
    serial_print(Raw_Azimuth_string);
    Serial.println(raw_azimuth);
    serial_print(Azimuth_Starting_Point_string);
    Serial.println(azimuth_starting_point);
    serial_print(azimuth_rotation_capability_string);
    Serial.println(azimuth_rotation_capability);
    if (azimuth_speed_voltage) {
      Serial.print(F("Current Az Speed Voltage: "));
      Serial.println(current_azimuth_speed_voltage, DEC);
    }
    Serial.write("AZ Rotation Status: ");
    switch (az_rotation_status) {
      case IDLE: Serial.print(F("IDLE\n")); break;
      case ROTATING_CW: Serial.print(F("ROTATING_CW\n")); break;
      case ROTATING_CCW: Serial.print(F("ROTATING_CCW\n")); break;
      case ROTATING_CW_AUTO: Serial.print(F("ROTATING_CW_AUTO\n")); break;
      case ROTATING_CCW_AUTO: Serial.print(F("ROTATING_CCW_AUTO\n")); break;
    }
    Serial.print(F("Target Azimuth: "));
    Serial.println(target_azimuth);

    #ifdef FEATURE_ELEVATION_CONTROL
    Serial.print(F("Analog Elevation: "));
    Serial.println(analog_el);
    Serial.print(F("Elevation: "));
    Serial.println(elevation);
    Serial.print(F("EL Rotation Status: "));
    switch (el_rotation_status) {
      case IDLE: Serial.print(F("IDLE\n")); break;
      case ROTATING_UP: Serial.print(F("ROTATING_UP\n")); break;
      case ROTATING_DOWN: Serial.print(F("ROTATING_DOWN\n")); break;
      case ROTATING_UP_AUTO: Serial.print(F("ROTATING_UP_AUTO\n")); break;
      case ROTATING_DOWN_AUTO: Serial.print(F("ROTATING_DOWN_AUTO\n")); break;
    }
    Serial.print(F("Target Elevation: "));
    Serial.println(target_elevation);
    #endif



    serial_print(Timed_interval_buffer_status_string);
    switch (timed_buffer_status) {
      case EMPTY: Serial.print(F("EMPTY\n")); break;
      case LOADED_AZIMUTHS: Serial.print(F("LOADED_AZIMUTHS\n")); break;
      case RUNNING_AZIMUTHS: Serial.print(F("RUNNING_AZIMUTHS\n")); break;
      #ifdef FEATURE_ELEVATION_CONTROL
      case LOADED_AZIMUTHS_ELEVATIONS:
```

```
Serial.print(F("LOADED_AZIMUTHS_ELEVATIONS\n")); break;
      case RUNNING_AZIMUTHS_ELEVATIONS:
Serial.print(F("RUNNING_AZIMUTHS_ELEVATIONS\n")); break;
      #endif
    }

    serial_print(Timed_interval_buffer_time_interval_seconds_string);
    Serial.println(timed_buffer_interval_value_seconds,DEC);
    Serial.print(F("Timed interval buffer entries loaded: "));
    Serial.println (timed_buffer_number_entries_loaded,DEC);
    Serial.print(F("Timed interval current entry pointer: "));
    Serial.println(timed_buffer_entry_pointer,DEC);
    Serial.print(F("Seconds since last timed interval action: "));
    Serial.println((millis()-last_timed_buffer_action_time)/1000);

    if (timed_buffer_number_entries_loaded > 0) {
      for (int x = 0;x < timed_buffer_number_entries_loaded; x++) {
        Serial.print(x+1);
        Serial.print(F("\t:"));
        Serial.print(timed_buffer_azimuths[x]);
        #ifdef FEATURE_ELEVATION_CONTROL
        Serial.print(F("\t- "));
        Serial.print(timed_buffer_elevations[x]);
        #endif
        Serial.println();
      }

    }

    Serial.print(F("analog_az_full_ccw: "));
    Serial.println(analog_az_full_ccw);
    Serial.print(F("analog_az_full_cw: "));
    Serial.println(analog_az_full_cw);
    #ifdef FEATURE_ELEVATION_CONTROL
    Serial.print(F("analog_el_0_degrees: "));
    Serial.println(analog_el_0_degrees);
    Serial.print(F("analog_el_max_elevation: "));
    Serial.println(analog_el_max_elevation);
    #endif


    last_debug_output_time = millis();

    Serial.print(F("GS-232"));
    #ifdef FEATURE_GS_232B_EMULATION
    Serial.print(F("B"));
    #endif
    #ifndef FEATURE_GS_232B_EMULATION
    Serial.print(F("A"));
    #endif
    Serial.print(F(" emulation\n"));

    #ifdef DEBUG_MEMORY
    void* HP = malloc(4);
    if (HP) {
      free (HP);
    }
    unsigned long free = (unsigned long)SP - (unsigned long)HP;
    if (free > 2048) {
      free = 0;
    }
    Serial.print((unsigned long)free,DEC);
    Serial.print(F(" bytes free\n"));
    #endif
```

```
    Serial.print(F("\n\n\n\n"));

  }
}

//-------------------------------------------------------------

void az_check_rotation()
{
  // if we're in autorotation, check to see if we're at our destination and stop

// old code - new code below is better at preventing rotation overruns
//  if ((az_rotation_status == ROTATING_CW_AUTO) || (az_rotation_status ==
ROTATING_CCW_AUTO)) {
//    if (abs(azimuth - target_azimuth) < AZIMUTH_TOLERANCE) {
//      delay(50);
//      read_azimuth();
//      if (abs(azimuth - target_azimuth) < AZIMUTH_TOLERANCE) {
//        az_manual_rotation(STOP);
//        if (debug_mode) {
//          serial_print(target_az_reach_stop_string);
//        }
//      }
//    }
//  }

  if (az_rotation_status == ROTATING_CW_AUTO) {
    if ((abs(azimuth - target_azimuth) < AZIMUTH_TOLERANCE) || ((azimuth >
target_azimuth) && ((azimuth - target_azimuth) < (AZIMUTH_TOLERANCE+5)))) {
      delay(50);
      read_azimuth();
      if ((abs(azimuth - target_azimuth) < AZIMUTH_TOLERANCE) || ((azimuth >
target_azimuth) && ((azimuth - target_azimuth) < (AZIMUTH_TOLERANCE+5)))) {
        az_manual_rotation(STOP);
        if (debug_mode) {
          serial_print(target_az_reach_stop_string);
        }
      }
    }
  }

  if (az_rotation_status == ROTATING_CCW_AUTO) {
    if ((abs(azimuth - target_azimuth) < AZIMUTH_TOLERANCE) || ((azimuth <
target_azimuth) && ((target_azimuth - azimuth) < (AZIMUTH_TOLERANCE+5)))) {
      delay(50);
      read_azimuth();
      if ((abs(azimuth - target_azimuth) < AZIMUTH_TOLERANCE) || ((azimuth <
target_azimuth) && ((target_azimuth - azimuth) < (AZIMUTH_TOLERANCE+5)))) {
        az_manual_rotation(STOP);
        if (debug_mode) {
          serial_print(target_az_reach_stop_string);
        }
      }
    }
  }

}

//-------------------------------------------------------------
void report_current_azimuth() {

  // The C command that reports azimuth in +0nnn format
```

```
  String azimuth_string;

  #ifndef FEATURE_GS_232B_EMULATION
  Serial.print(F("+0"));
  #endif
  #ifdef FEATURE_GS_232B_EMULATION
  Serial.print(F("AZ="));
  #endif
  //Serial.write("report_current_azimith: azimuth=");
  //Serial.println(azimuth);
  azimuth_string = String(azimuth, DEC);
  if (azimuth_string.length() == 1) {
    Serial.print(F("00"));
  } else {
    if (azimuth_string.length() == 2) {
      Serial.print(F("0"));
    }
  }
  Serial.print(azimuth_string);

  #ifdef FEATURE_ELEVATION_CONTROL
  #ifndef OPTION_C_COMMAND_SENDS_AZ_AND_EL
  if ((command_buffer[1] == '2') && (command_buffer_index > 1)) {     // did we
get the C2 command?
  #endif
    report_current_elevation();
  #ifndef OPTION_C_COMMAND_SENDS_AZ_AND_EL
  }
  #endif
  #endif
  Serial.print(F("\r\n"));
}

//--------------------------------------------------------------

void print_help(){

  // The H command

  serial_print(serial_help_string);

}

//--------------- Elevation -----------------------

#ifdef FEATURE_ELEVATION_CONTROL
void el_check_operation_timeout()
{

  // check if the last executed rotation operation has been going on too long

  if (((millis() - el_last_rotate_initiation) > OPERATION_TIMEOUT) &&
(el_rotation_status != IDLE)) {
    el_manual_rotation(STOP);
    if (debug_mode) {
      Serial.print(F("el_check_operation_timeout: timeout reached, aborting
rotation\r\n"));
    }
  }
}
#endif

//--------------------------------------------------------------
```

```
#ifdef FEATURE_ELEVATION_CONTROL
void el_manual_rotation (int rotation)
{

  switch(rotation) {
    case STOP:
rotator(DEACTIVATE,ROTATION_UP);rotator(DEACTIVATE,ROTATION_DOWN);/*digitalWrite
(rotate_up,LOW);digitalWrite(rotate_down,LOW);*/el_rotation_status = IDLE;break;
    case UP:
rotator(ACTIVATE,ROTATION_UP);/*digitalWrite(rotate_down,LOW);digitalWrite(rotat
e_up,HIGH);*/el_rotation_status = ROTATING_UP;el_last_rotate_initiation =
millis();break;
    case DOWN:
rotator(ACTIVATE,ROTATION_DOWN);/*digitalWrite(rotate_down,HIGH);digitalWrite(ro
tate_up,LOW);*/el_rotation_status = ROTATING_DOWN;el_last_rotate_initiation =
millis();break;
  }
}
#endif

//---------------------------------------------------------------

#ifdef FEATURE_ELEVATION_CONTROL
void el_autorotate_up()
{
  //digitalWrite(rotate_up,HIGH);
  //digitalWrite(rotate_down,LOW);
  rotator(ACTIVATE,ROTATION_UP);
  el_rotation_status = ROTATING_UP_AUTO;
  el_last_rotate_initiation = millis();
}
#endif



//---------------------------------------------------------------

#ifdef FEATURE_ELEVATION_CONTROL
void el_autorotate_down()
{
  //digitalWrite(rotate_up,LOW);
  //digitalWrite(rotate_down,HIGH);
  rotator(ACTIVATE,ROTATION_DOWN);
  el_rotation_status = ROTATING_DOWN_AUTO;
  el_last_rotate_initiation = millis();
}
#endif

//---------------------------------------------------------------

#ifdef FEATURE_ELEVATION_CONTROL
void el_initiate_auto_rotation (int elevation_in)
{
  target_elevation = elevation_in;
  if ((target_elevation > (elevation - ELEVATION_TOLERANCE)) &&
(target_elevation < (elevation + ELEVATION_TOLERANCE))) {
    if (debug_mode) {
      Serial.print(F("el_initial_auto_rotation: requested elevation within
tolerance\r\n"));
    }
  } else {
    if (target_elevation > elevation) {
      el_autorotate_up();
    } else {
```

```
      el_autorotate_down();
    }
  }
}
#endif

//------------------------------------------------------------

#ifdef FEATURE_ELEVATION_CONTROL
void az_el_initiate_auto_rotation ()
{
  // start autorotation, selecting the right direction

  int parsed_elevation = 0;
  int parsed_azimuth = 0;
  int parsed_value1 = 0;
  int parsed_value2 = 0;

  // parse out W command
  // Short Format: WXXX YYY = azimuth YYY = elevation
  // Long Format : WSSS XXX YYY  SSS = timed interval   XXX = azimuth    YYY =
elevation

  if (command_buffer_index > 8) {  // if there are more than 4 characters in the
command buffer, we got a timed interval command
    parsed_value1 = ((int(command_buffer[1])-48)*100) +
((int(command_buffer[2])-48)*10) + (int(command_buffer[3])-48);
    if ((parsed_value1 > 0) && (parsed_value1 < 1000)) {
      timed_buffer_interval_value_seconds = parsed_value1;
      for (int x = 5; x < command_buffer_index; x = x + 8) {
        parsed_value1 = ((int(command_buffer[x])-48)*100) +
((int(command_buffer[x+1])-48)*10) + (int(command_buffer[x+2])-48);
        parsed_value2 = ((int(command_buffer[x+4])-48)*100) +
((int(command_buffer[x+5])-48)*10) + (int(command_buffer[x+6])-48);
        if ((parsed_value1 > -1) && (parsed_value1 < 361) && (parsed_value2 >
-1) && (parsed_value2 < 181)) {  // is it a valid azimuth?
          timed_buffer_azimuths[timed_buffer_number_entries_loaded] =
parsed_value1;
          timed_buffer_elevations[timed_buffer_number_entries_loaded] =
parsed_value2;
          timed_buffer_number_entries_loaded++;
          timed_buffer_status = LOADED_AZIMUTHS_ELEVATIONS;
          if (timed_buffer_number_entries_loaded > TIMED_INTERVAL_ARRAY_SIZE) {
// is the array full?
            x = command_buffer_index;  // array is full, go to the first azimuth
and elevation

          }
        } else {   // we hit an invalid bearing
          timed_buffer_status = EMPTY;
          timed_buffer_number_entries_loaded = 0;
          Serial.print(F("?>\r\n"));  // error
          return;
        }
      }
    }
    timed_buffer_entry_pointer = 1;             // go to the first bearings
    parsed_azimuth = timed_buffer_azimuths[0];
    parsed_elevation = timed_buffer_elevations[0];
  } else {
    // this is a short form W command, just parse the azimuth and elevation and
initiate rotation
    parsed_azimuth = ((int(command_buffer[1])-48)*100) +
((int(command_buffer[2])-48)*10) + (int(command_buffer[3])-48);
```

```
    parsed_elevation = ((int(command_buffer[5])-48)*100) +
((int(command_buffer[6])-48)*10) + (int(command_buffer[7])-48);
  }

  if ((parsed_azimuth > -1) && (parsed_azimuth < 361)) {
    az_initiate_auto_rotation(parsed_azimuth);
  } else {

    if (debug_mode) {
      serial_print(el_w_cmd_erro_string);
    }
    Serial.print(F("?>\r\n"));      // bogus elevation - return and error and
don't do anything
    return;
  }

  if ((parsed_elevation > -1) && (parsed_elevation < 181)) {
    el_initiate_auto_rotation(parsed_elevation);
  } else {
    if (debug_mode) {
      serial_print(el_w_cmd_erro_string);
    }
    Serial.print(F("?>\r\n"));      // bogus elevation - return and error and
don't do anything
    return;
  }
  Serial.print(F("\r\n"));
}
#endif

//**************************************************************

#ifdef FEATURE_ELEVATION_CONTROL
void read_elevation()
{
  // read analog input and convert it to degrees

  analog_el = analogRead(rotator_analog_el);
  elevation = map(analog_el, analog_el_0_degrees, analog_el_max_elevation, 0,
ELEVATION_MAXIMUM_DEGREES);
  if (elevation < 0) {
    elevation = 0;
  }
}
#endif

//**************************************************************

#ifdef FEATURE_ELEVATION_CONTROL
void el_check_rotation()
{
  // if we're in autorotation, check to see if we're at our destination and stop

  if ((el_rotation_status == ROTATING_UP_AUTO) || (el_rotation_status ==
ROTATING_DOWN_AUTO)) {
    if ( abs(elevation - target_elevation) < ELEVATION_TOLERANCE ) {
      el_manual_rotation(STOP);
      if (debug_mode) {
        serial_print(target_el_reached_string);
        Serial.print(elevation,DEC);
        Serial.print(F("\r\n"));
      }
    }
  }
```

```
}
#endif

//--------------------------------------------------------------

#ifdef FEATURE_ELEVATION_CONTROL
void report_current_elevation() {

  // The C2 command that reports elevation in +0nnn format

  String elevation_string;

  #ifndef FEATURE_GS_232B_EMULATION
  Serial.print(F("+0"));
  #endif
  #ifdef FEATURE_GS_232B_EMULATION
  Serial.print(F("EL="));
  #endif
  elevation_string = String(elevation, DEC);
  if (elevation_string.length() == 1) {
    Serial.print(F("00"));
  } else {
    if (elevation_string.length() == 2) {
      Serial.print(F("0"));
    }
  }
  Serial.print(elevation_string);
  Serial.print(F("\r\n"));
}

#endif

//--------------------------------------------------------------

void rotator(byte rotation_action, byte rotation_type) {

  if (debug_mode) { Serial.print(F("rotator: ")); }


  switch(rotation_type) {
    case ROTATION_CW:
      if (debug_mode) { Serial.print(F("ROTATION_CW ")); }
      if (rotation_action == ACTIVATE) {
        if (debug_mode) { Serial.println(F("ACTIVATE")); }
        brake_release(AZ, ENGAGE);
        digitalWrite(rotate_cw,HIGH);
        digitalWrite(rotate_ccw,LOW);
      } else {
        if (debug_mode) { Serial.println(F("DEACTIVATE")); }
        digitalWrite(rotate_cw,LOW);
      }
      break;
    case ROTATION_CCW:
      if (debug_mode) { Serial.print(F("ROTATION_CCW ")); }
      if (rotation_action == ACTIVATE) {
        if (debug_mode) { Serial.println(F("ACTIVATE")); }
        brake_release(AZ, ENGAGE);
        digitalWrite(rotate_cw,LOW);
        digitalWrite(rotate_ccw,HIGH);
      } else {
        if (debug_mode) { Serial.println(F("DEACTIVATE")); }
        digitalWrite(rotate_ccw,LOW);
      }
      break;
```

```
    #ifdef FEATURE_ELEVATION_CONTROL
    case ROTATION_UP:
      if (debug_mode) { Serial.print(F("ROTATION_UP ")); }
      if (rotation_action == ACTIVATE) {
        if (debug_mode) { Serial.println(F("ACTIVATE")); }
        brake_release(EL, ENGAGE);
        digitalWrite(rotate_up,HIGH);
        digitalWrite(rotate_down,LOW);
      } else {
        if (debug_mode) { Serial.println(F("DEACTIVATE")); }
        digitalWrite(rotate_up,LOW);
      }
      break;
    case ROTATION_DOWN:
      if (debug_mode) { Serial.print(F("ROTATION_DOWN ")); }
      if (rotation_action == ACTIVATE) {
        if (debug_mode) { Serial.println(F("ACTIVATE")); }
        brake_release(EL, ENGAGE);
        digitalWrite(rotate_up,LOW);
        digitalWrite(rotate_down,HIGH);
      } else {
        if (debug_mode) { Serial.println(F("DEACTIVATE")); }
        digitalWrite(rotate_down,LOW);
      }
      break;
     #endif //FEATURE_ELEVATION_CONTROL
  }
}
```