API for RF receivers including ThinkRF WSA platform

Release

ThinkRF Corporation

October 10, 2014

Contents

Contents:

CHAPTER 1

Manual

1.1 Installation

1.1.1 Windows Dependencies

1. Download https://s3.amazonaws.com/ThinkRF/Support-Resources/pyrf-dependencies.zip
2. Extract the contents of the zipped file
3. Install Python 2.7.6 (python-2.7.6.msi)
4. Add the following to the windows PATH ';C:Python27;C:Python27Scripts'
5. Install Numpy (numpy-1.8.1-win32-superpack-python2.7)
6. Install Scipy (scipy-0.14.0-win32-superpack-python2.7)
7. Install Pyside (PySide-1.2.0.win32-py2.7)
8. Install Pyqtgraph (pyqtgraph-0.9.8.win32)
9. Install zope.interface (zope.interface-4.1.1.win32-py2.7)
10. Install twisted (Twisted-14.0.0.win32-py2.7)
11. Install pywin32 (pywin32-219.win32-py2.7)
12. Install netifaces (netifaces-0.10.4.win32-py2.7)
13. Using a command line, go to the qtreactor-qtreactor-pyrf-1.0 folder, and type 'setup.py install'
14. Using a command line, go to the setuptools-5.7 folder and type 'setup.py install'

Continue from *PyRF Installation* below.

1.1.2 Debian/Ubuntu Dependencies

Use packaged requirements:

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

Or install GUI requirements 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
```

Continue from *PyRF Installation* below.

1.1.3 PyRF Installation

Download the development version:

```
git clone git://github.com/pyrf/pyrf.git
cd pyrf
python setup.py install
```

Or download a stable release, then from the source directory:

```
python setup.py install
```

1.2 API for WSA RF Receiver

`pyrf.devices.thinkrf.WSA` is the class that provides access to WSA4000 and WSA5000 devices. Its methods closely match the SCPI Command Set described in the Programmers Reference available in ThinkRF Resources.

There are simple examples that use this API under the "examples" directory included with the source code.

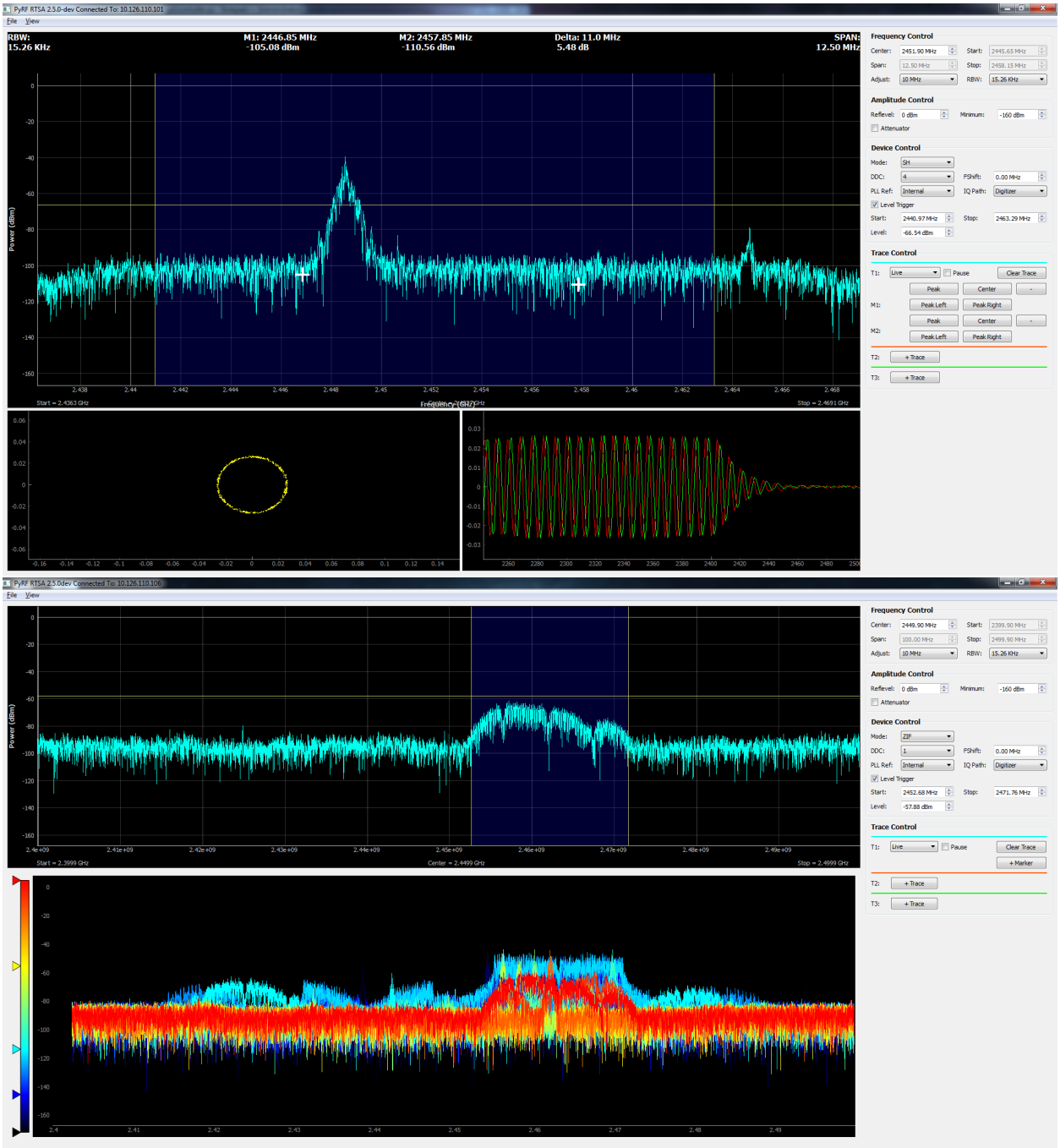
This API may be used in a blocking mode (the default) or in an asynchronous mode with using the **'Twisted'_** python library.

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 is created you must pass a `pyrf.connectors.twisted_async.TwistedConnector` instance as the connector parameter, as in *show_i_q.py / twisted_show_i_q.py*

1.3 PyRF RTSA



`rtsa-gui` is a cross-platform GUI application built with the Qt toolkit and PySideProject bindings for Python.

The GUI may be launched with the command:

```
rtsa-gui <hostname> [--reset]
```

If *hostname* is not specified a dialog will appear asking you to enter one. If `--reset` is used the WSA will be reset to defaults before the GUI appears.

CHAPTER 2

Reference

2.1 pyrf.devices

2.1.1 .thinkrf

2.2 pyrf.sweep_device

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

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

Parameters

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

capture_power_spectrum(*fstart*, *fstop*, *rbw*, *device_settings=None*, *mode='ZIF'*, *continuous=False*, *min_points=32*)

Initiate a capture of power spectral density by setting up a sweep list and starting a single sweep.

Parameters

- **fstart** (*float*) – starting frequency in Hz
- **fstop** (*float*) – ending frequency in Hz
- **rbw** (*float*) – requested RBW in Hz (output RBW may be smaller than requested)
- **device_settings** – antenna, gain and other device settings
- **mode** (*string*) – sweep mode, 'ZIF left band', 'ZIF' or 'SH'
- **continuous** (*bool*) – async continue after first sweep
- **min_points** (*int*) – smallest number of points per capture from real_device

exception pyrf.sweep_device.**SweepDeviceError**

class pyrf.sweep_device.**SweepStep**

Data structure used by SweepDevice for planning sweeps

Parameters

- **fcenter** – starting center frequency in Hz
- **fstep** – frequency increment each step in Hz
- **fshift** – frequency shift in Hz
- **decimation** – decimation value
- **points** – samples to capture
- **bins_skip** – number of FFT bins to skip from left
- **bins_run** – number of usable FFT bins each step
- **bins_pass** – number of bins from first step to discard from left
- **bins_keep** – total number of bins to keep from all steps

steps

to_sweep_entry(*device*, *rfe_mode*, ***kwargs*)

Create a SweepEntry for device matching this SweepStep,

extra parameters (gain, antenna etc.) may be provided as keyword parameters

pyrf.sweep_device.**plan_sweep**(*device*, *fstart*, *fstop*, *rbw*, *mode*, *min_points=32*)

Parameters

- **device** – a device class or instance such as `pyrf.devices.thinkrf.WSA`
- **fstart** (*float*) – starting frequency in Hz
- **fstop** (*float*) – ending frequency in Hz
- **rbw** (*float*) – requested RBW in Hz (output RBW may be smaller than requested)
- **mode** (*string*) – sweep mode, 'ZIF left band', 'ZIF' or 'SH'
- **min_points** (*int*) – smallest number of points per capture

The following device properties are used in planning the sweep:

device.properties.FULL_BW full width of the filter in Hz

device.properties.USABLE_BW usable portion before filter drop-off at edges in Hz

device.properties.MIN_TUNABLE the lowest valid center frequency for arbitrary tuning in Hz, 0(DC) is always assumed to be available for direct digitization

device.properties.MAX_TUNABLE the highest valid center frequency for arbitrart tuning in Hz

device.properties.DC_OFFSET_BW the range of frequencies around center that may be affected by a DC offset and should not be used

device.properties.TUNING_RESOLUTION the smallest tuning increment for fcenter and fstep

Returns (actual fstart, actual fstop, list of SweepStep instances)

The caller would then use each of these tuples to do the following:

1. The first 5 values are used for a single capture or single sweep
2. An FFT is run on the points returned to produce bins in the linear domain
3. bins[bins_skip:bins_skip + bins_run] are selected
4. take logarithm of output bins and appended to the result

5.for sweeps repeat from 2 until the sweep is complete

6.bins_pass is the number of selected bins to skip from the first capture only

7.bins_keep is the total number of selected bins to keep; for single captures bins_run == bins_keep

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** – device that will 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 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=128*)

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

Parameters

- **rfe_mode** – radio front end mode, e.g. 'ZIF', 'SH', ...
- **freq** – center frequency
- **rbw** (*float*) – requested RBW in Hz (output RBW may be smaller than requested)
- **device_settings** – attenuator, decimation frequency shift and other device settings
- **min_points** (*int*) – smallest number of points per capture from real_device

configure_device(*device_settings*)

Configure the device settings on the next capture

Parameters **device_settings** – attenuator, decimation frequency shift and other device settings

read_data(*packet*)

exception pyrf.capture_device.**CaptureDeviceError**

2.4 pyrf.connectors

2.4.1 .blocking

class pyrf.connectors.blocking.**PlainSocketConnector**

This connector makes SCPI/VRT socket connections using plain sockets.

connect(*host*)

disconnect()

eof()

has_data()

raw_read(*num*)

scpiget(*cmd*)

scpiset(*cmd*)

sync_async(*gen*)
Handler for the @sync_async decorator. We convert the generator to a single return value for simple synchronous use.

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

2.4.2 .twisted_async

class pyrf.connectors.twisted_async.**SCPIClient**

connectionMade()

dataReceived(*data*)

scpiget(*cmd*)

scpiset(*cmd*)

class pyrf.connectors.twisted_async.**SCPIClientFactory**

buildProtocol(*addr*)

clientConnectionFailed(*connector*, *reason*)

clientConnectionLost(*connector*, *reason*)

startedConnecting(*connector*)

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

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.

connect(*host*, *output_file=None*)

disconnect()

eof()

inject_recording_state(*state*)

raw_read(*num_bytes*)

scpiget(*cmd*)

scpiset(*cmd*)

set_recording_output(*output_file=None*)

sync_async(*gen*)

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

connectionLost(*reason*)

dataReceived(*data*)

eof = False

inject_recording_state(*state*)

makeConnection(*transport*)

set_recording_output(*output_file=None*)

class pyrf.connectors.twisted_async.**VRTClientFactory**(*receive_callback*)

buildProtocol(*addr*)

clientConnectionFailed(*connector*, *reason*)

clientConnectionLost(*connector*, *reason*)

startedConnecting(*connector*)

2.5 pyrf.config

class pyrf.config.**SweepEntry**(*fstart=2400000000*, *fstop=2400000000*, *fstep=100000000*, *fshift=0*, *decimation=0*, *antenna=1*, *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=True*, *rfe_mode='ZIF'*)

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

Parameters

- **fstart** – starting frequency in Hz
- **fstop** – ending frequency in Hz
- **fstep** – frequency step in Hz
- **fshift** – the frequency shift in Hz
- **decimation** – the decimation value (0 or 4 - 1023)
- **antenna** – the antenna (1 or 2)
- **gain** – the RF gain value ('high', 'medium', 'low' or 'vlow')
- **ifgain** – the IF gain in dB (-10 - 34)
- **hdr_gain** – the HDR gain in dB (-10 - 30)
- **spp** – samples per packet
- **ppb** – packets per block
- **dwell_s** – dwell time seconds
- **dwell_us** – dwell time microseconds
- **trigtype** – trigger type ('none', 'pulse' or 'level')
- **level_fstart** – level trigger starting frequency in Hz

- **level_fstop** – level trigger ending frequency in Hz
- **level_amplitude** – level trigger minimum in dBm
- **attenuator** – enable/disable attenuator

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

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

Parameters

- **trigtype** – "LEVEL" or "NONE" to disable
- **fstart** – starting frequency in Hz
- **fstop** – ending frequency in Hz
- **amplitude** – minumum level for trigger in dBm

exception pyrf.config.**TriggerSettingsError**

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. :param power_spectrum: array containing power spectrum to be used for

the channel power calculation

pyrf.numpy_util.**compute_fft**(*dut*, *data_pkt*, *context*, *correct_phase=True*, *hide_differential_dc_offset=True*, *convert_to_dbm=True*, *apply_window=True*, *apply_spec_inv=True*, *apply_reference=True*, *ref=None*)

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 containing context values
- **correct_phase** – apply phase correction for captures with IQ data
- **hide_differential_dc_offset** – mask the differential DC offset present in captures with IQ data
- **convert_to_dbm** – convert the output values to dBm

Returns numpy array of dBm values as floats

2.7 pyrf.util

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

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

Returns (data_pkt, context_values)

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

pyrf.util.**collect_data_and_context**(*dut*)
Wait for and return data packet and collect preceeding context packets.

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()

fields
a dict containing field names and values from the packet

is_context_packet(*ptype=None*)

Parameters **ptype** – "Receiver", "Digitizer" or None for any

packet type

Returns True if this packet matches the type passed

is_data_packet()

Returns False

class pyrf.vrt.**DataArray**(*binary_data*, *bytes_per_sample*)
Data Packet values as a lazy array read from *binary_data*.

Parameters **bytes_per_sample** – 1 for PSD8 data, 2 for I14 data or 4 for I24 data

numpy_array()
return a numpy array for this data

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 similar to:

```
array([[ -44,    8],
       [ -40,   60],
       [ -12,   92],
       ...,
       [-132,   -8],
       [-124,   56],
       [ -44,   80]], dtype=int16)
```

exception pyrf.vrt.**InvalidDataReceived**

pyrf.vrt.**generate_speca_packet**(*data*, *count=0*)

Parameters

- **data** – a python dict that can be serialized as JSON
- **count** – int count for the header of this packet

Returns (vrt packet bytes, next count int)

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.

CHAPTER 3

Examples

These examples may be found in the "examples" directory included with the PyRF source code.

3.1 discovery.py / twisted_discovery.py

- discovery.py
- twisted_discovery.py

These examples detect WSA devices on the same network

Example output:

```
WSA4000 00:50:c2:ea:29:14 None at 10.126.110.111
WSA4000 00:50:c2:ea:29:26 None at 10.126.110.113
```

3.2 show_i_q.py / twisted_show_i_q.py

These examples connect to a device 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
```

3.3 matplotlib_plot_sweep.py

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

- matplotlib_plot_sweep.py

3.4 pyqtgraph_plot_block.py

This example connects to a device specified on the command line, tunes it to a center frequency of 2.450 MHz then continually captures and displays an FFT in a GUI window.

- pyqtgraph_plot_block.py

CHAPTER 4

Changelog

4.1 PyRF 2.6.1

2014-09-30

- Upload corrected version with changelog

4.2 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

4.3 PyRF 2.5.0

2014-09-09

- Added Persistence plot
- Made markers draggable 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

4.4 PyRF 2.4.1

2014-08-19

- Added missing reqirement
- Fixed use with CONNECTOR IQ path

4.5 PyRF 2.4.0

2014-08-19

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

4.6 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

4.7 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

4.8 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

4.9 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

4.10 PyRF 2.0.2

2014-04-29

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

4.11 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

4.12 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

4.13 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

4.14 PyRF 1.1.0

2013-07-19

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

4.15 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

4.16 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

4.17 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

4.18 PyRF 0.2.5

2013-01-26

- fix for compute_fft calculations

4.19 PyRF 0.2.4

2013-01-19

- fix for missing devices file in setup.py

4.20 PyRF 0.2.3

2013-01-19

- add planned features to docs

4.21 PyRF 0.2.2

2013-01-17

- rename package from python-thinkrf to PyRF

4.22 python-thinkrf 0.2.1

2012-12-21

- update for WSA4000 firmware 2.5.3 decimation change

4.23 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

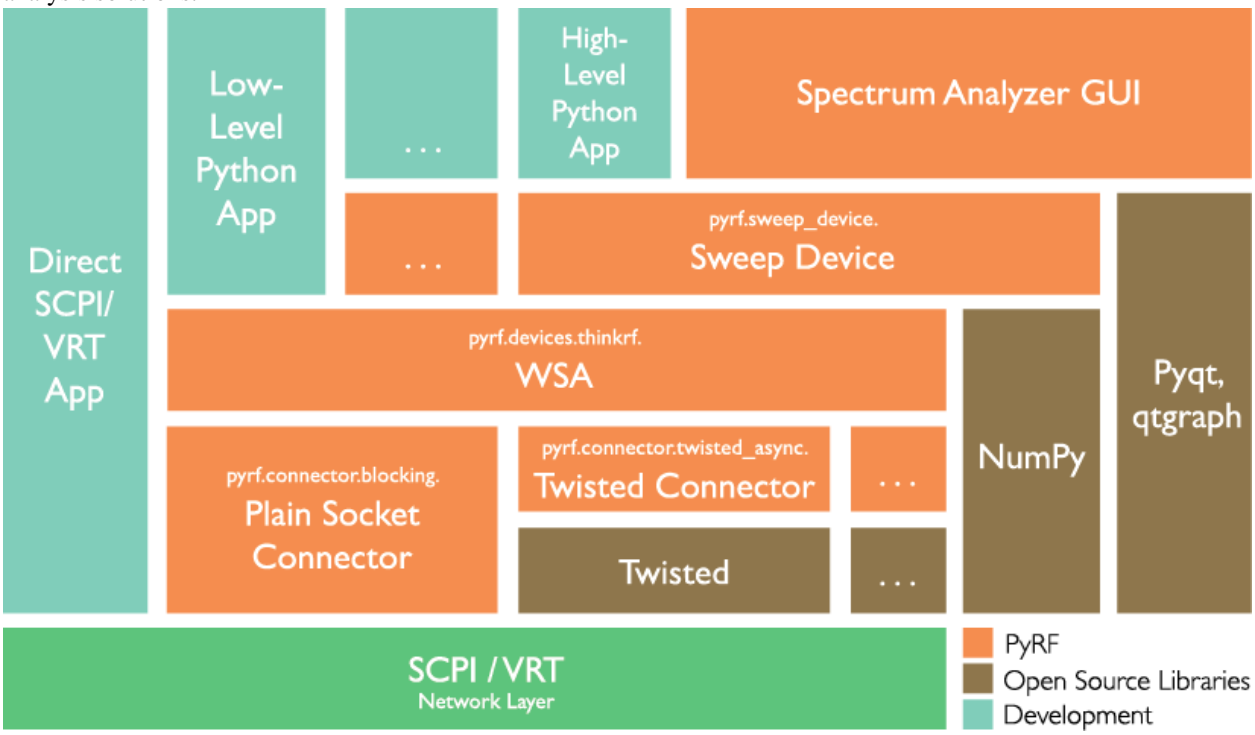
4.24 python-thinkrf 0.1.0

2012-12-01

- initial release

PyRF is an openly available, comprehensive development environment for wireless signal analysis. It enables rapid development of powerful applications that leverage the new generation of measurement-grade software-defined radio technology.

PyRF is built on the Python Programming Language and includes feature-rich libraries, example applications and source code, all specific to the requirements of signal analysis. PyRF is openly available, allowing commercialization of solutions through BSD open licensing and offering device independence via standard hardware APIs. PyRF handles the low-level details of real-time acquisition, signal processing and visualization, allowing you to concentrate on your analysis solutions.



CHAPTER 5

Hardware Support

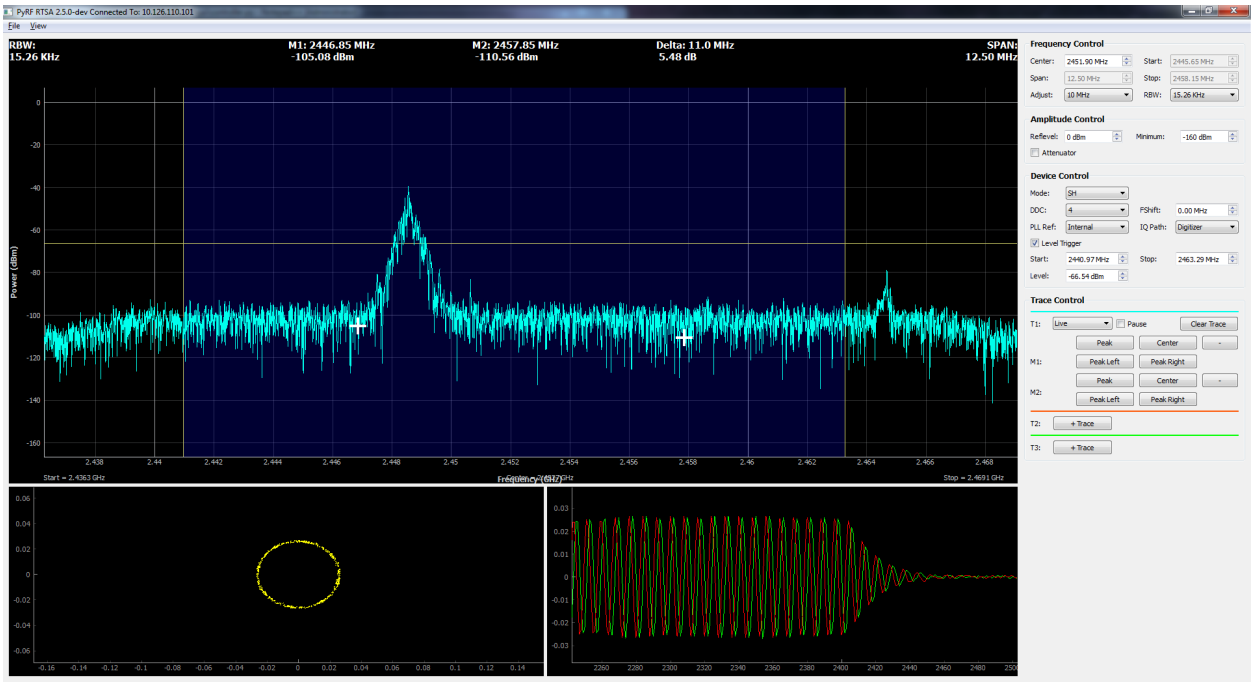
This library currently supports development for the ThinkRF WSA4000 and WSA5000 platforms.

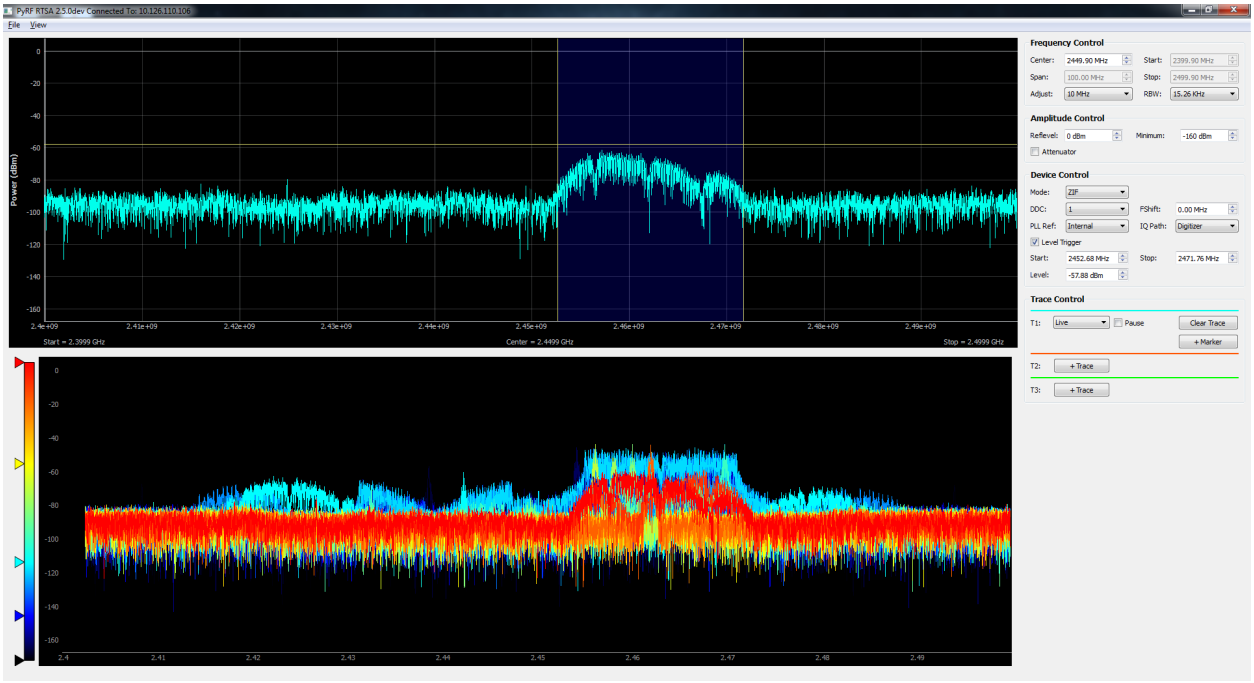
CHAPTER 6

Links

- Official github page
- Documentation for this API
- WSA4000/WSA5000 Documentation

PyRF RTSA


PyRF RTSA 2.5.0-dev Connected To: 10.126.110.101
File View
RBW: 15.26 KHz
M1: 2446.85 MHz -105.08 dBm
M2: 2457.85 MHz -110.56 dBm
Delta: 11.0 MHz 5.48 dB
SPAN: 12.50 MHz
Power (dBm)
Frequency (GHz)
Start = 2.4363 GHz
Stop = 2.4691 GHz
Frequency Control
Center: 2451.90 MHz
Start: 2445.65 MHz
Span: 12.50 MHz
Stop: 2458.15 MHz
Adjust: 10 MHz
RBW: 15.26 KHz
Amplitude Control
Reflevel: 0 dBm
Minimum: -160 dBm
Attenuator
Device Control
Mode: SH
DDC: 4
FShift: 0.00 MHz
PLL Ref: Internal
IQ Path: Digitizer
Level Trigger
Start: 2440.97 MHz
Stop: 2463.29 MHz
Level: -66.54 dBm
Trace Control
T1: Live
Pause
Clear Trace
Peak
Center
Peak Left
Peak Right
M1:
M2:
T2:
+ Trace
T3:
+ Trace


PyRF RTSA 2.5.0dev Connected To: 10.126.110.106
File View
Power (dBm)
Start = 2.3999 GHz
Center = 2.4499 GHz
Stop = 2.4999 GHz
Frequency Control
Center: 2449.90 MHz
Start: 2399.90 MHz
Span: 100.00 MHz
Stop: 2499.90 MHz
Adjust: 10 MHz
RBW: 15.26 KHz
Amplitude Control
Reflevel: 0 dBm
Minimum: -160 dBm
Attenuator
Device Control
Mode: ZIF
DDC: 1
FShift: 0.00 MHz
PLL Ref: Internal
IQ Path: Digitizer
Level Trigger
Start: 2452.68 MHz
Stop: 2471.76 MHz
Level: -57.88 dBm
Trace Control
T1: Live
Pause
Clear Trace
+ Marker
T2:
+ Trace
T3:
+ Trace

CHAPTER 8

Indices and tables

- *genindex*
- *search*

p

B

C

D

E

F

G

H

I