WaveConverter User Guide

Overview

WaveConverter extracts digital data from an input signal file based on a set of user-defined signal characteristics collectively referred to as a ***protocol***.

To start WaveConverter in GUI mode, navigate to the src subdirectory of the install directory and type:

```
./waveconverter.py -g
```

For information on additional command line options, see the section below.

Recommended File Naming

Although WaveConverter can process input files with any legal file name, I recommend that you use the following convention for raw RF (IQ) files:

```
<description>_c<center frequency>_s<sample rate>.iq
```

For example, a keyfob captured at a center frequency of 314.1MHz and a sample rate of 8MHz would be rendered as:

```
mykeyfob_c314p1M_s8M.iq
```

Embedding this information in the file name is always a good practice because it prevents you from losing or forgetting this valuable meta data. Furthermore, WaveConverter automatically extracts these two parameters from the file name, so that you don't have to enter them manually.

General Usage

The WaveConverter work flow moves through the following stages, each of which is represented by a tab in the GUI: **Demodulation**, **Deframing**, **Decoding**, **Statistical Analysis** and **CRC Verification**. These steps must be completed in the order given, as the output of the previous stage is used as input to the next.

You perform the **Demodulation** step by entering in the parameters that control the tuning, filtering and demodulation of the RF waveform. You may need to view your RF data in gnuradio-companion to see what kinds of values you should use here. The output of this step is a series of digital waveforms, each representing an individual transmission.

The **Deframing** step checks the preamble for each transmission to determine if it is legitimate and where the payload data starts. You perform this step by observing the digital waveform and defining the timing of the preamble sequence.

Next, you will **Decode** the portion of the transmission coming after the preamble by entering the timing

characteristics you observe in the digital waveform. This will produce a number of lines in the lower right-most pane of the Decode tab. Each line corresponds to an individual transmission and contains the binary (or hex if selected) data decoded from the waveform using the parameters you specified. You can also check the upper left corner to see the how many transmissions were correctly decoded.

You will then perform **Statistical Analysis** on the data you've just acquired to determine the function of each bit. Some of these bits may correspond to Identification Numbers, some to measured values and some to CRC values. You can define certain bit ranges as IDs or values and then re-run the Stats function to display the results of using those assumptions.

Finally, you can attempt to verify the results of any **CRC** fields by configuring the CRC parameters and re-running the decode function. *NOTE: The CRC tab's functions are in work and may have undiscovered issues. I've included it in this release because it may provide value to those with extensive CRC experience or to those willing to look through the source code.*

User Interface - GUI

Menu Bar

The UI contains a standard menu bar, with File and Help options.

File - Open RF IQ File

File – Open Baseband File

File - Quit

About - User Guide

Tool Bar

These controls allow you to save and retrieve protocols to the GUI, as well as other global display options.

Load Protocol – Brings up dialog with a list of stored protocols from which you may select

Save Protocol – If a protocol is currently loaded, this command replaces it with the contents of the GUI. If no protocol is loaded, this button is equivalent to clicking on “Save Protocol As” below.

Save Protocol As – Brings up a dialog asking for a name, a device type and a year. After entering this info and clicking OK, a new protocol is created with this info plus the contents of the GUI.

Demod – Demodulates the current IQ file using the parameters entered on the ***RF Demod Tab***, producing a set of baseband waveforms.

Decode – Decodes the currently loaded baseband waveforms using the parameters entered on the ***Framing*** and ***Payload Decode*** tabs. This baseband waveform may be loaded from a file or the

product of a demodulation operation in WaveConverter.

RunStat – Performs a statistical analysis of the decoded data, using the payload properties entered into the ***Payload Stats*** tab.

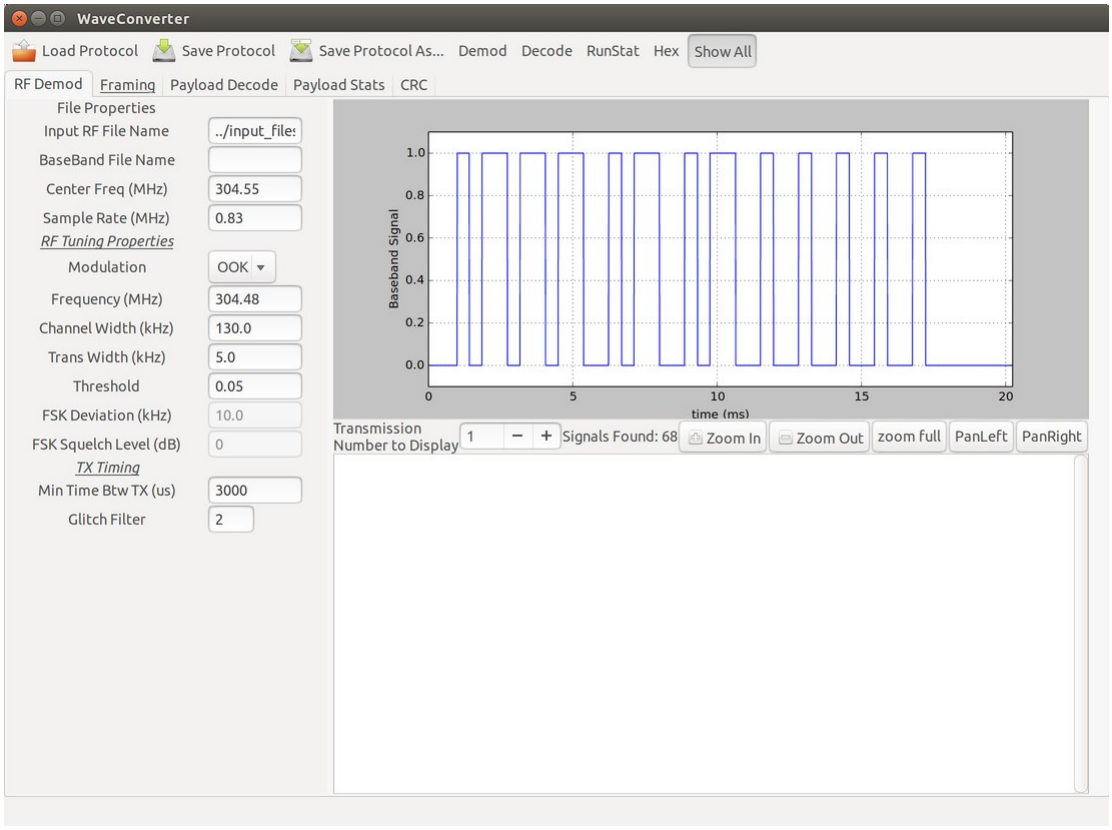
Hex – This is a toggle switch which is not active by default. When inactive, all payload data is displayed in binary format. When active, all payload data is displayed in hexadecimal format.

Show All – Toggle switch, inactive by default. When active, all transmissions are displayed and considered when computing payload statistics. When inactive, only the transmissions that pass framing and CRC checks are displayed and part of the statistics. Enabling this prevents bad transmissions from distorting your results. It is often useful to disable this during the early part of the reverse engineering process and enable it later on.

Tabbed Interface

The UI is further divided into 5 tabs. Each of the tabs is active at all times, so values entered into a previously opened tab remain unless replaced by a ***Load Protocol*** operation.

RF Demod Tab



This tab is used to define the RF characteristics of the input signal. WaveConverter's demodulator takes an input IQ File, tunes to the target frequency, filters anything but the target signal, decimates to a 100kHz sample rate (10 us samples) and finally demodulates the signal. In this way, WaveConverter

takes an IQ file and produces a digital baseband waveform.

The following five parameters are not part of any protocol definition but are user-defined. In other words, signals using the same protocol may have different values for the following five parameters.

Input RF File Name – Name and path of the file containing the RF data on which to operate. This file must contain complex data in an I-Q format, such as the files resulting from the direction of an SDR source to a File Sink in gnuradio.

Baseband File Name – Name and path of the file containing the baseband input data. This file must be a binary file consisting of single byte samples, each equal to 0x01 or 0x00. WaveConverter further assumes that this file was produced at a sample rate of 100ksamples/s. You should only load either an RF file or a baseband file, not both. You must still click the demodulation button after loading the baseband file, as this will apply the glitch filter and split up the baseband into individual transmissions.

Center Freq (MHz) – The center frequency of the captured RF data.

Sample Rate (MHz) – The sampling rate at which the I-Q file's data was captured.

Min Time Btw TX (us) – The minimum time in microseconds that separates each transmission. WaveConverter uses this parameter to separate the demodulated signal stream into individual transmissions.

Glitch Filter – The glitch filter ignores any transitions shorter than this value. This value is in units of 10us.

The remaining parameters on this page are part of the signal protocol and will be populated when a protocol is loaded.

Modulation – Directs WaveConverter to use an OOK or FSK demodulation scheme.

Frequency (MHz) – Frequency of the transmissions. It may be helpful to view an FFT or Waterfall plot of your I-Q file to determine the frequency of your signal.

Channel Width (kHz) – The bandwidth occupied by your signal.

Trans Width (kHz) – The transition width (difference between passband and stopband) of the channel filter used before demodulating the signal. Setting this to about 10% of the Channel Width is a good rule of thumb.

Threshold – Level that defines the difference between a digital zero and a digital one. Dependent on signal strength. Setting this too low may result in noise affecting the demodulator output, especially in OOK signals. Setting it too high may result in a failure to acquire any baseband waveform.

FSK Deviation (kHz) – Difference between space and mark frequencies. Only needed for FSK demodulation.

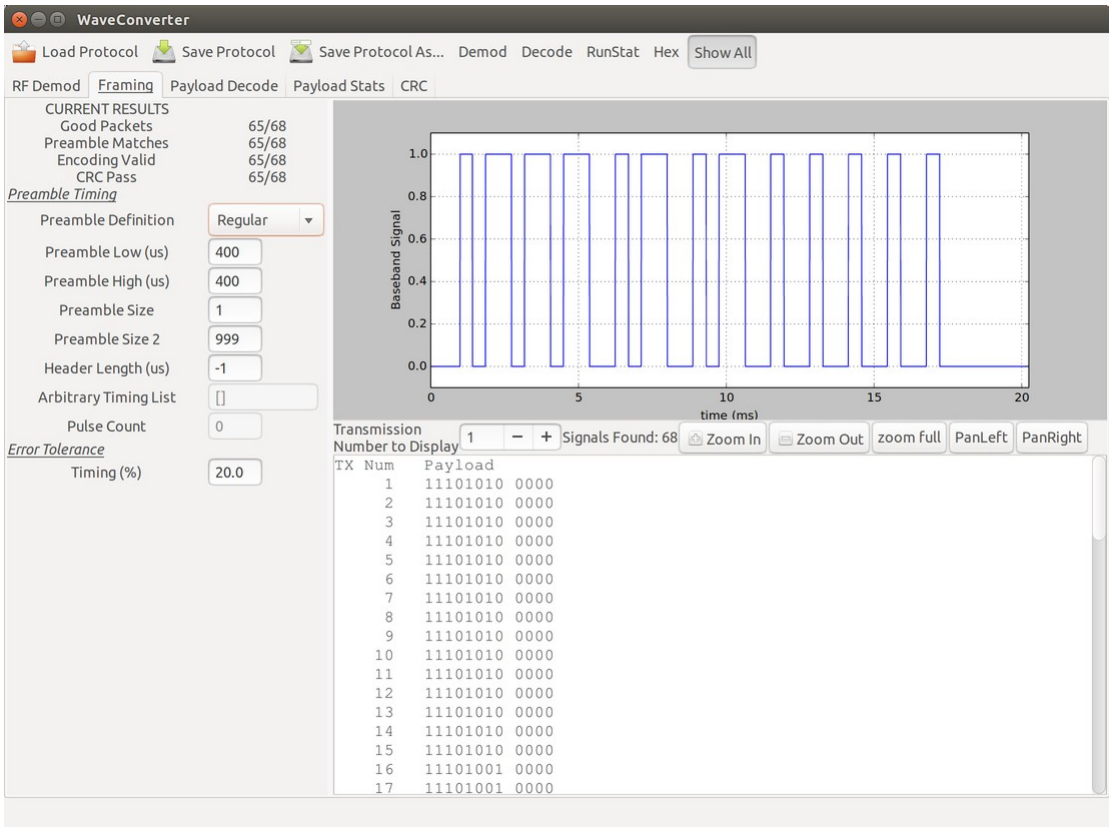
FSK Squelch (dB) – Prevents any signals below this value in negative dB to undergo demodulation. For example, entering 40 here will cause WaveConverter to zero out the signal when its instantaneous power drops below -40dB. This parameter is critical when working with FSK signals, or you will get a lot of noise between transmissions as the quadrature demodulator attempts to extract frequency information from the noise.

After demodulation, the recovered waveforms are displayed, one at a time, in the waveform display. Immediately after demodulation, the baseband waveform of the first transmission is displayed. To view other transmissions, simply enter the transmission number you'd like to see and press enter. Alternatively, you can click the '+' or '-' buttons to cycle to the next and previous waveform respectively.

You can navigate each waveform by clicking one of the five zoom buttons.

Below the waveform display controls, you can see the decoded transmission output. This will be blank until the decode operation has been executed.

Framing Tab



The framing tab is used to define the preamble and header for the transmissions, which both verifies the integrity of the transmission as well as prepares the transmission for extraction of the payload and

CRC.

At the top of the left-hand side you can see the results of the most recent decode operation, including the number of transmissions with valid preambles, valid encodings and valid CRCs.

Good Packets - The number of transmissions with no framing, encoding or CRC errors. Good packets also have payloads of the specified length (this length is given on the next tab)

Preamble Matches – The number of transmissions for which the preamble matches that entered into this tab.

Encoding Valid – The number of transmissions for which the payload is encoded in a valid manner. The encoding type is specified on the next tab.

CRC Pass – The number of transmissions for which the payload and the specified CRC are valid. For protocols with no specified CRC, transmissions are always considered to have passed. The CRC parameters are entered into the next tab.

The next group of parameters define the timing at the start of each transmission.

Preamble Definition – There are three different ways you can define your preamble.

> **Regular** – To use this definition your preamble must consist of a regular series of pulses followed by a low period called a "header."
>
> **Arbitrary** – If your preamble does not fit the definition above, you can define it explicitly with a Python list containing each of the transition timings in microseconds.
>
> **Pulse Count** – If your preamble has varying timings but a fixed number of transitions, you can use this mode, which skips timing checks altogether.

Preamble Low (us) – For each cycle of the preamble, this is the time in microseconds during which it is low.

Preamble High (us) - For each cycle of the preamble, this is the time in microseconds during which it is high.

Preamble Size – This is the number of full cycles that constitute a complete preamble.

Preamble Size 2 – For some transmissions, there may be two possible lengths for the preamble. For example, the first transmissions in a series may have 50 cycles of the preamble, while any successive transmissions will only have 10 cycles. If your preamble only has one length (which is most common), simply enter ***999*** for this.

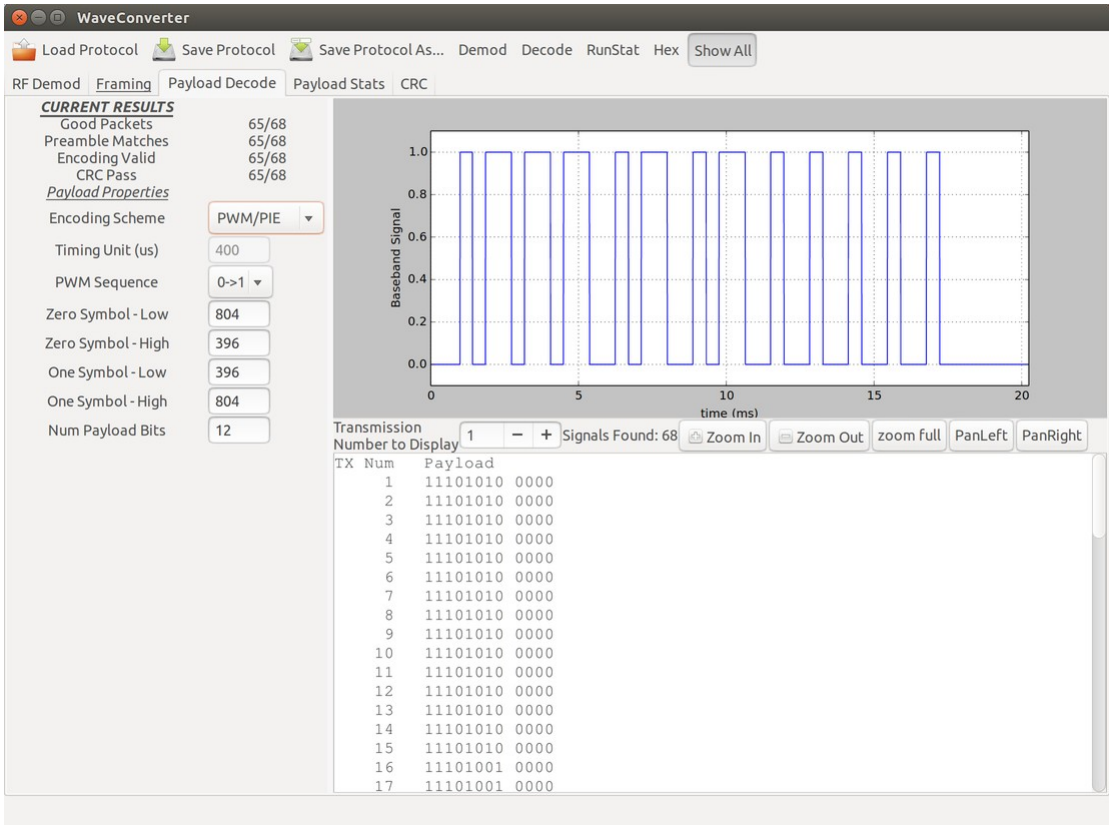
Header Size (us)– Some transmissions include a dead time (low logic level) after the preamble and before the payload. Enter the duration of this dead time here. If your transmission type contains no dead air, enter -***1***.

Arbitrary Preamble – You will enter the timings directly for your preamble in Python list form. For example, if a preamble consists of a 100us high time, followed by a 333us low time, followed by 150us high and 2000us low you would enter [100, 333, 150, 2000].

Pulse Count – When using the Pulse Count definition, you simply enter the number of transitions that comprise the preamble. Using the example for the arbitrary sequence above, you would simply enter 4.

Timing (%) - For WaveConverter to recognize any timings as valid, they must fall within this percentage of the specified timing.

Payload Decode Tab



Encoding Scheme – There are currently two options available for the encoding scheme:

> **Manchester** – This scheme requires you to provide a ***Timing Unit*** in microseconds. This is the smallest unit timing observed in the payload. The Symbol high and low times are not required for this mode.
>
> **Manchester Inverted** – NOT YET IMPLEMENTED
>
> **PWM/PIE** – Both of these modes encode data via distinct timing width. Pulse Width Modulation has a uniform symbol duration and encodes ones and zero by using different duty cycles. Pulse Interval Encoding also uses different pulse widths to encode the data, but the duty cycle is not uniform. Both of these schemes are defined by Symbol high and low times. No ***Timing Unit*** is required.

Timing Unit (us) – All signal timings in Manchester-encoded payloads are either one or two unit lengths. For this parameter, you will need to observe the baseband waveform and determine the shortest timing in the payload portion.

For PWM and PIE modes, there are distinct timings for the symbol representing a digital zero and for the symbol representing a digital one. To determine these values, simply observe the payload portion of the baseband waveform, you should see pulses of two different widths. Typically, the longer pulse represents a one and the shorter pulse a zero.

Each symbol is defined first by a low signal level, then by a high signal level.

PWM Sequence – Select whether each PWM symbol consists of a low time followed by a high time, or a high time followed by a low time.

Zero Symbol Low - For the shorter of the two pulse lengths, enter the low time duration into this parameter.

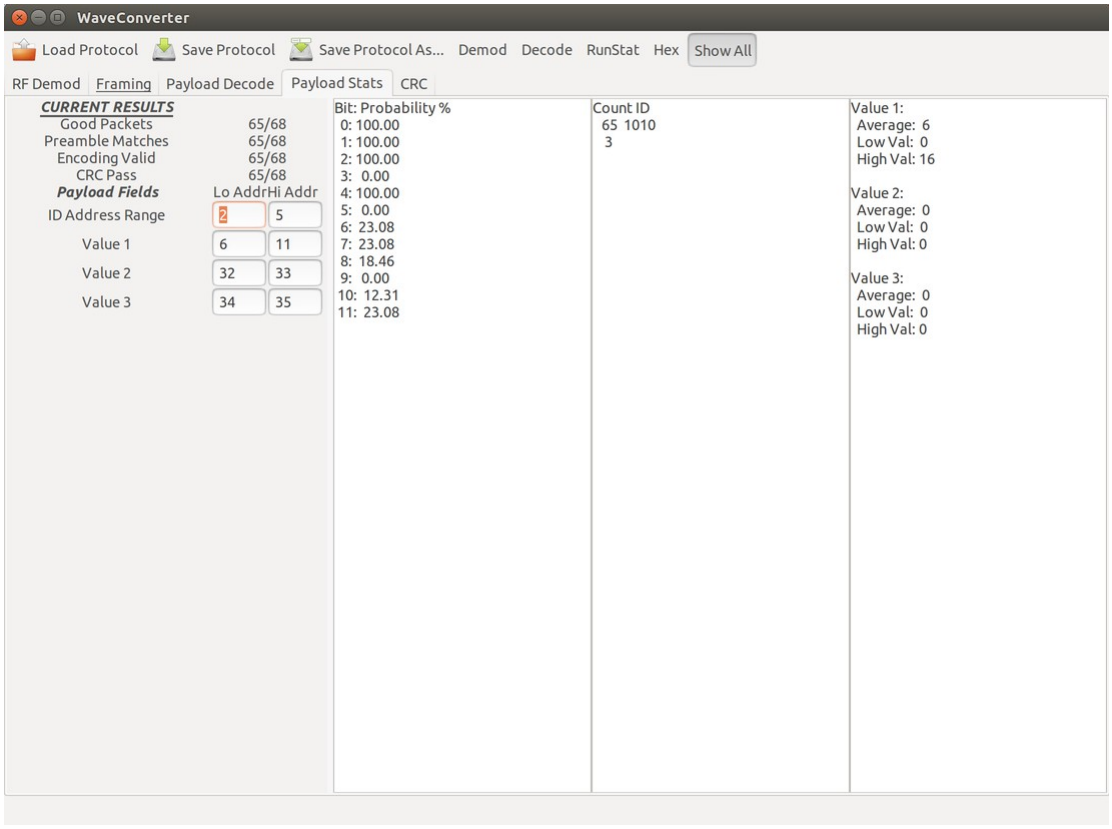
Zero Symbol High - For the shorter of the two pulse lengths, enter the duration of the pulse.

One Symbol Low - For the longer of the two pulse lengths, enter the low time duration into this parameter.

One Symbol High - For the longer of the two pulse lengths, enter the duration of the pulse.

Num Payload Bits – Enter the number of symbols in your payload

Payload Stats Tab



This tab provides statistical information about the transmissions you have decoded. To make full use of it, you must first define the bit boundaries of any fields.

Even if you don't enter these, however, you'll still be able to view the bit probabilities in the left-most pane of the display. These bit probabilities will help you determine the general function of the different parts of the payload. To derive useful information from this data, however, you will need to have a substantial number of transmissions or the statistical analysis will be invalid.

If you see a probability near 0 or 100%, these bits are likely used for some kind of identification field. If you suspect only one transmitter is sending data, then this could be the ID of the single transmitter. If you have multiple transmitters, you may see a number of consecutive bit probabilities with similar, or even identical values.

If you see several consecutive probabilities with a value nearly 50%, you are likely looking at a CRC or a set of encrypted payload data.

A string of bits with similar, but not identical probabilities often represents a measurement of some kind.

The following parameters will allow deeper analysis:

ID Address Range – Enter the bit addresses of the low and high extents of the payload that you

think represent an ID value. After clicking the RunStat button on the tool bar, the center pane will display the ID values residing at the specified address range along with how many times they occurred.

Value 1-3 Address Range – Enter the bit addresses of the low and high extents of the payload that you think represent measured values. After clicking RunStat, the right-most pane will display the range of values contained in those addresses, after first converting from binary to integer.

CRC Tab

Note: This tab is in work and has been included for those with extensive experience with CRCs or for those willing to look through the source code.

WaveConverter contains a CRC calculator that verifies the integrity of each received transmission. To make use of this, you must first identify the portion of the payload occupied by the CRC. You will then need to determine the CRC algorithm used by the transmitter. After entering the CRC information, go back and click the Decode button, and your "Current Results" will update based on the number of transmissions for which a correct CRC was computed.

CRC Length – The number of bits in the CRC polynomial

CRC Start Addr – The start address of the CRC contained in the transmission.

Addr for Data 0 – The address for the first bit over which the CRC is computed.

Addr for Data N-1 – The last bit address over which the CRC is computed.

CRC Polynomial (MSB 1st) – The CRC polynomial in the form of a Python list. For example the polynomial $x^3 + x^2 + 1$ would be rendered as [1, 1, 0, 1].

CRC Init (0 or 1) – The initial value of the "registers" used for CRC computation will be either all zeros or all ones.

Bit Ordering – The data will be fed into the CRC algorithm in one of three orders:

MSB->LSB – In order from the bit at Addr 0 to the bit at Addr N-1. (I know, this looks backwards...)

LSB->MSB – In reverse order, from the bit at Addr N-1 to the bit at Addr 0.

Reflected – The bytes are processed in MSB order, but each byte is individually reversed.

CRC Reverse Out – Whether to reverse the final CRC computation before checking it.

Final XOR – This is a Python list of 1s and 0s, equal in length to the CRC. Before final output (and after potential reversal), the CRC value will be XOR-d with this parameter.

CRC Pad – There are three padding modes for the input data to the CRC algorithm:

No Pad – Only the input data is applied to the algorithm

Pad to Byte – The input data is padded with zeros at the end such that its length is evenly divisible by the Pad Count below.

Absolute – The input data is padded with a fixed amount of zero, selected in the next parameter.

CRC Pad Count – You can select either 0, 8, 16 or 32.

User Interface – Command Line

WaveConverter also operates in command line mode so you can include it in more complex scripts of your own devising. There are also a few features that can only be accessed via the command line. The following command line options are available (both the short and long versions of each argument are given):

`-h, --help`

Shows a help message detailing WaveConverter usage and the command line options available.

`-q <IQ file name>, --iq <IQ file name>`

This flag must be followed by the name of the input RF file, otherwise known as an I-Q file.

`-b <baseband input file name>, --baseband <baseband input file name>`

This flag specifies an input baseband file name. You can generate this file by demodulating an IQ file yourself and gnuradio and saving the digital output to a file. This file must be composed of single byte samples, with each equal to 0x00 or 0x01. You should only supply an input baseband file OR and input IQ file, not both.

`-n <baseband output file name>, --bb_save <baseband output file name>`

Allows you to specify a file name to which the demodulated waveform can be saved. You can demodulate a very large IQ file, producing a vastly smaller baseband file. You can then use this baseband file for future WaveConverter inputs.

`-o <output file>, --output <output file>`

Specifies file name for WaveConverter output. After execution this file will contain the payloads of each transmission as well as the basic statistical analysis.

`-s <sample rate>, --samp_rate <sample rate>`

Supplies the sample rate of either the input IQ file or the input baseband file. This is an integer value in Hertz and must be greater than 100 kHz. The best practice is to use an IQ file with the naming convention defined above, in which case this entry is not necessary.

`-c <center frequency>, --center_freq <center frequency>`

Supplies the center frequency at which the input IQ file was captured. This is an integer value in Hertz. The best practice is to use an IQ file with the naming convention defined above, in which case this entry is not necessary.

`-p <protocol number>, --protocol <protocol number>`

Specifies the number of the protocol to use for demodulation and decoding. The protocols are numbered from 1 to n. Entering “0” for the protocol number will produce a list of the available protocols.

`-v, --verbose`

Tells WaveConverter to output additional information to stdout, mostly useful only to developers.

`-x, --hex_out`

By default WaveConverter outputs payloads and certain statistical information in binary form. This flag specifies hex output instead.

`-z, --hide_bad`

By default the data from all transmissions are displayed in the output file and included in the statistical processing. This flag causes bad transmissions (bad preambles, encoding or CRCs) to be omitted.

`-i <glitch filter count>, --glitch_count <glitch filter count>`

A glitch filter is applied to the demodulated waveform before the baseband signal is generated. This glitch filter has a sample rate of 100 kHz. The glitch filter will “smooth out” any transitions shorter than the time specified here. Each count of the glitch filter is equal to 10 microseconds, so providing a value a “2” will result in the elimination of all baseband transitions shorter than 20 microseconds.

`-f <frequency>, --freq <frequency>`

The frequency, in Hertz, of the actual transmission contained in the IQ file.

`-l <threshold value>, --threshold <threshold value>`

This value is used only for OOK modulated signals. It is the value of the signal magnitude above which

a one is signaled and below which a zero is signaled. You may need to visualize the signal in a gnuradio GUI sink to determine this value.

`-t <time between transmissions>, --time_between_tx <time btw tx>`

This is the minimum value (in microseconds) of the dead air time between transmissions. Currently WaveConverter assumes that each transmission contains a deterministic amount of dead air before and after. WaveConverter uses this time to separate the demodulated waveform into individual transmissions. If your transmission does not include this dead time, you may have difficulty demodulating and decoding it. Work is underway to remedy this shortcoming.

`-e <maximum timing error>, --timing_error <maximum timing error>`

You can specify a maximum percentage of allowable timing error. Any timing deviance between the captured waveform and the specified protocol will be considered correct if it is less than this value, as a percentage of the protocol value. This value is expressed in terms of integer percentage (10), not as a floating point value normalized to unity (0.10).

`-g, --gui`

Opens the GUI rather than runs in command line mode.

`-d, --db`

Builds a new sqllite database. This should only be used by developers.

`-r <protocol number>, --export_protocol <protocol number>`

You can export protocols to a text file for sharing with others and to send to the developer for inclusion in future releases of the protocol database. To do so you must supply the protocol number you wish to export and the file name (with -o) to which you wish to write the protocol. For example:

```
./waveconverter.py -r 1 -o my_protocol.txt
```

`-j <protocol text file>, --import_protocol <protocol text file>`

You can import protocols from a text file that have been shared with you. For example:

```
./waveconverter.py -j my_protocol.txt
```