This documentation has been depricated.

Go to

https://github.com/k3ng/k3ng_rotator_controller/wiki

for current documentation

Radio Artisan

Arduino Rotator Controller

Documentation

Version 2.0

Updated 2014-07-08

Contributions from Eric, WB6KCN

Table of Contents

Introduction

The Radio Artisan Arduino Rotator Controller interfaces a computer to a rotator or a commercial rotator controller, emulating the Yaesu GS-232A/B and Easycom protocols which are supported by a myriad of logging, contest, and control programs. It can be easily interfaced with commercial rotator control units. With the addition of a proper capacity power supply and several interface components such as relays, this unit can also serve as a total replacement for a rotator control unit or serve as the basis for a 100% homebrew rotation system. Several azimuth and elevation position sensors including potentiometers, rotary encoders, and I2C devices are supported. The code is very flexible, modular, and easy to read allowing intermediate and advanced experimenters and builders to customize it.

Features

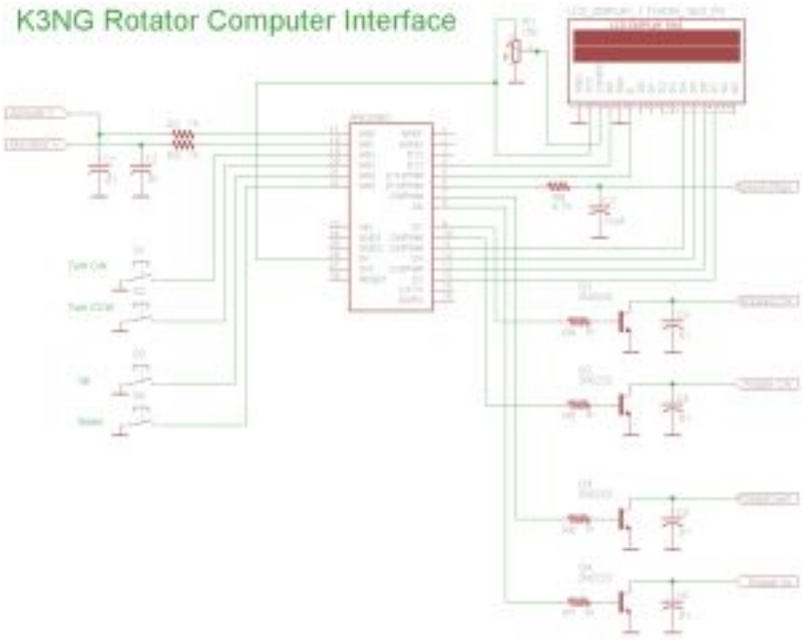
- Azimuth only and azimuth / elevation rotator support
- Serial interface using the standard Arduino USB port
- Control Port Protocol Support:
    - Yaesu GS-232A
    - Yaesu GS-232B
    - Easycom
- Support for position sensors:
    - Potentiometers / Analog Voltage
    - Rotary Encoders
    - Pulse Ouput
    - HMC5883L digital compass
    - ADXL345 accelerometer
    - LSM303 digital compass and accelerometer support under development
- LCD display (2 x 20 preferred, but any LCD over 15 columns supported)
- Can be interfaced with non-Yaesu rotators, including homebrew systems
- Directional indication on LCD display (North, South, North Northwest, etc.) along with degrees

- Intelligent automatic rotation (utilizes overlap on 450 degree rotators)
- Support for both 360 degree and 450 degree azimuth rotators or any rotation capability up to 719 degrees
- North Center and South Center support
- Support for any starting point (fully clockwise)
- Optional automatic azimuthal rotation slowdown feature when reaching target azimuth
- Optional rotation slow start (now with smooth ramp up)
- Optional brake engage/disengage lines for azimuth and elevation
- Buttons for manual rotation
- Command timeout
- Timed interval rotation
- Overlap LED Indicator
- Help screen
- Speed Control, both single PWM output (compatible with Yaesu controllers) and dual PWM rotate CW and rotate CCW outputs and dual elevate up and elevate down outputs
- Variable frequency outputs
- Preset Control using either potentiometers or rotary encoders with optional preset start button
- Speed Potentiometer
- Manual Rotation Limits
- Classic 4 bit, Adafruit I2C LCD, and Yourduino.com Display Support
- Optional tenth of a degree support with Easycom protocol (i.e. 123.4 degrees)
- Park button
- Azimuth and elevation calibration tables
- Host unit and Remote unit operation for remotely located sensors using two Arduinos or ATMega chips
- Works with hamlib rotctl/rotcltd
- Moon and sun tracking
- GPS Synchronization
- Realtime Clock Module Support
- Ethernet Shield Support

This unit can be interfaced with Yaesu rotator controller using the standard 6 pin DIN port on the back. Older controllers may not have the DIN interface and will require opening and soldered connections. This unit does not disable the rotator controller and all controls on it can continue to be used (although it is advisable not to attempt to control the rotator with manual controls and with the interface simultaneously.) Other rotator controllers can be interfaced as well with a little research and design.

Hardware

Schematic



Note: Refer to the pin numbers on the inside of the Arduino component outline (i.e. D1, D2, A0, A1, etc.) and not the numbers on the outside of it (1, 2, 3...).

The schematic above shows a basic setup supporting a 4 bit interface LCD, azimuth and elevation voltage inputs, manual buttons, and a single variable speed voltage output. Additional controls and options such as an azimuth present potentiometer, azimuth and elevation preset rotary encoders, and

brake control lines can be added as needed if the appropriate pins are chosen and set in code at compilation time. If using with an azimuth-only rotator, all components related to elevation can be omitted. So, again, keep in mind the above schematic is a basic unit that will work for most applications and additional features will require additional components.

This unit has been successfully interfaced to a Yaesu GS-1000DXA control unit, other various Yaesu G series control units, various other non-Yaesu rotator control units.

Arduino / AVR Hardware

For an azimuth-only system or an azimuth/elevation system with minimal features, an Arduino Uno, Arduino Nano, or similar variants, or an ATMega328 chip will have enough memory. The code has been tested on an Arduino Mega and Mega ADK. If you plan to use the master/slave functionality, you will need an Arduino Mega or Mega ADK, or an ATMega1284 or ATMMega2560 chip in order to have a second serial port. The remote slave unit can run on an Uno or an ATMega328 chip.

Theory of Operation

The rotator controller reads the azimuth (and elevation if optioned) from a sensor and reports the headings via a serial interface called the control port, when queried. The controller rotates the azimuth and elevation when commanded via the control port. The controller can be commanded for manual rotation (left, right, up, and down) or be commanded to rotate to a target azimuth and elevation. An LCD display can be optioned to display headings and various status messages. Human interface controls such as buttons, potentiometers, and rotary encoders can be added to control rotation. The control port can also be accessed via an Ethernet port using TCP/IP.

Motors and Motor control

The rotor software accommodates several different motor types each requiring unique interface hardware depending on motor type. Capacitor run single phase AC (typical CDE rotor), permanent

magnet DC (has brushes), synchronous AC motor and stepping motors. The motor controller must be selected to match the motor type. For a PMDC motor either relays or a dual half-bridge PWM controller (search ebay for BTS7960) is suitable. For Capacitor run single phase AC relays are usual. With relays you need a separate speed controller if speed control is desired. With a typical electronic motor control PWM speed control is built in. The controller has variable frequency outputs suitable for driving a synchronous AC motor but for a driver you are on your own. Step motors are typically 2/4 phase and require suitable micro stepping driver such as the DIV268N (search ebay)

In the K3NG rotor controller all motors are operated in a simple servo loop with the motor speed and stopping point determined by feedback from a positioning device. This is usual in all the above cases with the exception of step motors. In the K3NG application the step motor us used as a brushless DC motor (BLDC) which has maximum torque at zero RPM (stall). Most DC motors that have PWM speed control have near zero torque at very low motor RPM.

Software

Selecting Code Features

All features and options are configured in the *Features and Options* section of the code **in the rotator_features.h file**. Various #defines need to be enabled by uncommenting them (by removing double slashes "//") or disabled by commenting them out (prefixing with double slashes "//"). After the appropriate features and options are configured, compile the code, upload to your Arduino, and test.

Configuring Arduino / AVR Pins

All interface pins are defined in the *Pin Definitions* section of the code **in the rotator_pins.h file** . Many pins offer optional functionality which can be left disabled by leaving the pin defined with a zero (0) value. Pin settings and functionality are described in more detail below.

Computer Interface

The unit is interfaced to the computer simply using the native Arduino USB port. Currently Yaesu and Easycom protocols are supported.

To enable Yaesu GS-232A emulation, uncomment this line:

```
#define FEATURE_YAESU_EMULATION
```

To activate Yaesu GS-232B emulation, also uncomment this line:

```
#define OPTION_GS_232B_EMULATION
```

To activate Easycom emulation, uncomment just this line:

```
#define FEATURE_EASYCOM_EMULATION
```

Further information specific to the protocols is below.

Azimuth and Elevation Rotators

To activate the elevation functionality, uncomment this line in the code and recompile:

```
#define FEATURE_ELEVATION_CONTROL
```

If you are using this unit for an azimuth-only rotator, you can omit the following components: R6, Q3, C5, R7, Q4, C6, C2, R3, S3, and S4.

(Throughout this documentation you will often see elevation mentioned. If you have an azimuth-only rotator, rest assured that all functionality is supported for azimuth-only rotators and the code works equally well for azimuth-only and azimuth/elevation rotators.)

Commercial Hardware and Kits / Predefined Hardware Configurations

The software has some predefined hardware configuration options that conveniently map pins and enable features for specific pieces of hardware. These pre-configurations enabled in the rotator_hardware.h file.

M0UPU Board

Uncomment `#define HARDWARE_M0UPU` in the rotator_hardware.h file.

EA4TX

Uncomment `#define HARDWARE_EA4TX_ARS_USB` in the rotator_hardware.h file.

Position Sensors

Currently several types of position sensors are supported:

- Potentiometer / Analog Voltage Inputs
- Rotary Encoder
- Pulse
- ADXL345 Accelerometer
- HMC5883L digital compass
- LSM303 digital compass and accelerometer

You can mix and match azimuth and elevation sensors. For example, you can use a potentiometer for the azimuth and an ADXL345 accelerometer for the elevation. Aren't choices great?

Potentiometers / Analog Voltage Inputs

Most installations will use potentiometers as these are used in all mainstream commercial rotators (including Yaesu) and is often the easiest solution to implement in homebrew rotators. If you are using a commercial rotator and are interfacing with a rotator control unit that outputs an analog voltage for the azimuth (like the Yaesu G Series), this is the option you want to use. Potentiometer position sensors are activated with these lines:

```
#define FEATURE_AZ_POSITION_POTENTIOMETER
```

```
#define FEATURE_EL_POSITION_POTENTIOMETER
```

The input pins for potentiometer / voltage sensors are defined here:

```
#define rotator_analog_az A0

#define rotator_analog_el 0
```

You must use an analog capable pin such as A0, A1, A2, etc. The input expects 0 to 5 volts, with 0 volts being fully counterclockwise (CCW) and 5 volts fully clockwise (CW). (For elevation, 0 volts equates to 0 degrees, 5 volts equals 180 degrees or any maximum elevation you choose.) The Yaesu G series rotator controllers output this voltage.

Note that if you have an azimuth only rotator, you do not need to define anything associated with elevation, and disabling FEATURE_ELEVATION_CONTROL will cause all elevation related parameters and features to be ignored at compile and run time.

To activate use of the Arduino/AVR analog reference pin for any analog readings (i.e. azimuth and elevation potentiometer sensors), activate `OPTION_EXTERNAL_ANALOG_REFERENCE`. This can be used to increase the accuracy of readings by tieing the analog reference pin to the +5 volt line.

ADXL345 Accelerometer

The ADXL345 accelerometer can be used for elevation readings. Two libraries are supported, the Adafruit library and the Love Electronics library. Pick one or the other, but not both. Either one works fine.

To use the Adafruit library, uncomment this line:

```
#define FEATURE_EL_POSITION_ADXL345_USING_ADAFRUIT_LIB
```

To use the Love Electronics library, uncomment this line:

```
#define FEATURE_EL_POSITION_ADXL345_USING_LOVE_ELECTRON_LIB
```

The ADXL345 device is interfaced with the Arduino or ATMega chip via the I2C bus. Consult this for

more information on I2C.

Note that the accelerometer responds by rotating the sensor about the X axis.

Rotary Encoders

Rotary encoders can be enabled with these feature defines:

```
#define FEATURE_AZ_POSITION_ROTARY_ENCODER

#define FEATURE_EL_POSITION_ROTARY_ENCODER
```

Additionally, you need to configure the rotary encoder pins here:

```
#define az_rotary_position_pin1 0

#define az_rotary_position_pin2 0

#define el_rotary_position_pin1 0

#define el_rotary_position_pin2 0
```

Replace the zeros with your pin numbers. There are no restrictions on the pin assignments.

The last thing you need to do is set the amount of azimuthal or elevation heading change per rotary encoder pulse:

```
#define AZ_POSITION_ROTARY_ENCODER_DEG_PER_PULSE 0.5

#define EL_POSITION_ROTARY_ENCODER_DEG_PER_PULSE 0.5
```

Note that these options related to rotary encoder position sensors are enabled by default in the code:

```
#define OPTION_AZ_POSITION_ROTARY_ENCODER_HARD_LIMIT

#define OPTION_EL_POSITION_ROTARY_ENCODER_HARD_LIMIT
```

Pulse Inputs

Pulse inputs operate by increasing or decreasing the azimuth and/or elevation headings by a preset amount each time a pulse arrives. To enable this, first enable the appropriate feature defines:

```
#define FEATURE_AZ_POSITION_PULSE_INPUT

#define FEATURE_EL_POSITION_PULSE_INPUT
```

Then configure the pulse input pins. Note that these pins must be interrupt capable pins. (Pins 2 and 3 are interrupt capable on an Arduino Uno.)

```
#define az_position_pulse_pin 0

#define el_position_pulse_pin 0
```

And then set the appropriate interrupts for the pins you configured above

```
#define AZ_POSITION_PULSE_PIN_INTERRUPT 0

#define EL_POSITION_PULSE_PIN_INTERRUPT 0
```

On an Arduino Uno pin 2 uses interrupt 0 and pin 3 uses interrupt 1. Consult http://arduino.cc/en/Reference/AttachInterrupt for details on hardware and interrupts.

Last, set the amount of azimuth or elevation heading change in degrees:

```
#define AZ_POSITION_PULSE_DEG_PER_PULSE 0.5

#define EL_POSITION_PULSE_DEG_PER_PULSE 0.5
```

You may also be interested in these options. They are activated by default in the code:

```
#define OPTION_AZ_POSITION_PULSE_HARD_LIMIT // stop azimuth at
lower and upper limit rather than rolling over

#define OPTION_EL_POSITION_PULSE_HARD_LIMIT // stop elevation
```

```
at lower and upper limits rather than rolling over

#define OPTION_POSITION_PULSE_INPUT_PULLUPS
```

HMC5883L Digital Compass

The HMC5883L digital compass is supported using the Love Electronics library. Enable using this feature define:

```
#define FEATURE_AZ_POSITION_HMC5883L
```

The sensor is connected using the I2C bus.

Incremental Rotary Encoders

Incremental rotary shaft encoders are capable of the highest stability and accuracy, considerably better than 0.1 degree. The usual disadvantage of incremental encoders is that the position value accumulated is volatile with loss of power. K3NG software eliminates this disadvantage by maintaining the headings and recalibrating the bearing during normal rotor use. A good affordable encoder is the Omron

E6B2-CWZ6C 2000P. There are many different SE configurations available, most of which are useable if accommodation is made to the Arduino interface. The following text only applies to the Omron E6B2.There are two quadrature outputs A & B and an index pulse called Z. All three outputs are open-collector NPN transistors. The power required is a voltage between +5 and +24, the Arduino Vin voltage is a convenient source. The K3NG code provides for internal pullups which will work just fine on the bench. If the distance between the SE and the Arduino is significant discrete pullups (1K ~ 5K to 5 volts) at the Arduino should be provided. A and B are 50 % duty cycle squarewaves phased 90 degrees apart. This SE outputs 2000 A/B counts per revolution. Z is a narrow pulse that occurs once per SE revolution. Direction of rotation indication can be had by reversing the A and B SE outputs to the Arduino.

Configuration

```
#define FEATURE_AZ_POSITION_INCREMENTAL_ENCODER

#define FEATURE_EL_POSITION_INCREMENTAL_ENCODER



#define OPTION_INCREMENTAL_ENCODER_PULLUPS



#define az_incremental_encoder_pin_phase_a 2 // must be an
interrupt capable pin

#define az_incremental_encoder_pin_phase_b 3 // must be an
interrupt capable pin

#define az_incremental_encoder_pin_phase_z 4



#define AZ_POSITION_INCREMENTAL_ENCODER_A_PIN_INTERRUPT 0
// Uno: pin 2 = interrupt 0, pin 3 = interrupt 1

#define AZ_POSITION_INCREMENTAL_ENCODER_B_PIN_INTERRUPT 1
// Uno: pin 2 = interrupt 0, pin 3 = interrupt 1


// read http://arduino.cc/en/Reference/AttachInterrupt for
```

```
details on hardware and interrupts




#define el_incremental_encoder_pin_phase_a 2 // must be an
interrupt capable pin

#define el_incremental_encoder_pin_phase_b 3 // must be an
interrupt capable pin

#define el_incremental_encoder_pin_phase_z 4



#define EL_POSITION_INCREMENTAL_ENCODER_A_PIN_INTERRUPT 0
// Uno: pin 2 = interrupt 0, pin 3 = interrupt 1

#define EL_POSITION_INCREMENTAL_ENCODER_B_PIN_INTERRUPT 1



#define AZ_POSITION_INCREMENTAL_ENCODER_PULSES_PER_REV 8000.0

#define EL_POSITION_INCREMENTAL_ENCODER_PULSES_PER_REV 8000.0
```

Debugging

{under construction}

HH-12 / AS5045

{under construction}

Feature Activation

```
#define FEATURE_AZ_POSITION_HH12_AS5045_SSI

#define FEATURE_EL_POSITION_HH12_AS5045_SSI
```

Pin Definitions

```
#define az_hh12_clock_pin 11

#define az_hh12_cs_pin 12

#define az_hh12_data_pin 13


#define el_hh12_clock_pin 11

#define el_hh12_cs_pin 12

#define el_hh12_data_pin 13
```

Hardware Connection

LSM303

{under construction}

```
#define FEATURE_AZ_POSITION_LSM303

#define FEATURE_EL_POSITION_LSM303
```

Memsic 2125

Feature Activation

```
#define FEATURE_EL_POSITION_MEMSIC_2125
```

Pin Definitions

```
#define pin_memsic_2125_x 0

#define pin_memsic_2125_y 0
```

Human Interface

Languages

Language customization is supported for the LCD display. Currently these languages are available:

```
#define LANGUAGE_ENGLISH

#define LANGUAGE_SPANISH
```

Language customization support may be added in the future for the command line interface. We're looking for volunteers to create more translations!

LCD Display

Standard 4 Bit LCD Interface

To use a common classic 4 bit LCD display unit, uncomment out this line (remove the double-slashes "//") to enable it:

```
#define FEATURE_4_BIT_LCD_DISPLAY
```

The LiquidCrystal declaration specifies the pins to be used for interfacing. Consult the Arduino LiquidCrystal library documentation for details.

A 2 row by 20 column LCD display is preferred, but a 16 column can be used. Set the number of columns of your display in this line:

```
#define LCD_COLUMNS 20
```

Adafruit I2C LCD

To use an Adafruit I2C LCD unit, uncomment this line:

```
#define FEATURE_ADAFRUIT_I2C_LCD
```

The I2C are hard coded for whatever Arduino board you are using and are not set in the code. On an Uno, the I2C pins are A4 and A5.

Yourduino LCD Display

To use the YourDuino.com I2C display, configure these pins in the rotsator_pins.h file:

```
#ifdef FEATURE_YOURDUINO_I2C_LCD
#define En_pin  2
#define Rw_pin  1
#define Rs_pin  0
#define D4_pin  4
#define D5_pin  5
```

```
#define D6_pin  6

#define D7_pin  7

#endif //FEATURE_YOURDUINO_I2C_LCD
```

...and uncomment this feature define:

```
#define FEATURE_YOURDUINO_I2C_LCD
```

DFRobot LCD Display

To use a DFRobot LCD display, uncomment this line:

```
#define FEATURE_RFROBOT_I2C_DISPLAY
```

Settings can be customized in the rotator_settings.h file:

```
#ifdef FEATURE_RFROBOT_I2C_DISPLAY

LiquidCrystal_I2C lcd(0x27,16,2);

#endif //FEATURE_RFROBOT_I2C_DISPLAY
```

LCD Customization

The LCD display can be customized with the activation of various “widgets”. The first row of the LCD, herein reffered to as “row 1”, is reserved for status messages when an event such as rotation is in progress, but row 1 can also accommodate some widgets.

Widgets are activated by uncommenting various features. Each widget has customization settings in rotator_settings. h that may change the alignment and/or the row on which it appears. Here are the widgets available:

OPTION_DISPLAY_HHMM_CLOCK

Format: HH:MM clock on row 1

Alignment Configuration: `LCD_HHMM_CLOCK_POSITION` (LEFT or RIGHT)

OPTION_DISPLAY_HHMMSS_CLOCK

Format: HH:MM:SS clock on row 1

Alignment Configuration: `LCD_HHMMSS_CLOCK_POSITION` (LEFT or RIGHT)

OPTION_DISPLAY_ALT_HHMM_CLOCK_AND_MAIDENHEAD

Format: Alternating HH:MM Clock and Maidenhead

Row: `LCD_ALT_HHMM_CLOCK_AND_MAIDENHEAD_ROW`

Alignment: `LCD_ALT_HHMM_CLOCK_AND_MAIDENHEAD_POSITION`

OPTION_DISPLAY_CONSTANT_HHMMSS_CLOCK_AND_MAIDENHEAD

Format: Continual HH:MM:SS Clock and Maidenhead

Row: `LCD_ALT_HHMMSS_CLOCK_AND_MAIDENHEAD_ROW`

Alignment: `LCD_ALT_HHMMSS_CLOCK_AND_MAIDENHEAD_POSITION`

OPTION_DISPLAY_BIG_CLOCK

Format: YYYY-MM-DD HH:MM:SSZ

Row: `LCD_BIG_CLOCK_ROW`

Alignment: Always centered

OPTION_DISPLAY_GPS_INDICATOR

Format: “GPS” when GPS tracking is active

Row: `LCD_GPS_INDICATOR_ROW`

Alignment: LCD_GPS_INDICATOR_POSITION

OPTION_DISPLAY_MOON_TRACKING_CONTINUOUSLY

Format: Current moon azimuth and elevation

Row: LCD_MOON_TRACKING_ROW

Alignment: Always centered

OPTION_DISPLAY_DIRECTION_STATUS

Format: Displays current direction (N, S, E, W, etc.)

Row: Always Row 1

Alignment: Always centered

OPTION_DISPLAY_SUN_TRACKING_CONTINUOUSLY

Format: Current sun azimuth and elevation

Row: LCD_SUN_TRACKING_ROW

Alignment: Always centered

OPTION_DISPLAY_MOON_OR_SUN_TRACKING_CONDITIONAL

Format: Displays sun or moon tracking when it is activate. A "*" appears on each side during AOS and a "-" appears on each side during LOS.

Row: LCD_MOON_OR_SUN_TRACKING_CONDITIONAL_ROW

Alignment: Always centered

Customization:

```
#define TRACKING_ACTIVE_CHAR "*"
```

```
#define TRACKING_INACTIVE_CHAR "-"
```

OPTION_CLOCK_ALWAYS_HAVE_HOUR_LEADING_ZERO

This option activates the hour leading zero on several of the different clock widgets. For example, 1:23 will be display as 01:23.

Buttons

Button Switch Interfacing

All button switches are momentary normally off switches. When the button is pressed, the switch is closed which forces the Arduino interface pin to 0 volts or a logic low level.

Manual Rotation

Buttons are provided for manual rotation control if desired. These buttons can be omitted without any code changes if manual button rotation control is not desired. When not using a button, set its pin to 0 (zero) in the code prior to compilation:

```
#define button_cw 0

#define button_ccw 0

#define button_up 0

#define button_down 0
```

There is an optional stop button feature that can be enabled by assigning a pin number to this line:

```
#define button_stop 0
```

The stop button is mainly for halting automatic rotation initiated by the computer serial interface and not for stopping manual rotating activated by rotate CW and rotate CCW button depressing.

Park Function

The Park feature enables the user to press a button and have the rotator rotate to a preset azimuth (and elevation), usually to safeguard antennas from high winds or tuck away antennas on mobile arrays for travel. The Park function is enabled by activating FEATURE_PARK and defining a pin for the button here:

```
#define button_park 0
```

The preset azimuth and/or elevation is set here:

```
#define PARK_AZIMUTH 0.0

#define PARK_ELEVATION 0.0
```

Park can also be initiated with the \P command using the serial control port or Ethernet interface.

Controls

Preset Controls

A present control lets the user dial a desired azimuth or elevation and initiate rotation. Two types of preset controls are supported, potentiometers and rotary encoders. A present start button can be defined for either.

Preset Potentiometer

A preset potentiometer can be added by defining an analog pin in this line:

```
#define az_preset_pot 0
```

As of this writing the potentiometer is not on the schematic, however it's easy to connect. Merely take a 1K to 100K potentiometer, connect one end to ground, the other end to +5 volts, and the center potentiometer wiper pin to the analog Arduino pin defined in the line above. For good measure you can add a 0.01 uF capacitor from the +5V potentiometer to the ground pin.

The unit will automatically rotate a half second after the potentiometer is rotated. If you would prefer a start button to initiate rotation rather than having it occur automatically, define an input pin in this line:

```
#define az_preset_start_button 0
```

The button is not on the schematic, but it's simply a momentary normally open button that grounds the pin upon closure. No pull resistor is required. Upon pressing the button, the unit will initiate rotation based on the position of the potentiometer.

Both pin settings are defaulted in the code to 0 (zero) which disables them and the preset potentiometer functionality. (Do not enable this feature unless you have a potentiometer connected, otherwise stray noise on the analog pin will trigger false rotations.)

The settings and limits for the potentiometer can be set here:

```
#define AZ_PRESET_POT_FULL_CW 0

#define AZ_PRESET_POT_FULL_CCW 1023

#define AZ_PRESET_POT_FULL_CW_MAP 180

#define AZ_PRESET_POT_FULL_CCW_MAP 630
```

The default settings in the code are for a rotator with a starting azimuth of 180 degrees and 450 degrees of rotation capability. A rotator with a starting point of 0 degrees and 360 degree rotation capability would be set up as so:

```
#define AZ_PRESET_POT_FULL_CW 0

#define AZ_PRESET_POT_FULL_CCW 1023

#define AZ_PRESET_POT_FULL_CW_MAP 0

#define AZ_PRESET_POT_FULL_CCW_MAP 360
```

The preset potentiometer is currently implemented only for the azimuth, not elevation. Post on the Radio Artisan group if you're interested in an elevation preset potentiometer, or you can use the rotary encoder preset control which is implemented for both azimuth and elevation.

Preset Rotary Encoders

Preset controls using rotary encoders are supported for both azimuth and elevation. To enable, uncomment the appropriate lines:

```
#define FEATURE_AZ_ENCODER_SUPPORT

#define FEATURE_EL_ENCODER_SUPPORT
```

Note that `FEATURE_EL_ENCODER_SUPPORT` requires `FEATURE_AZ_ENCODER_SUPPORT`.

Pins for the controls are defined here:

```
#define az_rotary_pin1 0

#define az_rotary_pin2 0

#define el_rotary_pin1 0

#define el_rotary_pin2 0
```

The center pin of the rotary control should be grounded. No pull up resistors on CW and CCW pins of the rotary encoder are needed as internal Arduino pullups are enabled with this line:

```
#define OPTION_ENCODER_ENABLE_PULLUPS
```

By default the rotary encoders have "absolute" values that are initiated with the current azimuth (and elevation) at start up. If the azimuth or elevation is changed through other means such as commands or the manual rotation buttons, the rotary encoder presets maintain their values. If you would like the encoder to act in a "relative" fashion, enable this line:

```
#define OPTION_ENCODER_RELATIVE_CHANGE
```

This will cause the rotary encoder preset controls to decrease or increase the azimuth or elevation

relative to the current azimuth or elevation. If the unit is rotating or elevating to a heading when a rotary encoder preset is invoked, the current target azimuth and/or elevation is used and adjusted.

Speed Potentiometer

A speed potentiometer can be added by defining an analog pin in this line:

```
#define speed_pot 0
```

As of this writing the potentiometer is not on the schematic, however you can use a 1K to 100K potentiometer, connect one end to ground, the other end to +5 volts, and the center or wiper potentiometer pin to the analog Arduino pin defined in the line above. You may want to add a 0.01 uF capacitor from the +5V potentiometer to the ground pin.

The setting for this pin in the code is defaulted to 0 (zero) which disables the pin. As with all analog input pins, don't enable this feature unless you have a potentiometer connected, otherwise noise on the pin will cause false readings.

The mappings for the potentiometer can be set here:

```
#define SPEED_POT_LOW 0

#define SPEED_POT_HIGH 1023

#define SPEED_POT_LOW_MAP 1

#define SPEED_POT_HIGH_MAP 255
```

Joystick

{under construction}

```
#define FEATURE_JOYSTICK_CONTROL
```

```
#define pin_joystick_x A0
#define pin_joystick_y A1

#define JOYSTICK_WAIT_TIME_MS 100


//#define OPTION_JOYSTICK_REVERSE_X_AXIS
//#define OPTION_JOYSTICK_REVERSE_Y_AXIS
```

Indicators

Overlap LED

To activate the Overlap LED line, change the 0 (zero) in this line to whatever pin you wish to use:

```
#define overlap_led 0
```

To disable, set the pin to 0. The Overlap LED will be activated whenever the azimuth is greater than the rotator starting (fully CCW) azimuth.

Park LED

This pin goes high when the rotation system has been parked.

```
#define parked_pin 0
```

Park In Progress LED

This pin will go high when there is a rotation park operation in progress.

```
#define park_in_progress_pin 0
```

Serial Activity LED

If defined this pin will go high when there is serial control port activity.

```
#define serial_led 0
```

Run LED

This pin will alternate high and low when the system is running and can be used to drive an LED. This can be useful on headless units that lack an LCD display or serial port control, or remote slave units.

```
#define blink_led 0
```

GPS Sync LED

This pin when defined will go high when GPS tracking is enabled and there is a valid satellite fix.

```
#define gps_sync 0
```

Moon Tracking LED

This pin goes high when moon tracking is activated (regardless of moon visibility).

```
#define moon_tracking_active_pin 0
```

Sun Tracking LED

This pin goes high when sun tracking is activated (regardless of sun visibility).

```
#define sun_tracking_active_pin 13
```

Heading Analog Output Pins

Two pins are available to driving analog meters for azimuth and elevation headings. This is activated with:

```
#define FEATURE_ANALOG_OUTPUT_PINS
```

The pins are defined in this section of code:

```
#define pin_analog_az_out 0

#define pin_analog_el_out 0
```

Additionally, the corresponding elevation voltage maximum can be set in the rotator_setting.h file:

```
#define ANALOG_OUTPUT_MAX_EL_DEGREES 180
```

Control Port Interface

{under construction}

Rotate to Long Path

If you would like to automagically rotate to long path, the \L command is your friend.

Power Switch

If you would like to have the unit control its power, there is a pin which will go low after a set amount of time. The pin is defined here:

```
#define power_switch 0
```

The inactivity time is configured here (the unit is minutes):

```
#define POWER_SWITCH_IDLE_TIMEOUT 15
```

And the feature is enabled with this line:

```
#define FEATURE_POWER_SWITCH
```

Heading Calibration

Yaesu GS-232 Emulation

On the serial interface, issue the O command and manually rotate the rotator to full counter-clockwise 180 degrees and send a carriage return. Then issue the F command and manually rotate the rotator to full clockwise (270 degrees on a 450 degree rotator, or 180 degrees on a 360 degree rotator) and send a carriage return.

The elevation can be calibrated similarly with the O2 and F2 commands at 0 degrees and 180 degrees, respectively.

The calibration settings are written to non-volatile EEPROM memory.

Easycom Emulation

The Easycom protocol does not have calibration commands like the Yaesu protocol, therefore you must manually configure these settings at compile time:

```
#define ANALOG_AZ_FULL_CCW 4

#define ANALOG_AZ_FULL_CW 1009

#define ANALOG_EL_0_DEGREES 2

#define ANALOG_EL_MAX_ELEVATION 1018
```

Another option is to compile with Yaesu GS-232 support, calibrate the unit, then recompile the code with Easycom support and upload it.

Calibration Tables

If you have an azimuth or elevation sensor that is not linear and requires a calibration lookup table, that is provided in the code here:

```
#define AZIMUTH_CALIBRATION_FROM_ARRAY {180,630}

#define AZIMUTH_CALIBRATION_TO_ARRAY {180,630}

#define ELEVATION_CALIBRATION_FROM_ARRAY {-180,0,180}

#define ELEVATION_CALIBRATION_TO_ARRAY {-180,0,180}
```

To enable it, activate `FEATURE_AZIMUTH_CORRECTION` and/or `FEATURE_ELEVATION_CORRECTION` and configure the arrays above. The *from* array is for the values coming from the sensor, and the *to* values are what you want to translate the values to. Values that are between from values are extrapolated so there is no need to fill the arrays with every possible value, just end points and a reasonable number of points in between, depending on how nonlinear your sensor is and how much accuracy is needed. Note that you must have an equal number of members in each from and to array. The azimuth and elevation correction features apply to any sensor when activated.

Manual Heading Calibration

The azimuth and elevation can be manually calibrated using the \A and \B commands, respectively. For example, to calibrate the azimuth:

```
\A175.3
```

This feature is not intended to be a replacement for calibration using the Yaesu GS-232 emulation O and F commands for the potentiometer sensors, but is intended for tweaking of calibration during a session. For sensors like rotary encoders without a Z pulse and pulse inputs where the position is not absolute (not deterministically known at power up), the \A and \B commands may often needed to maintain accuracy.

Heading Calibration for Incremental Shaft Encoders with Z Pulse

Recalibration of the system will occur automatically every time the rotor bearing passes thru the Z pulse bearing. The Z pulse bearing should be placed mechanically at an azimuth bearing that the rotor will frequent, usually in the center of the range of motion, like due North. Similarly place the elevation Z pulse at a bearing that is conveniently measured, like 45 or 90 degrees.

Heading Calibration Using Sun or Moon Position

The azimuth and elevation can be calibrated to the current sun or moon position using the \XS and \XM commands, if the sun or moon is visible. This feature is intended to be used by manually rotating an antenna towards the sun or moon and peaking for maximum solar noise or EME signals, and then executing the appropriate command.

The rotator controller clock must be accurately set to Zulu time for this feature to work accurately.

Heading Sampling / Averaging

There are several settings to help you tweak the behavior of the heading sample.

```
#define AZIMUTH_SMOOTHING_FACTOR 0

#define ELEVATION_SMOOTHING_FACTOR 0
```

The smoothing factor settings make the unit average a fraction of the newly sampled heading with the current heading. These factors can be set from 0 to 99.9. The higher the number, the higher percentage of the current heading is used in the averaging formula.

```
#define AZIMUTH_MEASUREMENT_FREQUENCY_MS 100

#define ELEVATION_MEASUREMENT_FREQUENCY_MS 100
```

The measurement frequency settings determine how often headings are sampled. The unit is in

milliseconds (mS).

Interface Protocols Options and Tweaking

Yaesu GS-232 Emulation

Yaesu GS-232 Emulation and Ham Radio Deluxe

Uncomment the following lines to make the interface in Yaesu mode play better with HRD:

```
#define OPTION_C_COMMAND_SENDS_AZ_AND_EL

#define OPTION_DELAY_C_CMD_OUTPUT
```

Note: At the time of this writing, "Yaesu GS-232B Az/El" mode in the free version HRD Rotator program has a bug in which it will not properly parse and display the elevation. Use "Yaesu GS-232A Az/El" mode instead.

Yaesu GS-232B Emulation

When activating `FEATURE_YAESU_EMULATION` the code defaults to GS-232A emulation, however also activating `OPTION_GS_232B_EMULATION` will default the interface protocol to GS-232B emulation. This emulation adds support for the following commands:

Z - toggle north and south center mode

P36 - switch to 360 degree mode

P45 - switch to 450 degree mode

Additionally, the output of the B, C, and C2 commands is slightly different than in GS-232A emulation.

Yaesu Timed Buffer Commands

To enable Yaesu timed buffer commands, uncomment this line:

```
#define FEATURE_TIMED_BUFFER
```

Disabling this feature will free up some memory for other features.

Easycom Protocol

Surprisingly, the Easycom protocol does not offer a means of querying the azimuth or elevation from a controller. However, the code contains two non-standard extensions to the protocol that can be activated with these lines:

```
#define OPTION_EASYCOM_AZ_QUERY_COMMAND

#define OPTION_EASYCOM_EL_QUERY_COMMAND
```

With these options, the AZ and EL commands with no parameters will return the current azimuth or elevation.

Tenth of a Degree Resolution

Higher precision headings are activated with this line:

```
#define FEATURE_ONE_DECIMAL_PLACE_HEADINGS
```

The Easycom protocol natively supports tenth of a degree resolution (i.e. 123.4 degrees), however the Yaesu protocol does not. If this feature is enabled, the Yaesu protocol emulation will still report azimuths and elevations as integers. With this feature enabled, the LCD display will show headings with the increased resolution.

Basic Rotation Control Pins

{under construction}

Azimuth

```
#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 rotate_cw_ccw  0         // goes high for both CW and CCW
rotation

#define rotate_cw_pwm 0          // optional - PWM CW output - set to
0 to disable (must be PWM capable pin)

#define rotate_ccw_pwm 0         // optional - PWM CCW output - set
to 0 to disable (must be PWM capable pin)

#define rotate_cw_ccw_pwm 0      // optional - PWM on CW and CCW
output - set to 0 to disable (must be PWM capable pin)

#define rotate_cw_freq 0         // optional - CW variable frequency
output

#define rotate_ccw_freq 0        // optional - CCW variable frequency
output

#define az_stepper_motor_pulse 0

#define az_stepper_motor_direction 0
```

Elevation

```
#define rotate_up 8               // goes high to activate rotator
elevation up

#define rotate_down 9             // goes high to activate rotator
elevation down

#define rotate_up_or_down 0       // goes high when elevation up or
down is activated
```

```
#define rotate_up_pwm 0           // optional - PWM UP output - set
to 0 to disable (must be PWM capable pin)

#define rotate_down_pwm 0         // optional - PWM DOWN output - set
to 0 to disable (must be PWM capable pin)

#define rotate_up_down_pwm 0      // optional - PWM on both UP and
DOWN (must be PWM capable pin)

#define rotate_up_freq 0          // optional - UP variable frequency
output

#define rotate_down_freq 0        // optional - UP variable frequency
output
#define el_stepper_motor_pulse 0

#define el_stepper_motor_direction 0
```

Rotator Customization

Configuration for Other Than 450 Degree Rotators (Maximum Clockwise)

To change the rotation capability, change this setting:

```
#define AZIMUTH_ROTATION_CAPABILITY_DEFAULT 450
```

The settings is the rotation capability in degrees. Rotations of up to 719 degrees are supported. Note that if Yaesu GS-232B emulation is enabled, the P36 and P45 commands will switch between 360 degree and 450 degree modes and will override this setting for the session, however when the unit is rebooted, the #define above will take effect again. (Prior to version 1.9, the P36 and P45 commands wrote to non-volatile EEPROM. On version 1.9 and later, they do not,)

The Easycom protocol does not have a means of changing this at runtime, however the #define above is still applicable.

Alternate Starting Points (Maximum Counterclockwise)

By default the code is configured for the azimuth rotation starting at 180 degrees or "south center" in Yaesu terminology. The default can be changed with this setting:

```
#define AZIMUTH_STARTING_POINT_DEFAULT 180
```

To default to "north center" or 0 degrees, change this setting to 0. Note that if Yaesu GS-232B emulation is enabled, the Z command will toggle between north (0 degree) and south (180 degree) center modes and will override the setting.

The Easycom protocol does not have a means of changing this at runtime, however the #define above is still applicable.

Support for Elevation Rotators That Don't Rotate (Elevate) 180 Degrees

Change this setting to alter the maximum elevation:

```
#define ELEVATION_MAXIMUM_DEGREES 180
```

Rotation Speed Control

The controller in its simplest configuration activates the rotate_cw and rotate_ccw pins, and in the case of an AZ/EL rotator also the rotate_up and rotate_down pins. This is sufficient for many installations, however larger antenna arrays present some physical and engineering challenges and require variable speed control and acceleration and deacceleration to avoid damage to components. This controller supports such needs.

Single "Always On" PWM Output / Yaesu Control Unit Interfacing

The Yaesu X1, X2, X3, and X4 speed commands are supported through a pulse-width modulation (PWM) pin connected to the Yaesu rotator controller, pin 3. If this functionality is not desired, simply connect a jumper between pins 3 and 6 on the Yaesu controller. This will send +5V into the rotation speed pin and will hardwire the unit for the fastest speed.

The antenna speed voltage pin can be changed by modifying this line:

```
#define azimuth_speed_voltage 10
```

To disable the pin, set it to 0 (zero). Note that PWM is supported only on a specific Arduino pins; check your documentation if you intend on using a different pin.

The PWM frequency is 490 Hz. An RC filter using a 4.7k resistor and 10 uF cap creates a 2 Hz filter which "smooths out" the squarewave and provides a nice voltage to the rotator controller.

Note that the Easycom protocol does not have a standard means of changing rotational speed.

Switched Multiple PWM Outputs

If you need PWM outputs on the rotate CW and CCW lines instead of just one separate speed voltage pin (such as for a homebrew rotator), PWM pins can be defined here (replace the zeros with your desired pins):

```
#define rotate_cw_pwm 0

#define rotate_ccw_pwm 0
```

Elevation PWM pins can be defined here:

```
#define rotate_up_pwm 0

#define rotate_down_pwm 0
```

As if that wasn't enough, we have more choices for you. If you would like switched PWM that comes on when azimuthal or elevation rotation is occurring, we got you covered with these pins:

```
#define rotate_cw_ccw_pwm 0

#define rotate_up_down_pwm 0
```

As with most other pin definitions, setting them to 0 (zero) disables them. Again, remember that not all Arduino pins support PWM and you must select PWM capable pins for the output pins above for PWM to work.

Switched Variable Frequency Outputs

If you need variable frequency outputs, pins for this can be defined here:

```
#define rotate_cw_freq 0

#define rotate_ccw_freq 0

#define rotate_up_freq 0

#define rotate_down_freq 0
```

The minimum and maximum speed frequencies are defined here:

```
#define AZ_VARIABLE_FREQ_OUTPUT_LOW 1 // Frequency in hertz of
```

```
minimum speed

#define AZ_VARIABLE_FREQ_OUTPUT_HIGH 50 // Frequency in hertz
of maximum speed

#define EL_VARIABLE_FREQ_OUTPUT_LOW 1 // Frequency in hertz of
minimum speed

#define EL_VARIABLE_FREQ_OUTPUT_HIGH 50 // Frequency in hertz
of maximum speed
```

Automatic Azimuth Rotation Slowdown

To enable this feature, changed the 0 (zero) in this line to a 1 (one):

```
#define AZ_SLOWDOWN_DEFAULT 0
```

This option will slow the rotation down when automatically rotating and the target azimuth is within 10 degrees. This can save some wear-and-tear on your rotator, especially with larger installations. The point at which automatic slowdown kicks in can be adjusted with this line:

```
#define SLOW_DOWN_BEFORE_TARGET_AZ 10
```

The unit is degrees.

Azimuth Rotation Slow Start

To enable this feature, changed the 0 (zero) in this line to a 1 (one):

```
#define AZ_SLOWSTART_DEFAULT 0
```

This feature starts the rotation at a slower speed and gradually ramps it up to the currently set default speed. The amount of time spent in slow start is configured with this setting:

```
#define AZ_SLOW_START_UP_TIME 2000
```

The time is in milliseconds.

Automatic Elevation Rotation Slowdown

To enable this feature, changed the 0 (zero) in this line to a 1 (one):

```
#define EL_SLOWDOWN_DEFAULT 0
```

This option will slow the rotation down when automatically rotating elevation and the target elevation is within 10 degrees. The point at which automatic slowdown kicks in can be adjusted with this line:

```
#define EL_SLOW_DOWN_BEFORE_TARGET_EL 10
```

The unit is degrees.

Elevation Rotation Slow Start

To enable this feature, changed the 0 (zero) in this line to a 1 (one):

```
#define EL_SLOWSTART_DEFAULT 0
```

The amount of time spent in slow start is configured with this setting:

```
#define EL_SLOW_START_UP_TIME 2000
```

The time is in milliseconds.

Tweaking Slow Start and Slow Down Behavior

There are various settings available for altering the operation of slow start and slow down. The operation of them is not covered here, however if you need more information, please post on the Radio Artisan group.

```
#define AZ_SLOW_START_STARTING_PWM 1 // PWM starting value for
slow start

#define AZ_SLOW_START_STEPS 20
```

```
#define AZ_SLOW_DOWN_PWM_START 200 // starting PWM value for
slow down

#define AZ_SLOW_DOWN_PWM_STOP 20 // ending PWM value for slow
down

#define AZ_SLOW_DOWN_STEPS 20

#define EL_SLOW_START_STARTING_PWM 1 // PWM starting value for
slow start

#define EL_SLOW_START_STEPS 20

#define EL_SLOW_DOWN_PWM_START 200 // starting PWM value for
slow down

#define EL_SLOW_DOWN_PWM_STOP 20 // ending PWM value for slow
down

#define EL_SLOW_DOWN_STEPS 20
```

Slow Start and Slow Down for Stepping Motors

Step motors can be intolerant of high accelerations. If the magnetic field rotation leads or lags the mechanical position of the step motor rotor excessively the motor may freeze in position. The lines in the section above allow the designer to deal with this problem.

Brake Operation

Two I/O pins can be defined for brake operation, one for azimuth and one for elevation:

```
#define brake_az 0

#define brake_el 0
```

A setting of zero (0) disables the line.

The brake engage delay time is configured with these settings:

```
#define AZ_BRAKE_DELAY 3000

#define EL_BRAKE_DELAY 3000
```

The delay time is in milliseconds. Each brake line goes high when rotation is in progress, so the lines can be used to engage a relay which would supply voltage to activate a solenoid and disengage a brake.

Rotation Limits

Software

If you would like to set software limits for the azimuth rotation commands and buttons uncomment this line:

```
#define OPTION_AZ_MANUAL_ROTATE_LIMITS
```

...and customize these settings:

```
#define AZ_MANUAL_ROTATE_CCW_LIMIT 185

#define AZ_MANUAL_ROTATE_CW_LIMIT 535
```

The above settings will stop CCW rotation when an azimuth of 185 is reached, and stop CW rotation when 175 degrees is reached ("5:30" on a 180 degree starting azimuth rotator.)

This functionality is useful if for some reason you do not want your rotation system going beyond a certain point due to physical limits.

There is a corresponding feature and settings for elevation rotation limits:

```
#define OPTION_EL_MANUAL_ROTATE_LIMITS
```

```
#define EL_MANUAL_ROTATE_DOWN_LIMIT -1

#define EL_MANUAL_ROTATE_UP_LIMIT 181
```

Hardware / Limit Switches

{under construction}

```
#define FEATURE_LIMIT_SENSE

#define az_limit_sense_pin 0  // input - low stops azimuthal
rotation

#define el_limit_sense_pin 0  // input - low stops elevation
rotation
```

Rotation Pin Logic State Inversion

Normally the rotate_cw, rotate_ccw, rotate_up, and rotate_down pins operate in an "inactive low / active high" manner. If you wish to invert this behavior, modify these lines, switching the LOW and HIGH values:

```
#define ROTATE_PIN_INACTIVE_VALUE LOW

#define ROTATE_PIN_ACTIVE_VALUE HIGH
```

Direct Rotator Interfacing

Rotators can basically be divided into two major groups, DC rotated and AC rotated. Most Yaesu G series rotators are DC based, usually with a voltage from 0 to 24 volts and the polarity of the DC voltage is switched to provide clockwise and counter-clockwise rotation. The G-5500 is one exception; it has AC operated motors.

Older rotators are most often AC operated. Either dual motor windings will be used or a single winding with a switch phase shift capacitor provides alternate direction rotation. Voltages are usually in the 12 to 30 VAC range, however never assume and always use a voltmeter to measure actual voltages. Also, be aware that capacitors such as the phase shift capacitor you find in control units (and any large capacitor for that matter) can hold a lethal charge and should be discharged before working on equipment.

Often the most challenging part of interfacing to an older rotator is the azimuth potentiometer. These are often low resistance values, such as 25 to 100 ohms, originally intended to drive mechanical (needle) current meters in the control unit, and may not play well with a 5 volt supply voltage. The Arduino rotator control unit with `FEATURE_AZ_POSITION_POTENTIOMETER` and/or `FEATURE_EL_POSITION_POTENTIOMETER` requires a 0 to 5 volt azimuth and elevation voltage range to provide the best accuracy and precision. Therefore, if using a low resistance potentiometer, an operation amplifier ("op amp") such as a 741 will be needed to transform the small voltage swing to a 0-5 volt swing.

If the rotator has a reostat (a variable resistor with two leads) or a potentiometer with a grounded wiper which essentially makes it a reostat, you can place a fixed resistor of equal value in series with the reostat and supply the fixed resistor with 10 volts to get the desired 5 volt swing.

Questions about interfacing to rotators should be posted on the Radio Artisan group, which is frequented by many radio amateurs who have built homebrew rotation systems or have interfaced to older rotators.

Master and Remote Slave Unit Operation

Introduction

I2C sensors offer some real neat options for sensing azimuth and elevation. These devices are commonly used in game controllers and mobile devices to sense position or movement of the device. Magnetometers or digital compasses measure azimuth and accelerometers can be used to sense the Earth's gravity and provide an elevation reading.

One issue facing the builder using these devices is the distance limitations of the I2C bus. I don't know the exact limit, but anecdotal evidence leads me to believe it is several meters, which poses a problem when attempting to use these devices on an antenna up a tower or in an array perhaps several hundred meters from the operating position.

But, never fear, others have thought of this issue and as usual we have a solution. The controller code allows you to compile code for a remote unit which interfaces with a host unit located inside. The remote unit interfaces with the position sensor devices locally at the rotator/antenna location. Any normally supported position sensor can be used with the remote unit, including potentiometers, rotary encoders, and I2C devices. The host unit talks to the remote via serial or Ethernet using a simple protocol, querying it periodically for azimuth and elevation. The control protocol also offers some ancillary commands for controlling output pins, sniffing remote serial ports, and sending data out remote serial ports which may be useful for automatting other things out at the tower such as antenna switching. But I digress.

Hardware

Serial

For a serial link, the host unit needs to have two hardware serial ports, the usual Serial port ("Serial0") and Serial1. This means that an Arduino Mega or a homebrew "bare chip" ATMega2560 or ATMega1284P is required to provide the second serial port. The host unit Serial1 port interfaces to the Serial port (Serial0) on the remote unit. Since the remote unit needs only one serial port, an Uno, bare

chip ATMega328 or just about any Arduino variant will work.

Ethernet

For an Ethernet master/slave you need to have two Ethernet shields, one for the master and one for the slave. I would recommend using Arduino Mega boards or equivalent homebrew AVR units as you will need more memory to compile and run the Ethernet code.

Serial Control Link Details

To interface the host with the remote, you simply need to connect TX on the host to RX on the remote, and RX on the host to TX on the remote. On an Arduino Mega the Serial1 port TX and RX is pins 18 and 19, respectively. On the remote, using an Arduino Uno, the Serial TX and RX pins are pins 1 and 0, respectively.

Depending on your particular installation, some line and signal conditioning may be required. I have successfully sent the 9600 baud serial link though 700 meters of unshielded twisted pair CAT 5 cable, using two twisted pairs with one conductor in each pair carrying ground and the other conductor carrying the signal. An additional conductor in the cable can carry DC voltage to power the remote unit.

A more robust solution would be to use specialized chips or a simple transistor interface to convert the 5 volt logic levels to standard RS-232 voltage levels which are +12 to 25 volts for a logic low, and -12 to -25 volts for a logic low. This however may be overkill. I'm experimenting with a simple 0 volt / +12 volt system and will post a schematic soon.

RF bypassing is a consideration as the remote will be operating in a high RF level environment and any long cabling will undoubtedly pick up RF along the way. Bypass capacitors of 0.01 uF or 0.001 uF should be placed on both serial lines at the host and remote units. RF shielding is another consideration for the remote unit.

The speed of the host/remote serial link is defaulted to 9600 baud, however higher speeds are possible and conversely if issues are encountered the link speed may be lowered. Configuration tweaking details are below.

Host Unit Compilation and Configuration

The host unit needs to have the following configured:

1. A one computer interface protocol, such `FEATURE_YAESU_EMULATION` or `FEATURE_EASYCOM_EMULATION`.
2. The appropriate azimuth and/or elevation position sensors pointed to the remote unit by activating `FEATURE_AZ_POSITION_GET_FROM_REMOTE_UNIT` and/or `FEATURE_EL_POSITION_GET_FROM_REMOTE_UNIT`. Do not configure any position sensors unless they are directly connected to the host unit. (You can have one position sensor on a remote unit and one on the host.)
3. For a serial link uncomment `#define FEATURE_MASTER_WITH_SERIAL_SLAVE`. For an Ethernet link, uncomment `#define FEATURE_MASTER_WITH_ETHERNET_SLAVE` and `#define FEATURE_ETHERNET`.
4. Configure the usual button pins, rotate pins, etc.
5. For an Ethernet link, configure the Ethernet port settings:

```
#define ETHERNET_MAC_ADDRESS 0xDE,0xAD,0xBE,0xEF,0xFE,0xED
#define ETHERNET_IP_ADDRESS 192,168,1,172
#define ETHERNET_IP_GATEWAY 192,168,1,1
#define ETHERNET_IP_SUBNET_MASK 255,255,255,0
```

... and configure the address of the remote slave:

```
#define ETHERNET_SLAVE_IP_ADDRESS 192,168,1,173
```

Remote Unit Compilation and Configuration

To configure a remote unit, do the following:

1. Activate `FEATURE_REMOTE_UNIT_SLAVE`
2. If using an Etherlink, activate `FEATURE_ETHERNET`

3. Activate the appropriate azimuth and/or elevation sensors (potentiometers, rotary encoders, I2C sensors) and configure the appropriate pins.
4. DO NOT activate any computer interface protocols such as `FEATURE_YAESU_EMULATION` or `FEATURE_EASYCOM_EMULATION`. The serial port on the remote will talk the host/remote protocol.
5. If using an Ethernet link, configure the Ethernet port on the slave unit:

```
#define ETHERNET_MAC_ADDRESS 0xDE,0xAD,0xBE,0xEF,0xFE,0xEE

#define ETHERNET_IP_ADDRESS 192,168,1,173

#define ETHERNET_IP_GATEWAY 192,168,1,1

#define ETHERNET_IP_SUBNET_MASK 255,255,255,0
```

If using an Ethernetlink master/slave link, don't forget to use different IP and MAC addresses for the master and slave units!

Master/Slave Protocol

In order to operate a master/slave system you do not need to know the master/slave serial protocol, but the master/slave protocol is simple and easy to understand. The design goals included fixed length arguments, consistent responses and event messages, and human-creatable and readable commands to insure easy testing and debugging. Below is a summary of the protocol.

Host to Remote Commands

PG - ping; the remote should respond with "PG"
AZ - read azimuth
EL - read elevation
DOxx - digital pin initialize as output; xx = pin # (01, 02, A0,etc.)
DIxx - digital pin initialize as input; xx = pin #
DPxx - digital pin initialize as input with pullup; xx = pin #
DRxx - digital pin read; xx = pin #

DLxx - digital pin write low; xx = pin #

DHxx - digital pin write high; xx = pin #

DTxxyyyy - digital pin tone output; xx = pin #, yyyy = frequency

NTxx - no tone; xx = pin #

ARxx - analog pin read; xx = pin #

AWxxyyy - analog pin write; xx = pin #, yyy = value to write (0 - 255)

SWxy - serial write byte; x = serial port # (0, 1, 2, 3), y = byte to write

SDx - deactivate serial read event; x = port #

SSxyyyyyy... - serial write sting; x = port #, yyyy = string of characters to send (variable length)

SAx - activate serial read event; x = port #

RB - reboot

CL - read the clock

Remote to Host Responses

ERxx - report an error; xx = error #

EV - report an event

OK - report success

CS - report a cold start

Error Codes

ER01 - Serial port buffer timeout

ER02 - Command syntax error

Events

EVSxy - Serial port read event; x = serial port number, y = byte returned (serial port #0 = control port, port#1 = remote unit control port)

Each command and response is terminated with a carriage return. Line feed characters may be sent either way, but are ignored.

The RB (reboot) command works on the Arduino Uno, however it locks up the Arduino Mega.

For a real world example of master and remote slave operation, check out the Frankenrotator project page.

Serial Link Testing and Debugging

To test the remote unit, configure the code as described above, upload it to the remote unit Arduino/AVR chip and connect it to the computer with the appropriate baud rate (9600 baud by default). Set the terminal program for carriage return and line feed.

Upon start up, the remote should send a cold start notification such as:

```
CS2013042101
```

This indicates the version of software running on the remote.

Next, test if the remote will respond to commands. Send a ping command:

```
PG
```

The remote unit should respond back with a ping:

```
PG
```

If that works, you can then read the azimuth and/or elevation from the connected sensors using the AZ

and EL commands. For fun you can play around reading and writing to pins. The following commands turn the LED on pin 13 on and off:

```
DO13

DH13

DL13
```

If you have remote unit hardware that supports additional serial interfaces like Serial1, Serial2, etc., you can read and write byte and strings on those ports. The commands below turn on monitoring of Serial1 and sends characters out it.

```
SA1

SS1test 1 2 3
```

The remote unit code supports the \d (backslash d) debugging commands which activates the periodic debug dump.

If the remote unit appears to be functioning properly, it's time to connect the host and remote units, as described above. I recommend you do this on the bench and not with the remote installed already outside. I would also recommend doing a direct connection at +5V logic levels and forgo any line signal level converters until later after you have the units talking on the bench.

If all goes well, the host unit should be reading the azimuth and/or elevation from the remote unit and report normally, just as though the position sensors were connected directly to the host unit. If not, the \d command should be used to view debugging information. In host mode, the debug dump displays an additional line showing link command counters:

```
debug: 2013042001BETA GS-232B
```

```
 AZ: IDLE Q: - AZ: 178.3 (raw: 178.3) Target: 0.0 (raw: 0.0) AZ Speed Norm: 255
Current: 255
 EL: IDLE Q: - EL: -50.7 Target: 0.0
 AZ: 180-450 AZ ana: 4-1009 EL ana: 2-1018
 Remote: Command: 1 Good: 31736 Bad: 0 Index: 0 CmdTouts: 0 BuffTouts: 0 Result:
-50.70
```

Here's a description of the counters:

Command: The current or most recently executed link command. 1 = AZ, 2 = EL

Good: The number of successfully executed commands. This counter rolls over at 65,536.

Bad: The number of unsuccessful commands. This may included commands that timed out waiting for a response or responses received that were malformed or invalid. This counter rolls over at 65,536.

Index: The number of bytes currently in the serial receive buffer. A non-zero value indicates a response is in progress or errant bytes are sitting in the buffer. (If bytes sit in the buffer for a period of time, the buffer is cleared and a buffer timeout is registered.)

CmdTouts: The number of commands that timed out awaiting a response from the remote.

BuffTouts: The number of times we received bytes from the remote, but no terminating carriage return. Without a carriage return, the response is considered incomplete and the receive buffer on the host is cleared.

Result: This is the last valid result received from an AZ or EL command. Normally this value will match the azimuth or elevation displayed in the debug dump lines.

A number of remote communications debugging commands are available on the host for troubleshooting and overall playing around:

\T - sniff the remote port transmit

\R - sniff the remote port receive

\Z - suspend automatic remote commands

\S - send a text string to the remote

With the \T command you can see the commands that are being transmitted to the remote. The \R command lets you see what is coming back from the remote. The \Z command suspends automatic commands. This is useful if you want to temporarily stop all the AZ and EL commands spewing on the screen and want to issue some specific commands for testing purposes. The \S command sends a text string to the remote. The \S command in combination with the remote port receive sniffing activated (\R) basically gives you an open command prompt to the remote unit. For example, you could do this:

```
\R

\Z

\SPG
```

These commands turn on remote receive sniffing, suspend automatic commands, send a PG (ping) to the remote. You should see a ping back. As you would expect, with the \S command you can send all of the other host/remote protocol commands like AZ, EL, DH, DDL, etc.

Another trick you can do is the get the remote unit to do a period debug dump using this sequence:

```
\Z

\R

\S\D
```

Tweaking the Serial Link Parameters

The serial control link is configured by default for 9600 baud. To change the link speed on the remote modify this line:

```
#define CONTROL_PORT_BAUD_RATE 9600
```

And on the host unit, modify this line:

```
#define REMOTE_UNIT_PORT_BAUD_RATE 9600
```

Various other host/remote communications settings:

```
#define REMOTE_BUFFER_TIMEOUT_MS 250

#define REMOTE_UNIT_COMMAND_TIMEOUT_MS 2000

#define AZ_REMOTE_UNIT_QUERY_TIME_MS 150

#define EL_REMOTE_UNIT_QUERY_TIME_MS 150
```

Ethernet

Link Operation

{under construction}

Troubleshooting, Debugging, and Testing

{under construction}

Tweaking and Advanced Configuration

{under construction}

```
#define ETHERNET_SLAVE_RECONNECT_TIME_MS 250

#define ETHERNET_SLAVE_TCP_PORT 23
```

```
#define ETHERNET_PREAMBLE "K3NG"
```

Special Configurations and Features

One Sensor on Remote Slave and One on Master Unit

The master / remote functionality can be used with one sensor connected to the master and the other connected to the remote. On the master, specify

```
FEATURE_AZ_POSITION_GET_FROM_REMOTE_UNIT
```

or

```
FEATURE_EL_POSITION_GET_FROM_REMOTE_UNIT
```

for whichever sensor is located on the remote unit, then specify the appropriate locally connected sensor feature for the sensor connected to the master. The master unit will query the remote only for the _GET_FROM_REMOTE_UNIT feature you've configured.

Remote Unit Pin Control and Reading

I/O pins on the remote slave can be controlled and read by the master unit for most functions. To redirect a master unit pin function to the remote, simply add 100 to the pin number in the rotator_pins.h file of the master unit. For example, to redirect the pin for rotate_cw to pin 13 on the remote, specify this in the rotator_pins.h file:

```
#define rotate_cw 113
```

This will make pin 13 on the remote slave become the rotate_cw pin.

To use an analog pin in this way, do this:

```
#define rotate_cw A0+100
```

Moon and Sun Tracking

Moon Tracking

Operation

To start tracking the moon, issue the \M1 command. To stop moon tracking issue the \M0 command, or issue the Yaesu S command, press the stop button, or issue any rotation command.

Configuration

To activate the moon tracking functionality, uncomment the following line before compiling:

```
#ifdef FEATURE_MOON_TRACKING
```

The following pins are available for your convenience:

Goes high when moon tracking is active:

```
#define moon_tracking_active_pin 0
```

Ground this pin to activate moon tracking (not for use with a button):

```
#define moon_tracking_activate_line 0
```

Use with a normally open momentary switch to ground:

```
#define moon_tracking_button 0
```

The following settings can be adjusted to customize for your installation:

```
#define MOON_TRACKING_CHECK_INTERVAL 5000
#define MOON_AOS_AZIMUTH_MIN 0
```

```
#define MOON_AOS_AZIMUTH_MAX 360
#define MOON_AOS_ELEVATION_MIN 0
#define MOON_AOS_ELEVATION_MAX 180
```

Sun Tracking

Operation

To start tracking the sun, issue the \U1 command. To stop moon tracking issue the \U0 command, or issue the Yaesu S command, press the stop button, or issue any rotation command.

Configuration

To activate the sun tracking functionality, uncomment the following line before compiling:

```
#define FEATURE_SUN_TRACKING
```

The following pins are your friends:

Goes high when sun tracking is active:

```
#define sun_tracking_active_pin 13
```

Ground this pin to activate sun tracking (not for use with a button):

```
#define sun_tracking_activate_line 0
```

Use with a normally open momentary switch to ground:

```
#define sun_tracking_button 30
```

The following settings can be adjusted to customize for your installation:

```
#define SUN_TRACKING_CHECK_INTERVAL 5000

#define SUN_AOS_AZIMUTH_MIN 0

#define SUN_AOS_AZIMUTH_MAX 360

#define SUN_AOS_ELEVATION_MIN 0

#define SUN_AOS_ELEVATION_MAX 180
```

Time and Location

Both the moon and sun tracking features require the exact time and location for the tracking algorithms. The internal clock functionality is described further below. The location of the unit can be determined one of several ways:

1. Statically configured in the code with these settings:

```
#define DEFAULT_LATITUDE 40.889958

#define DEFAULT_LONGITUDE -75.585972
```

2. Configured at runtime using the \G command to set the Maidenhead grid square. For example:

```
\GFN20EV
```

This would set the coordinates for the center of grid square FN20ev. Note that a six character grid square is required.

3. Use the GPS synchronization functionality, described further below.

Tracking Status

Currently the status of moon and sun tracking can be viewed using the debug output, the \d command.

(A future update will provide a more user-friendly display.)

Clock

Operation

In order to support sun and moon tracking, a time of day clock function is provided. This uses the internal Arduino millis() function.

The clock can be queried using the \C command. The clock can be set with the \O command like so:

```
\O201401021435
```

The example above sets the clock to 2014-01-02 14:35 Z. If a realtime clock (RTC) module is connected, the \O will also program the module time.

The Arduino clock frequency can be rather inaccurate over time, often losing or gaining tens of seconds a day, and it can vary with temperature. A realtime clock module or GPS synchronization is recommended if accurate timekeeping is desired.

The internal clock does not continue to run when the Arduino loses power and must be set each session with the \O command, if an RTC or GPS is not used.

If the unit is equipped with both an RTC and GPS, the clock will synchronize off of the GPS first, and then synchronize from the RTC if GPS is not available.

If a master/slave system is used, the GPS (and RTC) can be remotely located and connected to the slave unit, and the master unit can be configured to synchronize its clock from the remote slave's clock.

Configuration

The clock feature is enabled with the follow line:

```
#define FEATURE_CLOCK
```

The timekeeping accuracy of the clock can be adjusted using this setting:

```
#define INTERNAL_CLOCK_CORRECTION 0
```

A positive number compensates for a slow clock and a negative number compensates for a fast clock. A number of 0.00145 would make the clock run 0.145% faster.

Master Clock Synchronization from Remote Slave Unit

To active this feature, activate `OPTION_SYNC_MASTER_CLOCK_TO_SLAVE`.

Realtime Clock Modules

Operation

DS1307 and PCF8583 realtime (RTC) clock modules are supported. These modules maintain the time, even when not powered as they have self-contained batteries.

The time of the realtime clock is set by the \O command, described above in the section on the Clock functionality. If the rotator controller is also GPS equipped, the RTC will be periodically updated with the GPS time.

Note that cheap RTC modules may have issues with timekeeping accuracy and battery charging. Consult Google if you have issues.

Configuration

To enable DS1307 module support, uncomment this line:

```
#define FEATURE_RTC_DS1307
```

To enable PCF8583 module support, uncomment this line:

```
#define FEATURE_RTC_PCF8583
```

This setting defines how often the internal Arduino clock is synchronized with the RTC module:

```
#define SYNC_WITH_RTC_SECONDS 59
```

This setting configures how often the RTC is synchronized with GPS, if it is available:

```
#define SYNC_RTC_TO_GPS_SECONDS 12
```

Hardware Connection

Both modules connect to the I2C bus, the SDA and SCA pins (pins 20 and 21 on an Arduino Mega).

Debugging

Use the \d command to see the status of the clock. The clock status is highlighted below:

```
debug:    1.9.2014021501-UNSTABLE                2014-02-16 01:38:06Z          RTC_SYNC  40.8806 -75.5865              GS-232B

          AZ: IDLE  Q: - AZ: 240.0 (raw: 240.0) (raw: 0.0)  Analog: 140 (4-1009) [180+450]  AZ Speed Norm: 253 Current: 253
Offset: 0.00

          EL: IDLE  Q: - EL: 24.0 EL Analog: 141 (2-1018)   EL Speed Norm: 253 Current: 253         Offset: 0.00

          moon: AZ: 103.90 EL: 21.27    TRACKING_INACTIVE   sun: AZ: 284.64 EL: -34.65    TRACKING_INACTIVE
```

The clock states are:

FREE_RUNNING - No synchronization with RTC or GPS

RTC_SYNC - Using the realtime clock for time

GPS_SYNC - Synchronized to GPS

SLAVE_SYNC - The master unit clock is synchronized to the slave unit clock

GPS

Operation

GPS units with a standard NMEA serial output can be used with rotator controller. The GPS unit interfaces to the Serial2 port by default. The rotator controller parses the GPS output and derives time and location information.

After power up, it may take 10 seconds to 5 minutes for GPS synchronization to occur. Most GPS units will obtain a satellite fix in less than thirty seconds if they have recent almanac information stored, which is usually considered less than four hours old. If almanac information is not available, the GPS goes into a "cold start" state where it has to obtain almanac information from a satellite before it can acquire and calculate accurate time and position information.

Configuration

To enable the GPS feature, uncomment the following line:

```
#define FEATURE_GPS
```

Set the serial port baud rate here:

```
#define GPS_PORT_BAUD_RATE 9600
```

If you want the clock to synchronize with the GPS unit, set this to 1(this is 1 by default):

```
#define SYNC_TIME_WITH_GPS 1
```

Location synchronization is enabled here (this is enabled by default):

```
#define SYNC_COORDINATES_WITH_GPS 1
```

Several settings adjust the behavior of GPS synchronization. This setting determines how long the clock will remain synchronized from a valid GPS fix:

```
#define GPS_SYNC_PERIOD_SECONDS 10
```

This setting determines how old a satellite fix from a GPS will be accepted as valid:

```
#define GPS_VALID_FIX_AGE_MS 10000
```

If you wish to have an LED or output pin to indicate GPS synchronicity, define this pin:

```
#define gps_sync 0
```

This setting compensates for processing and serial port propagation delay:

```
#define GPS_UPDATE_LATENCY_COMPENSATION_MS 200
```

Debugging

Use the \d command to see the status of the GPS. The clock status and current coordinates are highlighted below:

```
debug:    1.9.2014021501-UNSTABLE                2014-02-16 01:38:06Z          GPS_SYNC  40.8806 -75.5865              GS-232B

          AZ: IDLE  Q: - AZ: 240.0 (raw: 240.0) (raw: 0.0)  Analog: 140 (4-1009) [180+450]  AZ Speed Norm: 253 Current: 253
Offset: 0.00

          EL: IDLE  Q: - EL: 24.0 EL Analog: 141 (2-1018)   EL Speed Norm: 253 Current: 253         Offset: 0.00

          moon: AZ: 103.90 EL: 21.27    TRACKING_INACTIVE   sun: AZ: 284.64 EL: -34.65    TRACKING_INACTIVE
```

The clock states are:

FREE_RUNNING - No synchronization with RTC or GPS

RTC_SYNC - Using the realtime clock for time

GPS_SYNC - Synchronized to GPS

The coordinates highlighted above:

40.8806 - Latitude

75.5865 - Longitude

If GPS_SYNC is shown and SYNC_COORDINATES_WITH_GPS is set to 1 in rotator_settings.h, the coordinates will update automatically from GPS.

GPS Mirror Port

If you wish to use the GPS data with another application, such as having a computer read the NMEA data and update its clock, you can configure the rotator controller to mirror all data coming into the GPS port out another Arduino/AVR serial port. To enable this, uncomment this line and set the appropriate serial port:

```
#define GPS_MIRROR_PORT &Serial3
```

And set the baud rate for the port:

```
#define GPS_MIRROR_PORT_BAUD_RATE 9600
```

Ethernet

Operation

The code supports the standard Arduino Ethernet shield (http://arduino.cc/en/Main/ArduinoEthernetShield#.Uv_5G9-9uR0) and any compatible shields. The unit accepts up to two telnet connections and all commands and protocols supported on the serial control port are supported on the Ethernet port.

The default configuration listens on two TCP ports, port 23 and port 24. Port 23 is the standard telnet port. You can telnet to the Ethernet port on most computer operating systems using the telnet command like so:

```
telnet 192.168.1.173
```

To telnet to port 24, use this command:

```
telnet 192.168.1.173 24
```

Note that only one telnet client can connect to a port at a time. If you connect more than one client, the operation will be erratic.

Configuration

The Ethernet functionality is enabled with this option:

```
#define FEATURE_ETHERNET
```

The layer two MAC address of the Ethernet port is set here, in the rotator_settings.h file:

```
#define ETHERNET_MAC_ADDRESS 0xDE,0xAD,0xBE,0xEF,0xFE,0xED
```

The layer three IP address, gateway, and subnet mask is set here:

```
#define ETHERNET_IP_ADDRESS 192,168,1,173
#define ETHERNET_IP_GATEWAY 192,168,1,1
```

```
#define ETHERNET_IP_SUBNET_MASK 255,255,255,0
```

The layer four TCP ports that are the telnet listeners are configured here:

```
#define ETHERNET_TCP_PORT_0 23

#define ETHERNET_TCP_PORT_1 24
```

Testing and Debugging

{under construction}

Other Interfacing Options

Rotation Indicator Output Pin

This output pin will change state based on the rotation status. The pin is defined on this line:

```
#define rotation_indication_pin 0
```

Related settings:

```
#define ROTATION_INDICATOR_PIN_ACTIVE_STATE HIGH
#define ROTATION_INDICATOR_PIN_INACTIVE_STATE LOW
#define ROTATION_INDICATOR_PIN_TIME_DELAY_SECONDS 0
#define ROTATION_INDICATOR_PIN_TIME_DELAY_MINUTES 0
```

Heading Reading Inhibit Pin

This input pin will stop azimuth and elevation readings from being taken when the pin is taken high. The pin is defined here:

```
#define heading_reading_inhibit_pin 0
```

This feature is useful for when sensors are interfered with during transmit. If the transmitter PTT line is interfaced with this pin, the rotator controller will stop taking sensor measurements and will use the last headings until the heading reading inhibit pin goes low.

Ancillary Pin Control

This feature allows the user to manually control hardware pins via the command line.

Feature Activation:

```
FEATURE_ANCILLARY_PIN_CONTROL
```

Command Line Commands:

`\Fxx` - Turn pin off; xx = pin number

`\Nxx` - Turn pin on; xx = pin number

`\Wxxyyyy` - Turn on pin PWM; xx = pin number, yyy = PWM value (0-255)

Power Switch

The controller software can control an external circuit to deactivate power to the unit after a period of inactivity.

Feature Activation:

```
FEATURE_POWER_SWITCH
```

Pins:

```
#define power_switch 0
```

Related settings:

```
#define POWER_SWITCH_IDLE_TIMEOUT 15
```

The timeout unit is minutes.

Analog Reference Pin

To activate use of the Arduino/AVR analog reference pin for any analog readings (i.e. azimuth and elevation potentiometer sensors), activate `OPTION_EXTERNAL_ANALOG_REFERENCE`. This can be used to increase the accuracy of readings by tieing the analog reference pin to the +5 volt line.

Advanced Configuration Options

Serial Port Mapping

By default the serial ports are mapped as follows:

Serial (the Arduino USB port): control port (Yaesu / Easycom protocol, backslash commands)

Serial1 : remote unit control port (the port a master talks to a remote slave with)

Serial2: GPS interface port

These mappings along with port baud rates can be changed in the rotator_settings.h file in this section:

```
#define CONTROL_PORT_BAUD_RATE 9600
#define REMOTE_UNIT_PORT_BAUD_RATE 9600
#define GPS_PORT_BAUD_RATE 9600
#define CONTROL_PORT_MAPPED_TO &Serial
#define REMOTE_PORT_MAPPED_TO &Serial1
#define GPS_PORT_MAPPED_TO &Serial2
```

Note that Serial1 and Serial2 are not available on an Arduino Uno. You must use another unit like an Arduino Mega or an Atmel ATMega 2560.

Configuration Reference

{under construction}

`OPTION_BUTTON_RELEASE_NO_SLOWDOWN`

This disables automatic slowdown when manual rotation buttons are depressed and released.

Debug Reference

{under construction}

EMI and ESD

When planning a rotor system based on the Arduino processor the designer should be aware that the Arduinos and their attendant shields were intended for physically small systems, like a robot or a CW keyer. In contrast an antenna rotor is a large distributed system with unique challenges relating to moving digital data hundreds of feet in an electrical environment that is unforgiving. Things to consider:

Clocked ICs radiating into the antenna.

High RF fields getting into the IC inputs.

External fields, such as generated by nearby lightning, destroying IC inputs.

Electrical pickup by long conductors in the near field of the antenna.

Magnetic and electrical radiation from brushed DC motors.

SPI and I2C signals were not intended to be used for distances of more than a few feet.

The required interface ICs and associated EMI suppression can easily become more complex than the Arduino modules.

As a minimum all electronics should be enclosed by metal with shielded cable between units.

Getting the Source Code

Source code is located on SourceForge. The master tree is the main stable release tree and is what you should use for production units. Code with new features which may not be fully tested (and not documented here) is located in unstable tree.

Support and Feature Requests

Support for this software can be obtained on the Radio Artisan Group (https://groups.yahoo.com/neo/groups/radioartisan/info).

Acknowledgements

John, W3SA, has tested on a Yaesu Az/El unit, contributed several updates to the elevation code, and tweaked the code for a 16 column LCD display.

Anthony, M0UPU, wrote about his rotator controller construction and is offering PC boards.

Bent, OZ1CT, has contributed several ideas and feature requests, and performed testing.