yaesuControl Python Class Documentation

Vance Martin

yaesuControl Python Class Documentation

This class is designed to control the the serial interface between a computer running Linux and a Yaesu 857d amateur radio transceiver. In the amateur radio community, the single board computers such as the Beaglebone and the Raspberry Pi are very popular for use in the "Shack" and for various radio related experiments and projects. This class was written with the intent of being able to run on a Raspberry Pi computer, and to handle the communication between the Raspberry Pi and the transceiver, so that operators can design their own user interface software for controlling the transceiver. While it is quite likely that the class would operate on other operating systems and on other types of computers, it has not been tested in these other applications, and would likely need some modifications to take control of the appropriate communication ports on systems other than the Raspberry Pi. Another primary goal of this class was to remain lightweight, so that operators who wish to design software to allow remote control of the radio using the Raspberry Pi, can do so. This class imports the pySerial class for performing the functions needed to set up the communications port, so any users of this class will also need to download, and prepare for use, the pySerial module available at pyserial.sourceforge.net/.

This documentation will outline each class object attribute, the data to be passed into the attribute, and any data returned by the attribute. Each section will also briefly outline possible use cases for the attribute, where appropriate.

__init__ ( )

Initialization calls the pySerial class and creates a serial object. The port is set to the default for the Raspberry Pi (/dev/ttyAMA0), and the communication port settings are set-up for establishing a connection between the Raspberry Pi and the transceiver.

radioModes { }

This attribute is a dictionary containing the available operating modes for the radio as

the keys, and the piece of the hexadecimal command that the transceiver understands as their equivalent. The mode abbreviations used are the standard abbreviations for amateur operations. The available modes are:

LSB: Lower Side Band

USB:Upper Side Band

CW: Continuous Wave (Morse Code)

CWR: Continuous Wave, Reverse Side Band

AM: Amplitude Modulation

FM: Frequency Modulation

NFM: Narrow Bandwidth Frequency Modulation

DIG: Digital

PKT: Packet

CTCSSTones { }

The CTCSSTones attribute is a dictionary containing the standard CTCSS tones used for tone squelch in amateur operations, as the keys. The values are the associated hexadecimal commands that the transceiver understands as their equivalent. Below are the available CTCSS tones.

67.0	69.3	71.9	74.4	77.0
79.7	82.5	85.4	88.5	91.5
94.8	97.4	100.0	103.5	107.2
110.9	114.8	118.8	123.0	127.3

131.8	136.5	141.3	146.2	151.4
156.7	159.8	162.2	165.5	167.9
171.3	173.8	177.3	179.9	183.5
186.2	189.9	192.8	196.6	199.5
203.5	206.5	210.7	218.1	225.7
229.1	233.6	241.8	250.3	254.1

DCSCodes { }

The DCSCodes attribute is a dictionary containing the standard DCS codes used for digitally coded squelch in amateur operations, as the keys. The values are the associated hexadecimal commands that the transceiver understands as their equivalent. Below are the available DCS codes

.

006	007	015	017	021	023	025	026	031	032	036
043	047	050	051	053	054	065	071	072	073	074
114	115	116	122	125	131	132	134	141	143	145
152	155	156	162	165	172	174	205	212	214	223
225	226	243	244	245	246	251	252	255	261	263
265	266	271	274	306	311	315	325	331	332	343
346	351	356	364	365	371	411	412	413	423	431
432	445	446	452	454	455	462	464	465	466	503

506	516	523	526	532	546	565	606	612	624	627
631	632	654	662	664	703	712	723	731	732	734
743	754									

startRadioComm ( )

This method takes no arguments, initializes serial communication with the radio, and returns nothing. This method must be called, to establish communications with the radio, prior to sending any commands to the radio.

StopRadioComm ( )

This method takes no arguments, closes communication with the radio, and returns nothing. This method should be called when done sending commands and receiving data from the radio, to ensure that the communications port is released for other applications or uses.

lockOn ( )

This method takes no arguments, locks the radio keypad, and returns nothing. Locking the radio keypad disables the pushbuttons and other controls on the face of the transceiver. When remotely accessing the radio, it may be a best practice to engage the keypad lock to prevent someone from accidentally changing radio settings at the radio, which could interfere with the intended remote operations.

lockOff ( )

This method takes no arguments, unlocks the radio keypad, and returns nothing.

pttOn ( )

This method takes no arguments, engages the transceiver's push-to-talk circuitry, and returns nothing. Engaging the push-to-talk circuitry effectively turns on the radio's transmitter, and opens the

audio/data path so that voice or data sent to the radio through the inputs or microphone, can be broadcast on the current transmitting frequency.

pttOff ( )

This method takes no arguments, disengages the transceiver's push-to-talk circuitry, and returns nothing. This turns off the radio's transmitter, and puts the radio into receive mode.

clarifierOn ( )

This method takes no arguments, turns on the radio's clarifier function, and returns nothing. The clarifier sets the radio to transmit and receive on different frequencies. The offset between the transmit and receive frequencies is set using the clarifierOffset( ) method. More details on clarifierOffset( ) are provided later in this document.

clarifierOff ( )

This method takes no arguments, turns off the radio's clarifier function, and returns nothing.

vfoToggle ( )

This method takes no arguments, toggles between the radio's two variable frequency oscillators, and returns nothing. The Yeasu 857d is equipped with two variable frequency oscillators, referred to as VFO A and VFO B. Unfortunately, the radio is not equipped with a feedback method to read through the serial port which VFO is in use. Software that uses this method should either be written in a manner that doesn't rely on knowing which VFO is in use, or should clearly direct the user to always manually set the radio to the desired VFO before starting remote control.

splitOn ( )

This method takes no arguments, turns on the radio's split operation function, and returns nothing. Similar to the clarifier function, the split operation function allows transmitting and receiving on two different frequencies. It allows for greater offsets between transmit and receive however, because it receives using one VFO and transmits using the other. A common practice is to choose a

receive frequency on the current VFO, switch VFO's, choose a transmit frequency, and then switch back to the first VFO, before engaging the split function.

splitOff ( )

This method takes no arguments, turns off the radio's split operation function, and returns nothing.

readFreqAndMode ( )

This method takes no arguments, queries the radio for its frequency and mode, and returns a string representing the currently active frequency and mode of the radio. The string returned is the string representation of the data the radio returns. The first 8 characters represent the frequency:

1st digit : 100MHz

2nd digit: 10MHz

3rd digit: 1MHz

4th digit: 100kHz

5th digit: 10kHz

6th digit: 1kHz

7th digit: 100Hz

8th digit: 10Hz

The last 2 characters represent the current operating mode of the radio:

00: LSB

01: USB

02: CW

82: CW-N

03: CWR

04: AM

06: WFM

08: FM

88: NFM

0A: DIG

0C: PKT

For example, if the current frequency of the radio is 14.313 and the operating mode is USB, calling readFreqAndMode( ) would return '0143130001'.

readFrequency ( )

This method takes no arguments, performs operations using readFreqAndMode( ), and returns a floating point number representing the current operating frequency. The frequency is represented in MHz, with a precision of up to 5 elements after the decimal.

readMode ( )

This method takes no arguments, performs operations using readFreqAndMode( ), and returns a string representing the current operating mode of the radio. The possible returned values are the key vales in the radioModes{ } dictionary described earlier in this document.

ReadThis ( )

This method takes no arguments, prints characters output by the radio, and returns nothing. This function is primarily for troubleshooting. It prints one character of the radio output at a time, until the radio is sending no data. Because this function prints directly to the screen, it is recommended that it only be used for development purposes, or in cases where the application is text based, and run at a command line.

setFrequency (frequency)

This method takes a floating point number representing the desired frequency, sets the radio to that frequency, and returns nothing. The value of frequency needs to be the desired frequency in MHz,

and must be between, and including, 1.8 – 450 MHz.

setOperatingMode (mode)

This method takes a string representing the desired operating mode, sets the radio to that operating mode, and returns nothing. The value of mode must be one of the keys included in the mode{ } dictionary described at the beginning of this document (case sensitive).

setClarifierOffset (kHzOffset)

This method takes a floating point number representing frequency in kHz, sets the offset amount for the clarifier function, and returns nothing. This method is intended to be used with the setClarifierOn( ) and setClarifierOff( ) methods described earlier, to allow transmitting and receiving on frequencies up to 9.99kHz apart. When using this function the transmit frequency remains constant, but the receive frequency is offset by the amount specified by the value of kHzOffset.

setRptOffsetDirection (offsetDirection)

This method takes a string, sets the offset direction for repeater operation based on the value of the string, and returns nothing. Due to the internal circuitry of the radio, this function will only take action on the radio when the radio is set to the FM operating mode, because it is intended for use with repeaters, which operate using frequency modulation. The value of the string offsetDirection must be either '+' for a repeater that has an input frequency higher than its output frequency, '-' for a repeater that has an input frequency lower than its output frequency, or 'simp' for simplex (non-repeater operation.)

setRptOffsetFrequency (mHzOffset)

This method takes a floating point number representing frequency, sets the offset amount for repeater operation, and returns nothing. The value of mHzOffset must be in MHz, and must be between, and including, 0 to 99.99MHz. When using repeaters, the receive frequency displayed while receiving remains the same, and the transmit frequency, which is offset by the value of mHzOffset and

in the direction of setRptOffsetDirection( ), will be displayed while transmitting.

setDCSmode (DCSflag)

This method takes a string, sets the DCS mode of the transceiver based on the value of the string, and returns nothing. The allowable values of DCSflag are 'both', 'encode', or 'decode'. When the DCS mode is 'both' the transceiver will use the DCS code set by the method setDCSCode( ), during both transmitting and receiving. When the mode is 'encode' it will only use the code when transmitting, and when the mode is 'decode' the it will only use the code when receiving. Due to the radio's internal design, this function will only take action on the radio when the radio is in FM operating mode. Additionally, the 'encode' and 'decode' modes will only work if split CTCSS/DCS encoding is first enabled through the radios internal menu (menu item 097.)

setCTCSSmode (CTCSSflag)

This method takes a string, sets the CTCSS mode of the transceiver based on the value of the string, and returns nothing. The allowable values of CTCSSflag are 'both', 'encode', or 'decode'. When the CTCSS mode is 'both' the transceiver will use the CTCSS tone set by the method setCTCSSTone( ), during both transmitting and receiving. When the mode is 'encode' it will only use the code when transmitting, and when the mode is 'decode' it will only use the code when receiving. Due to the radio's internal design, this function will only take action on the radio when the radio is in FM operating mode. Additionally, the 'encode' and 'decode' modes will only work if split CTCSS/DCS encoding is first enabled through the radios internal menu (menu item 097.)

setCTCSSTone (toneTX, toneRX)

This method takes two floating point numbers, sets the CTCSS tones used for transmit (encode) and receive (decode), and returns nothing. The value of toneTX must be one of the keys in the dictionary CTCSSTones{ } described earlier in this document, and is used to set the tone used during transmitting. The value of toneRX must also be one of the keys in the dictionary CTCSSTones{ }, and

is used to set the tone used during receiving. The values of toneTX and toneRX will often be the same, as most repeaters use the same tone for transmit and receive.

setDCSCode (codeTX, codeRX)

This method takes two integers, sets the DCS code used for transmit (encode) and receive (decode), and returns nothing. The value of codeTX must be one of the keys in the dictionary DCSCodes{ } described earlier in this document, and is used to set the code used during transmitting. The value of codeRX must also be one of the keys in the dictionary DCSCodes{ }, and is used to set the code used during receiving. The values of codeTX and codeRX will often be the same, as most repeaters use the same codes for transmit and receive.

increaseFrequency (stepUp)

This method takes a floating point number representing frequency, increases the frequency of the radio, and returns nothing. The value of stepUp is in MHz, and represents the amount by which the operating frequency should be increased. Although there is a separate method for decreasing frequency, this method can also by used in decrease frequency, but using a negative value for stepUp.

decreaseFrequency (stepDown)

This method takes a floating point number representing frequency, decreases the frequency of the radio, and returns nothing. The value of stepDown is in MHz, and represents the amount by which the operating frequency should be decreased. Although there is a separate method for increasing frequency, this method can also by used in increase frequency, but using a negative value for stepDown.

splitOperation (splitOffset, receiveFrequency)

This method takes two floating point numbers representing frequency, sets up the radio for split operations, engages split operations, and returns nothing. The value of splitOffset, either negative or positive, determines the distance between the transmit and receive frequencies. The value of

receiveFrequency sets the receive frequency for operations. When operating in split mode, the transceiver will be receiving on the frequency set by receiveFrequency, and when transmitting, will adjust the frequency based on the value of splitOffset.

repeaterSetup (repeaterOutput, rptOffset, direction,toneORdcs, tone)

This method takes several arguments as detailed below, sets up the radio for basic repeater operations, and returns nothing. This method works primarily by calling other methods already described , and exists simply to have one method that will work for setting up operations on most repeaters. This function is designed to work with basic repeaters, and will not work for repeaters with separate transmit and receive tones or codes. The arguments needed for this method are:

repeaterOutput: This should be a floating point number that represents the repeaters output frequency (the users receiving frequency) in MHz. It is set using the setFrequency( ) method.

rptOffset: This is a floating point number representing the desired repeater offset distance. It is set using setRptOffsetFrequency( ), so please see the associated section in this document for allowable values and details.

direction: This is a string representing the the desired repeater offset direction. It is set using setRptOffsetDirection( ), so please see the associated section in this document for allowable values and details.

toneORdcs: This is a string that is used to identify if the repeater to be used uses CTCSS or DCS encoding. A value of 't' identifies that CTCSS tones will be used, and a value of 'd' identifies that DCS encoding will be used.

tone: This is a floating point number or an integer that represents the tone or code to be used for CTCSS or DCS encoding. It is set using either setCTCSSTone( ) or setDCSCode( ), so please so the associated sections in this document for allowable values and details.

bandSelection (band, mode)

This method takes two strings, sets the radio to a specific frequency and mode, and returns nothing. This method is designed to mimic the functions performed by the band selection buttons on the front of most modern radios. The value of band indicates the desired frequency band, and the value of mode indicates which portion of the band to choose. Calling this function sets the frequency to the low end of the selected portion of the chosen band. Allowable values are:

Band: '160m', '80m', '40m', '30m', '20m', '17m', '15m', '12m', '10m', '6m', '2m', '70cm'

Mode: 'd', for the digital portion of the band; any other value for the voice portion of the band.