API for RF receivers including ThinkRF RTSA platforms

ThinkRF Corporation

Mar 29, 2019

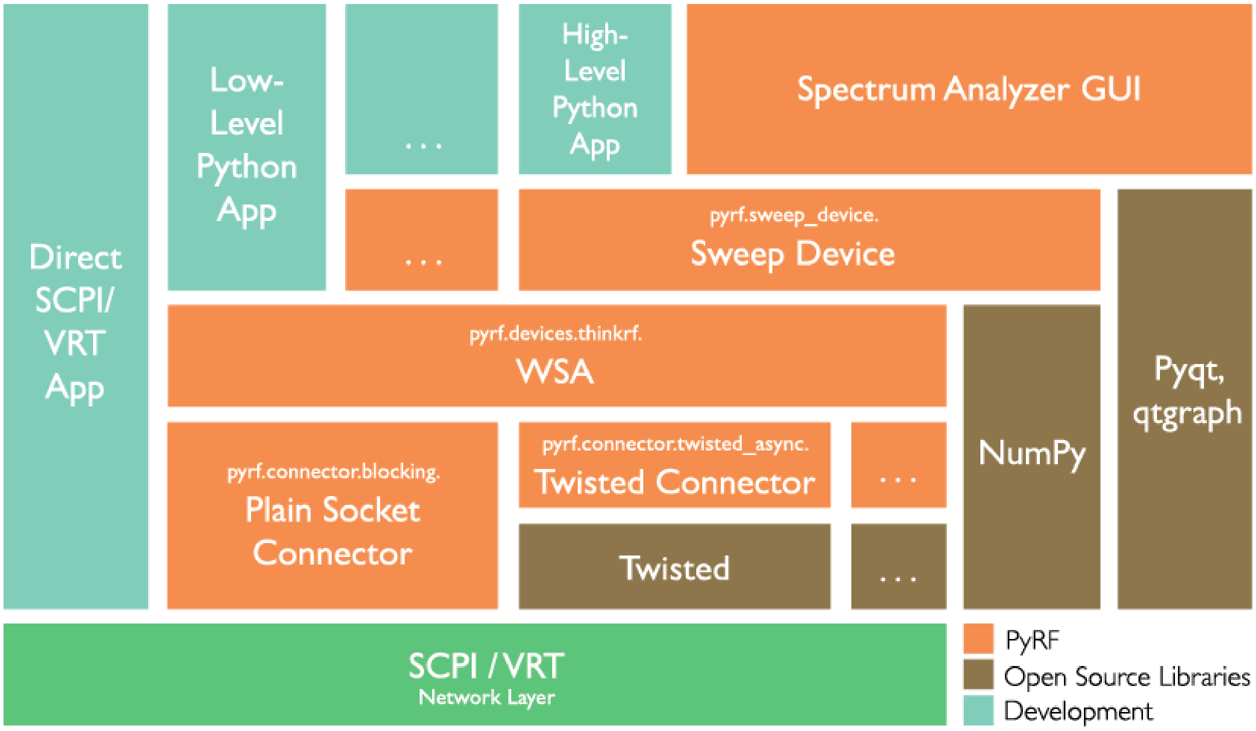
Contents

CHAPTER 1

Overview

PyRF is an openly available, comprehensive development environment for wireless signal analysis. PyRF handles the low-level details of configuring a device, real-time data acquisition and signal processing, allowing you to concentrate on your analysis solutions. Hence, it enables rapid development of powerful applications that leverage the new generation of measurement-grade software-defined radio technology, such as ThinkRF Real-Time Spectrum Analysis Software.

PyRF is built on the Python Programming Language (v2.7) and includes feature-rich libraries, examples including visualization applications and source code, all specific to the requirements of signal analysis. It is openly available, allowing commercialization of solutions through BSD open licensing and offering device independence via standard hardware APIs.


Direct SCPI/ VRT App
Low-Level Python App
. . .
High-Level Python App
Spectrum Analyzer GUI
. . .
pyrf.sweep_device.
Sweep Device
Pyqt, qtgraph
pyrf.devices.thinkrf.
WSA
NumPy
pyrf.connector.blocking.
Plain Socket Connector
pyrf.connector.twisted_async.
Twisted Connector
. . .
Twisted
. . .
SCPI / VRT
Network Layer
PyRF
Open Source Libraries
Development

CHAPTER 2

Table of Contents

2.1 Manual

2.1.1 Installation

This section provides information on how to install the required python packages.

Note: Python v2.7.x is the required version for PyRF, not v3.x or higher.

Windows Setup

1. Set-up Python v2.7

- Install Python v2.7 from https://www.python.org/downloads/release/python-2715/
- Add to the windows PATH: **C:\Python27** and **C:\Python27\Scripts**

2. Install Dependencies

These installation steps make use of pip software to install required libraries. Open a command prompt window and type **pip**, if a help menu appears, **pip** is already in your system. If **pip** has not yet been installed, follow these instructions:

- Download get-pip.py (right mouse click and save)
- Open a command prompt window, navigate to get-pip.py and run:

```
python get-pip.py
```

- Now use **pip** to install the dependencies by typing into the command prompt window:

```
pip install numpy scipy pyside==1.2.2 pyqtgraph twisted zope.interface␣
↪setuptools pywin32
pip install netifaces
```

Notes:

- **pySide v1.2.2** is needed, not the latest
- When installing **netifaces**, **MS Visual C++ 9.0** is required, follow the recommended instruction, such as `error: Microsoft Visual C++ 9.0 is required. Get it from http://aka.ms/vcpython27`

- To install **qtreactor**, choose one of the following option:
- If you have git, run:

```
pip install -e git://github.com/pyrf/qtreactor.git#egg=qtreactor
```

- Otherwise, download qtreactor-pyrf-1.0 to your computer, unzip and then go into the extracted folder in a command prompt window and type:

```
python setup.py install
```

Continue with *PyRF Installation* below.

Linux Setup

These instructions are tested on Debian/Ubuntu system, equivalent **apt-get** command might be needed for your system.

- Install **python2.7** package if not already available in your system
- Install required libraries (sudo privilege might be needed):

```
apt-get install pip
pip install numpy scipy pyside==1.2.2 pyqtgraph twisted netifaces zope.
↪interface setuptools
pip install -e git://github.com/pyrf/qtreactor.git#egg=qtreactor
```

- Or install dependencies from source:

```
apt-get install qt-sdk python-dev cmake libblas-dev libatlas-dev␣
↪liblapack-dev gfortran
export BLAS=/usr/lib/libblas/libblas.so
export ATLAS=/usr/lib/atlas-base/libatlas.so
export LAPACK=/usr/lib/lapack/liblapack.so
pip install -r requirements.txt
pip install pyside==1.2.2
```

Continue with *PyRF Installation* below.

PyRF Installation

- Download the development version by either:

- Using git and run:

```
git clone git://github.com/pyrf/pyrf.git
```

- Or download a stable release here and extract
- Navigate to *pyrf* directory (`cd pyrf`), run:

```
python setup.py install
```

2.1.2 PyRF API for ThinkRF RTSA Products

`pyrf.devices.thinkrf.WSA` is the class that provides access to ThinkRF Real-Time Spectrum Analyzers (RTSA, also formerly known as WSA) devices. Its methods closely match the SCPI Command Set described in the product's Programmer's Guide (available on ThinkRF Resources).

There are simple examples illustrating usage of this API under the *examples* directory included with the source code directory. Some are mentioned in the *Examples* section of this document .

This API may be used in a **blocking** mode (the default) or in an **asynchronous** mode with using the Twisted python library.

In **blocking** mode, all methods that read from the device will wait to receive a response before returning.

In **asynchronous** mode, all methods will send their commands to the device and then immediately return a Twisted Deferred object. If you need to wait for the response or completion of this command, you can attach a callback to the Deferred object and the Twisted reactor will call it when ready. You may choose to use Twisted's `inlineCallbacks` function decorator to write Twisted code that resembles synchronous code by yielding the Deferred objects returned from the API.

To use the **asynchronous**, when a WSA instance of a device (ex. `dut = WSA()`) is created, you must pass a `pyrf.connectors.twisted_async.TwistedConnector` instance as the connector parameter, as shown in *discovery.py / twisted_discovery.py*

2.2 Reference

2.2.1 pyrf.devices

.thinkrf

class `pyrf.devices.thinkrf.`**`WSA`**(*connector=None*)

Interface for ThinkRF's R5500, R5700, and WSA5000 (EOL).

Parameters **`connector`** – Connector object to use for SCPI/VRT connections, defaults to a new `PlainSocketConnector` instance

`connect()` must be called before other methods are used.

Note: The following methods will either block then return a result or if you passed a `TwistedConnector` instance to the constructor, they will immediately return a Twisted Deferred object.

The methods are grouped and listed by functionalities.

Connection Related Methods:

connect (*host*)
Connect to an RTSA (aka WSA).
Parameters **host** (`str`) – the hostname or IP to connect to
Usage:

```
dut.connect('123.456.789.1')
```

disconnect ()
Close a connection to an RTSA (aka WSA).

async_connector ()
Return True if the connector being used is asynchronous

set_async_callback (*callback*)
Set the asynchronous callback for a function when the device receives a VRT packet. Use with Twisted setup. :param callback: callback to set. Set to *None* to disable receiving packets.

Direct SCPI commands:

scpiget (*cmd*)
Send a SCPI *query* command and wait for the response.

This is the lowest-level interface provided. See the product's Programmer's Guide for the SCPI commands available.
Parameters **cmd** (`str`) – the SCPI command to send
Returns the response output from the box if any

scpiset (*cmd*)
Send a SCPI command of set type (i.e. not query command).

This is the lowest-level interface provided. See the product's Programmer's Guide for the SCPI commands available.
Parameters **cmd** (`str`) – the command to send

errors ()
Flush and return the list of errors from past commands sent to the RTSA. An empty list is returned when no errors are present.

Device System Related:

id()
Returns the RTSA's identification information string.
Returns "<Manufacturer>,<Model>,<Serial number>,<Firmware version>"

reset ()
Resets the RTSA to its default configuration. It does not affect the registers or queues associated with the IEEE mandated commands.

locked (*modulestr*)
This command queries the lock status of the RF VCO (Voltage Control Oscillator) in the Radio Front End (RFE) or the lock status of the PLL reference clock in the digitizer card.
Parameters **modulestr** (`str`) – 'VCO' for rf lock status, 'CLKREF' for ref clock lock status
Returns True if locked

Data Acquisition Related Methods:

- Get permission:

has_data()

Check if there is VRT data to read.

Returns True if there is a packet to read, False if not

request_read_perm()

Acquire exclusive permission to read data from the RTSA.

Returns True if allowed to read, False if not

have_read_perm()

Check if we have permission to read data from the RTSA.

Returns True if allowed to read, False if not

- Set capture size for stream or block mode capture:

ppb(*packets=None*)

This command sets the number of IQ packets in a capture block

Parameters **packets** (*int*) – the number of packets for a block of capture, or *None* to query

Returns the current ppb value if the packets parameter is *None*

spp(*samples=None*)

This command sets or queries the number of Samples Per [VRT] Packet (SPP).

The upper bound of the *samples* is limited by the VRT's 16-bit packet size field. However, since the SPP must be a multiple of 32, the maximum is thus limited by (2**16 - 32) or 65504.

Parameters **samples** (*int*) – the number of samples in a VRT packet (256 to 65504, a multiple of 32), or *None* to query

Returns the current spp value if the samples parameter is *None*

- Stream setup

stream_status()

This query returns the current running status of the stream capture mode.

Returns 'RUNNING' or 'STOPPED'

stream_start(*stream_id=None*)

This command begins the execution of the stream capture. It will also initiate data capturing. Data packets will be streamed (or pushed) from the RTSA whenever data is available.

Parameters **stream_id** (*int*) – optional unsigned 32-bit stream identifier

stream_stop()

This command stops the stream capture. After receiving the command, the RTSA system will stop when the current capturing VRT packet is completed. Recommend calling `flush()` after stopping.

- Sweep setup:

sweep_add(*entry*)

Add a sweep entry to the sweep list

Parameters **entry** (*pyrf.sweepDevice.sweepSettings*) – the sweep entry settings to add to the list

sweep_clear()

Remove all entries from the sweep list.

sweep_iterations(*count=None*)
Set or query the number of iterations to loop through a sweep list.
Parameters **count** (*int*) – the number of iterations, 0 for infinite, or *None* to query
Returns the current number of iterations if count is *None*

sweep_read(*index*)
Read a sweep entry at the given sweep *index* from the sweep list.
Parameters **index** – the index of the entry to read
Returns settings of that sweep entry
Return type *pyrf.config.SweepEntry*

sweep_start(*start_id=None*)
Start the sweep engine with an optional ID.
Parameters **start_id** (*int*) – An optional 32-bit ID to identify the sweep

sweep_stop()
Stop the sweep engine. Recommend calling `flush()` after stopping.

- VRT data acquisition related methods:

capture(*spp*, *ppb*)
This command will **start** the single block capture of *ppb* packets of *spp* samples in each packet. The data within a single block capture trace is continuous from one packet to the other, but not necessary between successive block capture commands issued. Used for **stream** or **block** capture mode. To read data back, use `read()` method. See show_i_q.py as an example.
Parameters
- **spp** (*int*) – the number of samples in a VRT packet
- **ppb** (*int*) – the number of packets in a block of capture

capture_mode()
This command queries the current capture mode
Returns the current capture mode

raw_read(*num*)
Raw read of VRT socket data of *num* bytes from the RTSA.
Parameters **num** (*int*) – the number of bytes to read
Returns bytes

read()
Read and return a single **parsed** VRT packet from the RTSA, either context or data.

read_data(*spp*)
Read and return a data packet, as well as computed power spectral density data, of *spp* (samples per packet) size, the associated context info and the computed power spetral data. If a block of data is requested (such as *ppb* is more than 1), loop through this function to retreive all data. See also data other capture functions: `pyrf.util.capture_spectrum()`, `pyrf.capture_device.capture_time_domain()`
Parameters **spp** (*int*) – the number of samples in a VRT packet (256 to 65504) in a multiple of 32
Returns data, context dictionary, and power spectral data array

abort()
This command will cause the RTSA to stop the data capturing, whether in the

manual trace block capture, triggering or sweeping mode. The RTSA will be put into the manual mode; in other words, process such as streaming, trigger and sweep will be stopped. The capturing process does not wait until the end of a packet to stop, it will stop immediately upon receiving the command.

flush()

This command clears the RTSA's internal data storage buffer of any data that is waiting to be sent. Thus, it is recommended that the flush command should be used when switching between different capture modes to clear up the remnants of captured packets.

eof()

Check if the VRT stream has closed.

Returns True if no more data, False if more data

Device Configuration Methods for Non-Sweep Setup:

attenuator(*atten_val=None*)

This command enables, disables or queries the RTSA's RFE attenuation.

Parameters **atten_val** – see Programmer's Guide for the attenuation value to use for your product; *None* to query

Returns the current attenuation value if *None* is used

decimation(*value=None*)

This command sets or queries the rate of decimation of samples in a trace capture. The supported rate is 4 - 1024. When the rate is set to 1, no decimation is performed on the trace capture.

Parameters **value** (*int*) – new decimation value (1 or 4 - 1024); *None* to query

Returns the decimation value if *None* is used

freq(*freq=None*)

This command sets or queries the tuned center frequency of the RTSA.

Parameters **freq** (*int*) – the center frequency in Hz (range vary depnding on the product model); *None* to query

Returns the frequency in Hz if *None* is used

fshift(*shift=None*)

This command sets or queries the frequency shift value.

Parameters **freq** (*int*) – the frequency shift in Hz (0 - 125 MHz); *None* to query

Returns the amount of frequency shift if *None* is used

hdr_gain(*gain=None*)

This command sets or queries the HDR gain of the receiver. The gain has a range of -10 to 30 dB.

Parameters **gain** (*int*) – float between -10 and 30 to set; *None* to query

Returns the hdr gain in dB if *None* is used

iq_output_path(*path=None*)

This command sets or queries the RTSA's current IQ path. It is not applicable to R5700.

Parameters **path** (*str*) – 'DIGITIZER', 'CONNECTOR', 'HIF', or *None* to query

Returns the current IQ output path type if *None* is used

pll_reference(*src=None*)
This command sets or queries the RTSA's PLL reference source

Parameters **src** (*str*) – 'INT', 'EXT', 'GPS' (when available with the model) or *None* to query

Returns the current PLL reference source if *None* is used

psfm_gain(*gain=None*)
This command sets or queries one of the Pre-Select Filter Modules's (PSFM) gain stages.

Parameters **gain** (*str*) – sets the gain value to 'high', 'medium', 'low', or *None* to query

Returns the RF gain value if *None* is used

Usage:

```
dut.psfm_gain('HIGH')
```

rfe_mode(*mode=None*)
This command sets or queries the RTSA's Receiver Front End (RFE) mode of operation.

Parameters **mode** (*str*) – 'ZIF', 'DD', 'HDR', 'SHN', 'SH', or *None* to query

Returns the current RFE mode if *None* is used

trigger(*settings=None*)
This command sets or queries the type of trigger event. Setting the trigger type to "NONE" is equivalent to disabling the trigger execution; setting to any other type will enable the trigger engine.

Parameters **settings** (*dictionary*) – the new trigger settings; *None* to query

Returns the trigger settings if *None* is used

apply_device_settings(*settings*, *force_change=False*)
This command takes a dict of device settings, and applies them to the RTSA

Parameters

- **settings** (*dict*) – dict containing device's settings such as attenuation, decimation, etc
- **force_change** (*bool*) – to force the change update or not

DSP and Data Processing Related Methods:

measure_noisefloor(*rbw=None*, *average=1*)
Returns a power level that represents the top edge of the noisefloor

Parameters

- **rbw** (*int*) – rbw of spectral capture (Hz) (will round to nearest native RBW) or None

- **average** (*int*) – number of capture iterations

Returns noise_power

peakfind(*n=1*, *rbw=None*, *average=1*)
Returns frequency and the power level of the maximum spectral point computed using the current settings, Note this function disables

Parameters

- **n** (*int*) – determine the number of peaks to return
- **rbw** (*int*) – rbw of spectral capture (Hz) (will round to nearest native RBW) or None
- **average** (*int*) – number of capture iterations

Returns [(peak_freq1, peak_power1), (peak_freq2, peak_power2) , . . . , (peak_freqn, peak_powern)]

Data Recording Related Methods:

inject_recording_state(*state*)
Inject the current RTSA state into the recording stream when the next capture is received. Replaces previous data if not yet sent.

set_recording_output(*output_file=None*)
Dump a recording of all the received packets to output_file

Device Discovery Functions:

pyrf.devices.thinkrf.**discover_wsa**(*wait_time=0.125*)

pyrf.devices.thinkrf.**parse_discovery_response**(*response*)
This function parses the RTSA's raw discovery response

Parameters **response** – The RTSA's raw response to a discovery query

Returns Return (model, serial, firmware version) based on a discovery response message

2.2.2 pyrf.connectors

.blocking

class pyrf.connectors.blocking.**PlainSocketConnector**
This connector makes SCPI/VRT socket connections using plain sockets, of blocking type.

pyrf.connectors.blocking.**socketread**(*socket*, *count*, *flags=None*)
Retry socket read until *count* amount of data received, like reading from a file.

Parameters

- **count** (*int*) – the amount of data received
- **flags** – socket.recv() related flags

.twisted_async

class pyrf.connectors.twisted_async.**TwistedConnector**(*reactor*, *vrt_callback=None*)
A connector that makes SCPI/VRT connections asynchronously using Twisted method.

Parameters

- **reactor** – a twisted reactor, (ex: "from twisted.internet import reactor")
- **vrt_callback** (*callback*) – A callback may be assigned to *vrt_callback* that will be called with VRT packets as they arrive. When *vrt_callback* is None (the default), arriving packets will be ignored.

exception `pyrf.connectors.twisted_async.`**`TwistedConnectorError`**

class `pyrf.connectors.twisted_async.`**`VRTClient`**(*receive_callback*)

A Twisted protocol for the VRT connection.

Parameters **receive_callback** – a function that will be passed a vrt DataPacket or ContextPacket when it is received

2.2.3 pyrf.capture_device

class `pyrf.capture_device.`**`CaptureDevice`**(*real_device*, *async_callback=None*, *device_settings=None*)

Virtual device that returns power levels generated from a single data packet

Parameters

- **real_device** – the device that will be used for capturing data, typically a `pyrf.thinkrf.WSA` instance.
- **async_callback** – callback to use for async operation (not used if *real_device* is using a blocking `PlainSocketConnector`)
- **device_settings** – initial device settings to use, passed to `pyrf.capture_dvice.CaptureDevice.configure_device()` if given

capture_time_domain(*rfe_mode*, *freq*, *rbw*, *device_settings=None*, *min_points=256*, *force_change=False*)

Initiate a capture of raw time domain IQ or I-only data

Parameters

- **rfe_mode** (*str*) – radio front end mode, e.g. 'ZIF', 'SH', ...
- **freq** (*int*) – center frequency in Hz to set
- **rbw** (*float*) – the resolution bandwidth (RBW) in Hz of the data to be captured (output RBW may be smaller than requested)
- **device_settings** (*dict or None*) – rfe_mode, freq, decimation, fshift and other device settings
- **min_points** (*int*) – smallest number of data points per capture from the device
- **force_change** (*bool*) – force the configuration to apply device_settings changes or not

Returns (fstart, fstop, data) where fstart & fstop are frequencies in Hz & data is a list

configure_device(*device_settings*, *force_change=False*)

Configure the device settings on the next capture

Parameters

- **device_settings** (*dict*) – rfe mode, attenuation, decimation and other device settings
- **force_change** (*bool*) – force the configuration to apply device_settings changes or not

exception pyrf.capture_device.**CaptureDeviceError**

2.2.4 pyrf.sweep_device

class pyrf.sweep_device.**SweepDevice**(*real_device*, *async_callback=None*)

Virtual device that generates power spectrum from a given frequency range by sweeping the frequencies with a real device and piecing together the FFT results.

Parameters

- **real_device** – the RF device that will be used for capturing data, typically a *pyrf.devices.thinkrf.WSA* instance.
- **async_callback** – a callback to use for async operation (not used if *real_device* is using a blocking PlainSocketConnector)

capture_power_spectrum(*fstart*, *fstop*, *rbw*, *device_settings=None*, *mode='SH'*, *continuous=False*)

Initiate a data capture from the *real_device* by setting up a sweep list and starting a single sweep, and then return power spectral density data along with the **actual** sweep start and stop frequencies set (which might not be exactly the same as the requested *fstart* and *fstop*).

Note: This function does not pipeline, and if the last sweep isn't received before starting a new one, it will generate a failure.

Parameters

- **fstart** (*int*) – sweep starting frequency in Hz
- **fstop** (*int*) – sweep ending frequency in Hz
- **rbw** (*float*) – the resolution bandwidth (RBW) in Hz of the data to be captured (output RBW may be smaller than requested)
- **device_settings** (*dict*) – attenuation and other device settings
- **mode** (*str*) – sweep mode, 'ZIF', 'SH', or 'SHN'
- **continuous** (*bool*) – set sweep to be continuously or not (once only)

Returns fstart, fstop, power_data

exception pyrf.sweep_device.**SweepDeviceError**

Exception for the sweep device to state an error() has occured

class pyrf.sweep_device.**SweepPlanner**(*dev_prop*)

An object that plans a sweep based on given paramaters.

Parameters **dev_prop** (*dict*) – the sweep device properties

class pyrf.sweep_device.**SweepSettings**

An object used to keep track of the sweep settings

2.2.5 pyrf.config

class pyrf.config.**SweepEntry**(*fstart=2400000000*, *fstop=2400000000*, *fstep=100000000*, *fshift=0*, *decimation=0*, *gain='vlow'*, *ifgain=0*, *hdr_gain=-10*, *spp=1024*, *ppb=1*, *trigtype='none'*, *dwell_s=0*, *dwell_us=0*, *level_fstart=50000000*, *level_fstop=10000000000*, *level_amplitude=-100*, *attenuator=30*, *rfe_mode='SH'*)

Sweep entry setup for `pyrf.devices.thinkrf.WSA.sweep_add()`

Parameters

- **fstart** (*int*) – starting frequency in Hz
- **fstop** (*int*) – ending frequency in Hz
- **fstep** (*int*) – frequency step in Hz
- **fshift** (*int*) – the frequency shift in Hz
- **decimation** (*int*) – the decimation value (0 or 4 - 1023)
- **gain** (*str*) – the RF gain value ('high', 'medium', 'low' or 'vlow')
- **ifgain** (*int*) – the IF gain in dB (-10 - 34)

  Note: parameter is deprecated, kept for a legacy device

- **hdr_gain** (*int*) – the HDR gain in dB (-10 - 30)
- **spp** (*int*) – samples per packet (256 - max, a multiple of 32) that fit in one VRT packet
- **ppb** (*int*) – data packets per block
- **dwell_s** (*int*) – dwell time seconds
- **dwell_us** (*int*) – dwell time microseconds
- **trigtype** (*str*) – trigger type ('none', 'pulse' or 'level')
- **level_fstart** (*int*) – level trigger starting frequency in Hz
- **level_fstop** (*int*) – level trigger ending frequency in Hz
- **level_amplitude** (*float*) – level trigger minimum in dBm
- **attenuator** – vary depending on the product
- **rfe_mode** (*str*) – RFE mode to be used, such as 'SH', 'SHN', 'DD', etc.

Returns a string list of the sweep entry's settings

class pyrf.config.**TriggerSettings**(*trigtype='NONE'*, *fstart=None*, *fstop=None*, *amplitude=None*)

Trigger settings for `pyrf.devices.thinkrf.WSA.trigger()`.

Parameters

- **trigtype** (*str*) – "LEVEL", "PULSE", or "NONE" to disable
- **fstart** (*int*) – trigger starting frequency in Hz
- **fstop** (*int*) – trigger ending frequency in Hz

- **amplitude** (*float*) – minimum level for trigger in dBm

Returns a string in the format: TriggerSettings(trigger type, fstart, fstop, amplitude)

exception pyrf.config.**TriggerSettingsError**
Exception for the trigger settings to state an error() has occured

2.2.6 pyrf.numpy_util

pyrf.numpy_util.**calculate_channel_power**(*power_spectrum*)
Return a dBm value representing the channel power of the input power spectrum.

Parameters **power_spectrum** (*list*) – an array containing the power spectrum to be used for the channel power calculation

Returns the channel power result

pyrf.numpy_util.**calculate_occupied_bw**(*pow_data*, *span*, *occupied_perc*)
Return the occupied bandwidth of a given spectrum, and occupied percentage

Parameters

- **pow_data** (*list*) – spectral data to be analyzed
- **span** (*int*) – span of the given spectrum, in Hz
- **occupied_perc** (*float*) – Percentage of the power to be measured

Returns float value of the occupied bandwidth

pyrf.numpy_util.**calibrate_time_domain**(*power_spectrum*, *data_pkt*)
Return a list of the calibrated time domain data

Parameters

- **power_spectrum** (*list*) – spectral data of the time domain data
- **data_pkt** (pyrf.vrt.DataPacket) – a RTSA VRT data packet

Returns a list containing the calibrated time domain data

pyrf.numpy_util.**compute_fft**(*dut*, *data_pkt*, *context*, *correct_phase=True*, *iq_correction_wideband=True*, *hide_differential_dc_offset=True*, *convert_to_dbm=True*, *apply_window=True*, *apply_spec_inv=True*, *apply_reference=True*, *ref=None*, *decimation=1*)
Return an array of dBm values by computing the FFT of the passed data and reference level.

Parameters

- **dut** (pyrf.devices.thinkrf.WSA) – WSA device
- **data_pkt** (pyrf.vrt.DataPacket) – packet containing samples
- **context** (*dict*) – context values, such as 'bandwidth', 'reflevel', etc.
- **correct_phase** (*bool*) – apply phase correction for captures with IQ data or not
- **iq_correction_wideband** (*bool*) – apply wideband IQ correction or not
- **hide_differential_dc_offset** (*bool*) – mask the differential DC offset present in captures with IQ data or not

- **convert_to_dbm** (*bool*) – convert the output values to dBm or not
- **apply_window** (*bool*) – apply windowing to FFT function or not
- **apply_spec_inv** (*bool*) – apply spectral inversion to the FFT bin or not. *Recommend to leave as default*
- **apply_reference** (*bool*) – apply reference level correction or not
- **ref** (*float*) – a reference value to apply to the noise level
- **decimation** (*int*) – the decimation value (1, 4 - 1024)

Returns numpy array of spectral data in dBm, as floats

2.2.7 pyrf.util

pyrf.util.**capture_spectrum**(*dut*, *rbw=None*, *average=1*, *dec=1*, *fshift=0*)

Returns the spectral data, and the usable start and stop frequencies corresponding to the RTSA's current configuration

Parameters

- **rbw** (*int*) – rbw of spectral capture (Hz) (will round to nearest native RBW)
- **average** (*int*) – number of capture iterations
- **dec** (*int*) – decimation factor applied
- **fshift** (*int*) – the fshift applied, in Hz

Returns (fstart, fstop, pow_data) where pow_data is a list

pyrf.util.**read_data_and_context**(*dut*, *points=1024*)

Initiate capture of one VRT data packet, wait for and return data packet and collect preceeding context packets.

Parameters **points** (*int*) – Number of data points to capture

Returns (data_pkt, context_values)

Where *context_values* is a dict of {field_name: value} items from all the context packets received.

2.2.8 pyrf.vrt

class pyrf.vrt.**ContextPacket**(*packet_type*, *count*, *size*, *tmpstr*, *has_timestamp*)

A Context Packet received from *pyrf.devices.thinkrf.WSA.read()*. See VRT section of the product's Programmer's Guide for more information.

Parameters

- **packet_type** – VRT packet type
- **count** – VRT packet counter (see VRT protocol)
- **size** (*int*) – The VRT packet size, less headers and trailer words
- **tmpstr** – hold the raw data for parsing
- **has_timestamp** (*bool*) – to indicate timestamp is available with the packet

fields

a dict containing field names and values from the packet

is_context_packet(*ptype=None*)

Parameters **ptype** (*str*) – "Receiver", "Digitizer" or None for any packet type

Returns True if this packet matches the *ptype* passed

is_data_packet()

To indicate this VRT packet is not of data type as it's a ContextPacket

Returns False

class pyrf.vrt.**DataPacket**(*count*, *size*, *stream_id*, *tsi*, *tsf*, *payload*, *trailer*)

A Data Packet received from `pyrf.devices.thinkrf.WSA.read()`

data

a `pyrf.vrt.IQData` object containing the packet data

is_context_packet(*ptype=None*)

Returns False

is_data_packet()

Returns True

class pyrf.vrt.**IQData**(*binary_data*)

Data Packet values as a lazy collection of (I, Q) tuples read from *binary_data*.

This object behaves as an immutable python sequence, e.g. you may do any of the following:

```
points = len(iq_data)

i_and_q = iq_data[5]

for i, q in iq_data:
    print i, q
```

numpy_array()

Return a numpy array of I, Q values for this data

exception pyrf.vrt.**InvalidDataReceived**

pyrf.vrt.**vrt_packet_reader**(*raw_read*)

Read a VRT packet, parse it and return an object with its data.

Implemented as a generator that yields the result of the passed *raw_read* function and accepts the value sent as its data.

Parameters **raw_read** (*list*) – VRT packet of raw data (bytes)

2.3 Examples

This section explains **some** of the examples included with the PyRF source code.

Typical Usage:

```
python <example_file>.py [device_IP_when_needed]
```

2.3.1 discovery.py / twisted_discovery.py

- discovery.py
- twisted_discovery.py

These examples detect RTSA devices on the local network.

Example output:

```
R5700-427 180601-661 1.5.0 10.126.110.133
R5500-408 171212-007 1.5.0 10.126.110.123
R5500-418 180522-659 1.4.8 10.126.110.104
```

2.3.2 show_i_q.py / twisted_show_i_q.py

These examples connect to a device of IP specified on the command line, tunes it to a center frequency of 2.450 MHz then reads and displays one capture of 1024 i, q values.

- show_i_q.py
- twisted_show_i_q.py

Example output (truncated):

```
0,-20
-8,-16
0,-24
-8,-12
0,-32
24,-24
32,-16
-12,-24
-20,0
12,-32
32,-4
0,12
-20,-16
-48,16
-12,12
0,-36
4,-12
```

2.3.3 pyqtgraph_plot_single_capture.py / pyqtgraph_plot_block.py

These examples connect to a device of IP specified on the command line, tunes it to a center frequency, then continually capture and display the computed spectral data using pyqtgraph.

- pyqtgraph_plot_single_capture.py
- pyqtgraph_plot_block.py

2.3.4 pyqtgraph_plot_sweep.py

This example connects to a device of IP specified on the command line, makes use of sweep_device.py to perform a single sweep entry monitoring and plots computed spectral results using pyqtgraph.

- pyqtgraph_plot_sweep.py

2.3.5 matplotlib_plot_sweep.py

This example connects to a device specified on the command line, and plots a large sweep of the spectrum using NumPy and matplotlib.

- matplotlib_plot_sweep.py

2.3.6 simple_gui

This folder contains a simple example on creating a GUI (using pyqtgraph along with Twisted) to plot real-time data acquired from ThinkRF's RTSA device. It displays the spectral density data in the top plot, and the raw I &/or Q data (when available) in the lower plot.

- simple_gui

Usage:

```
python run_gui.py <device_ip>
```

2.4 Change Logs

2.4.1 PyRF 2.9.1

- Updated this PyRF Manual/Documentation
- Added more examples

2.4.2 PyRF 2.9.0

2018-10-12

- Added GNSS support for R5700 RTSA products including VRT GNSS context packet
- Added support for R5500 products
- Added flush and reset to capture setup
- Added calibrate_time_domain() function for a given time-domain data point
- Added calculate_occupied_bw() function for a given spectrum and occupied percentage
- Refactored sweep_device functions
- Restructured ThinkRF device properties and removed deprecated ones
- Improved IQ offset algorithm
- Enabled "100 kHz span" (HDR mode) for R5500 products
- Enabled trigger feature
- Changed Baseband (DD mode)'s MIN_TUNABLE to 32.25 MHz
- Changed R5500's minimum frequency to 10 kHz
- Changed WSA5000's minimum frequency to 100 kHz

- Changed minimum sample size for sweep to be 32
- Fixed zero-span setting
- Fixed bugs related to RBW setup
- Fixed bugs related sweep setup
- Fixed lock-up issue due to unexpected data packet received
- Fixed attenuation setting for R5500
- Fixed sample sizes being off by 32 samples
- Fixed capture_device bug related to number of data points
- Fixed bugs related to CSV file and settings

2.4.3 PyRF 2.8.0

2015-08-12

- Removed RTSA Instructions from the web page
- Fixed windows installation instructions
- Added capture spectrum function
- Added find peak function
- Added Measure noisefloor function
- Changed default span settings
- Added saturation level value for each device

2.4.4 PyRF 2.7.2

2014-12-16

- Added capture control widget
- Changed default save file names to represent date and time of capture
- Fixed baseband mode frequency axis issue
- Netifaces library is no longer a hard requirement
- Improved overall marker controls
- Added 'Enable mouse tune' option to frequency widget
- Default HDR gain is now 25

2.4.5 PyRF 2.7.1

2014-11-13

- Discovery widget now queries for new WSA's on the network every 10 seconds
- Fixed issue where switching from sweep to non-sweep wrongly changed center frequency
- Fixed issue where Minimum control not behaving as designed

- Fixed issue where trigger controls were not disabled for non-trigger modes
- Fixed frequency axis texts
- Y-axis in the persistence plot now corresponds with spectral plot's y-axis

2.4.6 PyRF 2.7.0

2014-11-04

- All control widgets are now dockable
- Enabled mouse control of spectral plot's y-axis
- Added lower RBW values in non-sweep modes

2.4.7 PyRF 2.6.2

2014-10-10

- HDR gain control in GUI now allows values up to +20 dB
- Sweep ZIF (100 MHz steps) now only shown in GUI when developer menu is enabled
- GUI PLL Reference control now works in Sweep mode
- Darkened trace color in GUI for attenuated edges and dc offset now matches trace color
- Alternate sweep step color in GUI now matches trace color
- DC offset region now limited to middle three bins in GUI (was expanding when decimation was applied)
- Correction to usable region in ZIF and SH modes with decimation applied
- Fixed HDR center offset value
- Added device information dialog to GUI

2.4.8 PyRF 2.6.1

2014-09-30

- Upload corrected version with changelog

2.4.9 PyRF 2.6.0

2014-09-30

- Added channel power measurement feature to GUI
- Added Export to CSV feature to GUI for saving streams of processed power spectrum data
- Added a power level cursor (adjustable horizontal line) to GUI
- Added reference level offset adjustment box to GUI
- Trigger region in GUI is now a rectangle to make it visibly different than channel power measurement controls

- Update mode drop-down in GUI to include information about each mode instead of showing internal mode names
- Use netifaces for address detection to fix discover issue on non-English windows machines

2.4.10 PyRF 2.5.0

2014-09-09

- Added Persistence plot
- Made markers drag-able in the plot
- Added version number to title bar
- Moved DSP options to developer menu, developer menu now hidden unless GUI run with -d option
- Rounded center to nearest tuning resolution step in GUI
- Fixed a number of GUI control and label issues
- Moved changelog into docs and updated

2.4.11 PyRF 2.4.1

2014-08-19

- Added missing requirement
- Fixed use with CONNECTOR IQ path

2.4.12 PyRF 2.4.0

2014-08-19

- Improved trigger controls
- Fixed modes available with some WSA versions

2.4.13 PyRF 2.3.0

2014-08-12

- Added full playback support (including sweep) in GUI
- Added hdr_gain control to WSA class
- Added average mode and clear button for traces
- Added handling for different REFLEVEL_ERROR on early firmware versions
- Disable triggers for unsupported WSA firmware versions
- Added free plot adjustment developer option
- Fixed a number of GUI interface issues

2.4.14 PyRF 2.2.0

2014-07-15

- Added waterfall display for GUI and example program
- Added automatic re-tuning when plot dragged of zoomed
- Added recording speca state in recorded VRT files, Start/Stop recording menu
- Added GUI non-sweep playback support and command line '-p' option
- Added marker controls: peak left, right, center to marker
- Redesigned frequency controls, device controls and trace controls
- Default to Sweep SH mode in GUI
- Added developer options menu for attenuated edges etc.
- Refactored shared GUI code and panels
- SweepDevice now returns numpy arrays of dBm values
- Fixed device discovery with multiple interfaces
- Replaced reflevel adjustment properties with REFLEVEL_ERROR value
- Renamed GUI launcher to rtsa-gui

2.4.15 PyRF 2.1.0

2014-06-20

- Refactored GUI code to separate out device control and state
- Added SPECA defaults to device properties
- Restored trigger controls in GUI
- Added DSP panel to control fft calculations in GUI
- Fixed a number of GUI plot issues

2.4.16 PyRF 2.0.3

2014-06-03

- Added simple QT GUI example with frequency, attenuation and rbw controls
- Added support for more hardware versions
- Fixed plotting issues in a number of modes in GUI

2.4.17 PyRF 2.0.2

2014-04-29

- Removed Sweep ZIF mode from GUI
- Fixed RFE input mode GUI issues

2.4.18 PyRF 2.0.1

2014-04-21

- Added Sweep SH mode support to SweepDevice
- Added IQ in, DD, SHN RFE modes to GUI
- Added IQ output path and PLL reference controls to GUI
- Added discovery widget to GUI for finding devices
- Fixed a number of issues

2.4.19 PyRF 2.0.0

2014-01-31

- Added multiple traces and trace controls to GUI
- Added constellation and IQ plots
- Added raw VRT capture-to-file support
- Updated SweepDevice sweep plan calculation
- Created separate GUI for single capture modes
- Updated device properties for WSA5000 RFE modes
- Show attenuated edges in gray, sweep steps in different colors in GUI
- Added decimation and frequency shift controls to single capture GUI
- Fixed many issues with WSA5000 different RFE mode support
- Removed trigger controls, waiting for hardware support
- Switched to using pyinstaller for better windows build support

2.4.20 PyRF 1.2.0

2013-10-01

- Added WSA5000 support
- Added WSA discovery example scripts
- Renamed WSA4000 class to WSA (supports WSA5000 as well)
- Separated device properties from WSA class

2.4.21 PyRF 1.1.0

2013-07-19

- Fixed some py2exe issues
- Show the GUI even when not connected

2.4.22 PyRF 1.0.0

2013-07-18

- Switched to pyqtgraph for spectrum plot
- Added trigger controls
- Added markers
- Added hotkeys for control
- Added bandwidth control
- Renamed GUI launcher speca-gui
- Created SweepDevice to generalize spectrum analyzer-type function
- Created CaptureDevice to collect single captures and related context

2.4.23 PyRF 0.4.0

2013-05-18

- pyrf.connectors.twisted_async.TwistedConnector now has a vrt_callback attribute for setting a function to call when VRT packets are received.

  This new callback takes a single parameter: a pyrf.vrt.DataPacket or pyrf.vrt.ContextPacket instance.

  The old method of emulating a synchronous read() interface from a pyrf.devices.thinkrf.WSA4000 instance is no longer supported, and will now raise a pyrf.connectors.twisted_async.TwistedConnectorError exception.
- New methods added to pyrf.devices.thinkrf.WSA4000: abort(), spp(), ppb(), stream_start(), stream_stop(), stream_status()
- Added support for stream ID context packets and provide a value for sweep ID context packet not converted to a hex string
- wsa4000gui updated to use vrt callback
- "wsa4000gui -v" enables verbose mode which currently shows SCPI commands sent and responses received on stdout
- Added examples/stream.py example for testing stream data rate
- Updated examples/twisted_show_i_q.py for new vrt_callback
- Removed pyrf.twisted_util module which implemented old synchronous read() interface
- Removed now unused pyrf.connectors.twisted_async.VRTTooMuchData exception
- Removed wsa4000gui-blocking script
- Fix for power spectrum calculation in pyrf.numpy_util

2.4.24 PyRF 0.3.0

2013-02-01

- API now allows asynchronous use with TwistedConnector

- GUI now uses asynchronous mode, but synchronous version may still be built as wsa4000gui-blocking
- GUI moved from examples to inside the package at pyrf.gui and built from the same setup.py
- add Twisted version of show_i_q.py example
- documentation: installation instructions, requirements, py2exe instructions, user manual and many other changes
- fix support for reading WSA4000 very low frequency range
- pyrf.util.read_data_and_reflevel() was renamed to read_data_and_context()
- pyrf.util.socketread() was moved to pyrf.connectors.blocking.socketread()
- added requirements.txt for building dependencies from source

2.4.25 PyRF 0.2.5

2013-01-26

- fix for compute_fft calculations

2.4.26 PyRF 0.2.4

2013-01-19

- fix for missing devices file in setup.py

2.4.27 PyRF 0.2.3

2013-01-19

- add planned features to docs

2.4.28 PyRF 0.2.2

2013-01-17

- rename package from python-thinkrf to PyRF

2.4.29 python-thinkrf 0.2.1

2012-12-21

- update for WSA4000 firmware 2.5.3 decimation change

2.4.30 python-thinkrf 0.2.0

2012-12-09

- GUI: add BPF toggle, Antenna switching, –reset option, “Open Device” dialog, IF Gain control, Span control, RBW control, update freq on finished editing
- create basic documentation and reference and improve docstrings

- bug fixes for GUI, py2exe setup.py
- GUI perfomance improvements

2.4.31 python-thinkrf 0.1.0

2012-12-01

- initial release

CHAPTER 3

Hardware Support

This library currently supports development for the following ThinkRF Real-Time Spectrum Analyzer (RTSA) platforms:

- R5500
- R5700
- WSA5000 (EOL)

CHAPTER 4

Links

- Official PyRF github page
- PyRF Documentation
- ThinkRF RTSA Documentation and Resources

CHAPTER 5

Indices and Tables

- genindex
- search

Python Module Index

p

Index

A

C

D

E

F

H

I

L

M

N

P

R

S

T

V

W