Reference Guide

74-0039-190702

Using the ThinkRF Real-Time Spectrum Analyzer with MATLAB

thinkRF
monitor. detect. analyze

Contents

Introduction

The ThinkRF Real-Time Spectrum Analyzer (RTSA) has the performance of traditional high-end lab spectrum analyzers at a fraction of the cost, size, weight and power consumption and is designed for distributed deployment.

MATLAB (MATrix LABoratory) is a multi-paradigm numerical computing environment and proprietary programming language developed by MathWorks. MATLAB allows matrix manipulations, plotting of functions and data, implementation of algorithms, creation of user interfaces, and interfacing with programs written in other languages, including C, C++, C#, Java, Fortran and Python.

To make use of MATLAB, ThinkRF provides the RTSA MATLAB API (Application Programming Interface). This API Reference Guide will help you to easily and quickly integrate your ThinkRF RTSA into your existing or new MATLAB application development. It will walk you through the ThinkRF provided RTSA MATLAB API scripts and examples.

This document also assumes you have some network knowledge and have access to a RTSA product. Otherwise, you can connect via the Internet to a ThinkRF's evaluation RTSA unit at www.thinkrf.com/demo.

Software and System Requirements

To use the RTSA MATLAB API, the following software and system is required:

- Windows 7/higher with 32-bit/64-bit operating system
- MATLAB R2014b or higher
- To run the GUI examples, MATLAB R2016 and newer is required
- rtsaInterface.dll, a C++ DLL provided by ThinkRF (included in the MATLAB API in the `..\include` folder).

The latest RTSA firmware and software may be downloaded from http://www.thinkrf.com/firmware-updates/.

Installation Procedure

To Install the Software simply extract the package to a location of your choosing (e.g. C:\...\RTSA\).

MATLAB will be able to call the driver functions, as soon as the API folder is added to the MATLAB path. To make sure the path is present, run the following code in MATLAB:

```
>> Userpath
```

In this case, it returned the directory 'C:\Users\user\Documents\MATLAB'. Navigate to that directory.

If there is a script called "startup.m", add the following line, replacing <driver path> with the path you created during the extract step. If there is no such script, create it and add the same line.

```
>> addpath(genpath(<driver path>));
```

API Overview

The API is built as a class called rtsaInterface() with properties (storing all relevant data) and methods to perform setup and data acquisition tasks. Here's and overview of the class data and methods:

- rtsaInterface
  - **Properties**
    - hndl
    - connected
    - ***Device Properties***
      - manufacturer
      - device
      - model
      - serial
      - firmware
      - minFreq
      - maxFreq
    - *Error handling*
      - result
      - message
      - stack
    - *struct:* traceData
      - rfeMode
      - centerFreq
      - pllRefSource
      - decimation
      - attenuation
    - *struct:* dataContext
      - streamID
      - spectralInversion
      - spp
      - ppb
      - timestampSec
      - timestampPSec
      - refLvl
      - bandwidth
      - centerFreq
    - *struct:* sweepDeviceInput
      - startFreq
      - stopFreq
      - rbw
      - rfeMode
      - attenuation
    - *struct:* sweepSetup
      - startFreqActual
      - stopFreqActual
      - sweepBuffer
    - *struct:* triggerData
      - triggerType
      - startFreq
      - stopFreq
      - amplitude
  - **Methods**
    - ***Construct/Destruct***
      - rtsaInterface()
      - delete()
    - ***Connect/Disconnect***
      - connect()
      - disconnect()
    - ***SCPI Command***
      - querySCPI()
      - getIDN()
      - getTemperatures()
      - getSCPIError()
      - setSCPI()
      - sytemReset()
      - systemFlush()
    - ***Trace Config and Data Capture***
      - setTraceAttenuation()
      - getTraceAttenuation()
      - traceConfigure()
      - setTraceLevelTrigger()
      - traceCapture()
      - captureTraceSpectrum()
    - ***Sweep-Device***
      - sweepDeviceSetup()
      - captureSweepSpectrum()
      - sweepAndPeakFind()
    - ***Other Data Capture***
      - readIFPacket
    - ***Data Processing***
      - findPeak()
      - computeChannelPower()
      - computeOccupiedBandwidth()
    - ***Error Related***
      - getErrorMessage()
      - displayErrors()

The properties are mostly intended as a storage space in the API, but they can be accessed to get more information about the session and the acquired data. In the following chapters, we will take a closer look at the methods. See also `rtsaInterface.m` for more complete description.

Quick start

The easiest way to get started with the API is to take a look at the examples provided in `..\examples` folder:

- script examples (`scripts\*.m`) which contain short scripts that perform a single type of data acquisition. The scripts could be run in any MATLAB version.
- the graphical examples (`GUI\*.mlapp`) which offer a user interface for the configuration of the device and data display. These examples are built using MATLAB AppDesigner RR2016b so that they could be opened in MATLAB R2016b version or higher.

In general, steps for interfacing to an RTSA and performing data acquisition are as follows:

1. Instantiate a rtsaInterface object, such as:

```
Interface = rtsaInterface();
```

2. Connect to the device:

```
Interface.connect(ipAddress);
```

3. Optional setup before data acquisition:

```
Interface.systemReset()
Interface.systemFlush()
Interface.setSCPI()
```

4. **For trace mode acquisition:**	4. **For sweep acquisition:**

Configure the trace setup:
`Interface.traceConfigure()`
Set frequency level trigger if needed:
`Interface.setLevelTrigger()`
Acquire time-domain data only:
`Interface.traceCapture();`
Acquire power spectral density (PSD) data:
`Interface.captureTraceSpectrum();`
	Configure the sweep:
	`Interface.sweepDeviceSetup()`
	To capture only spectral data:
	`Interface.captureSweepSpectrum()`
	To capture spectral data and find peak power:
	`Interface.sweepAndPeakFind()`

5. Other data processing could be applied to the spectral data:

```
Interface.findPeak()
Interface.computeChannelPower()
Interface.computeOccupiedBandwidth()
```

6. Disconnect from the device:

```
Interface.disconnect();
```

7. Display if errors have occurred during the execution:

```
Interface.displayErrors();
```

8. Deconstruct the rtsaInterface object:

```
Interface.delete();
```

Error Handling

If errors occur during the communication with the device, most methods will report back the error number as well as the error string through its output. These error outputs serve the purpose of informing the user during the execution and as conditions for the program control. To get the error, either read the `result`, call the `displayErrors()` method to display them in the command window.

There are two methods that do not skip execution on error: `disconnect()` and `delete()`, as they will try to deinitialize the connection even on error.

rtsaInterface methods

Construct/Destruct

rtsaInterface()

Constructs the rtsaInterface and loads the DLL into memory. It takes up to a minute to load the driver DLL into memory. This has to be done only during construction, so this method should be called at the beginning of your script.

delete()

Releases the object from memory and unloads the rtsaInterface driver DLL.

Connect/Disconnect

connect()

Connects to the RTSA device at the given IP Address. At success, the handle remains open for future access by other functions and is stored internally in the rtsaInterface class.

disconnect() must be called at the end of the program.

disconnect()

Close an already established connection to the RTSA.

SCPI Command

querySCPI()

Sends a SCPI query to the device and receives its response. See the product's Programmer's Guide for the list of SCPI query commands.

getIDN()

Queries the device identification over SCPI.

getTemperatures()

Queries the device's temperatures over SCPI (degree C). For R5xx0, the temperatures are <RF>,<Mixer>,<Digital>.

getSCPIError()

Queries the device's SCPI Error log for any error. Recommend to use (usually initially) after each SCPI command to make sure no error resulted from that command (ex. typo, invalid values, etc.).

setSCPI()

Sends a SCPI command to configure the device. Use getSCPIError() if needed, to determine if the command has been set successfully.

systemReset()

Does a system reset over SCPI.

systemFlush()

Does system flush command to clean up the device's internal memory.

Trace Configuration and Data Capture

The methods in this section are used for trace capture mode (ie. block or stream capture, not sweep-device capture). Several examples are provided, to name a few: captureTraceSpectrumExample.m, readIQ.m, or CaptureTraceSpectrumExample.mlapp GUI.

setTraceAttenuation()

Sets the attenuation for a trace capture.

For R5xx0, supported values are : 0, 10, 20, 30 dB. For non-RTSA legacy products, check the product's Programmer's Guide.

getTraceAttenuation()

Gets the attenuation value of a trace capture.

traceConfigure()

Resets the device's settings and configures it with the provided settings (for a quick setup).

Notes:

- This configuration is for trace capture mode only, not with sweep-device.

- The settings here are not comprehensive, see the product's Programmer's Guide for other settings (ex. gain, frequency shift) and use setSCPI() to apply them.

setTraceLevelTrigger()

Configures a frequency-domain level trigger for trace capture (not to be used with sweep-device capture) or turns it off. Once turned on, the settings remain until it is turned off or a systemReset() is applied.

traceCapture()

Reads a block of time-domain I/Q Data from the device along with the associated VRT context information. A block is defined by ppb * spp (or Packets per Block * Sampes per Packet) and is

contiguous and continuous. The data returned is not normalized. See readIQ.m script example for how to normalize the data.

Note that this is for trace capture (for sweep capture, see Sweep-Device Section).

captureTraceSpectrum()

Captures a block of samples (trace mode), performs an FFT and compute the 'usable' spectrum data (power spectral density or PSD). A block is defined by `ppb` * `spp` (or Packets per Block * Samples per Packet) and is contiguous and continuous. See captureTraceSpectrumExample.m script as a usage example.

Note that this is for trace capture (for sweep capture, see Sweep-Device Section).

Sweep-Device

These methods are high-level methods that abstract away the complex setup needed for performing an RTSA sweep. See sweepDeviceExample.m script or `SweepDeviceExample.mlapp` for usage examples.

sweepDeviceSetup()

Sets up the sweep-device for capture and returns the sweepBuffer (number of data points) for the given sweep-device configuration.

captureSweepSpectrum()

Performs the sweep-device setup created in sweepDeviceSetup(), captures the sweep data and computes the PSD data.

sweepAndPeakFind()

Performs the sweep-device setup created in sweepDeviceSetup(), captures and finds the peak and its frequency of that sweep range.

Other Data Capture Related Methods

This section contain method(s) that could be applied to both trace and manual sweep (which users set up through manual SCPI commands, instead of using sweep-device).

readIFPacket()

Reads one IF (I/Q) data packet along with its associated VRT context information. This function could be used for trace or manual sweep (manually setup using SCPI commands) mode. It is used in traceCapture().

Data Processing

These methods provide some useful data processing capabilities that could facilitate and ease user's application development.

findPeak()

Returns the peak amplitude of the given spectral data array and its frequency. It is used directly sweepAndPeakFind().

See captureTraceSpectrumExample.m, CaptureTraceSpectrumExample.mlapp, sweepDeviceExample.m, or SweepDeviceExample.mlapp for example.

computeChannelPower()

Calculates the channel power of a given range of spectral (PSD) data.

See captureTraceSpectrumExample.m, CaptureTraceSpectrumExample.mlapp, sweepDeviceExample.m, or SweepDeviceExample.mlapp for example.

computeOccupiedBandwidth()

Calculates the occupied power bandwidth for the specified occupied percentage of a given spectral (PSD) data.

See captureTraceSpectrumExample.m, CaptureTraceSpectrumExample.mlapp, sweepDeviceExample.m, or SweepDeviceExample.mlapp for example.

Error Related Methods

getErrorMessage()

Retrieves the error message for the given API error code.

displayErrors()

Displays the error messages in the MATLAB console.

Examples

GUI Examples

The GUI examples are programmed in the MATLAB APP designer and are available for MATLAB version **RR2016b and later only**. The examples are located in `..\examples\GUI` folder. The examples execute the basic tasks of the RTSA and are provided with code to demonstrate how to execute them.

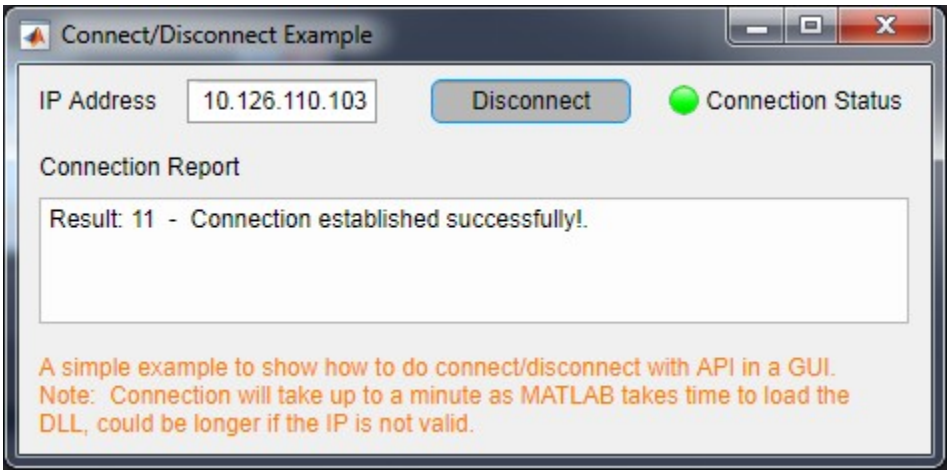
To view the code of modify the mlapp file for your own application, right-mouse click on the *.mlapp file and select **Open**.

Since GUI examples are developed in MATLAB R2016b version (so that later version could run it), to use them in a newer MATLAB version, we recommend to “Open” the example in that newer version first and then save the exampes in that version to avoid any potential version related errors during run time.

ConnectDisconnect.mlapp

This example shows how to do connect/disconnect to the RTSA device. When the app starts, enter the IP-Address of the RTSA in the field and click connect. Once the connection is made, the Connection Status will go green, as well as a message in Connection report. Click on Disconnect to properly stop accessing that device.

The connect process is identical throughout all the GUI examples.

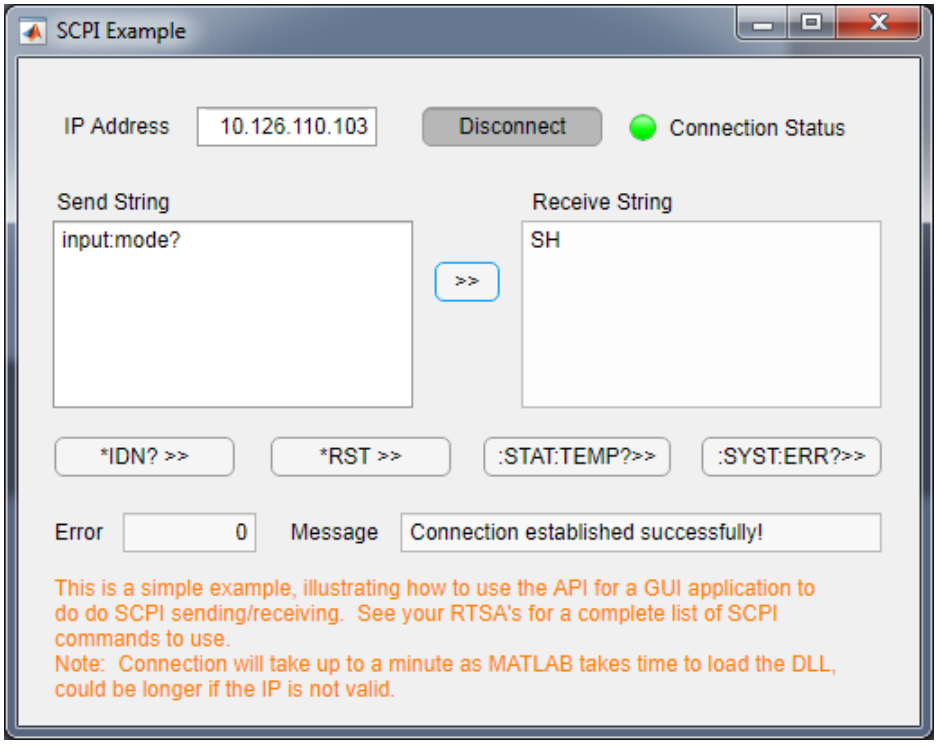


SCPIExample.mlapp

This example GUI connects to a device and sends SCPI query or set commands. After the connection has been established, SCPI commands can be sent by entering them in the "Send String" field and clicking the **[>>]** button.

Refer to the RTSA Programmer's Guide for a complete list of available commands and queries. Some common commands are available through the buttons at the bottom. The "Error" and "Message" fields display the error status of the SCPI sent when an error occurred.

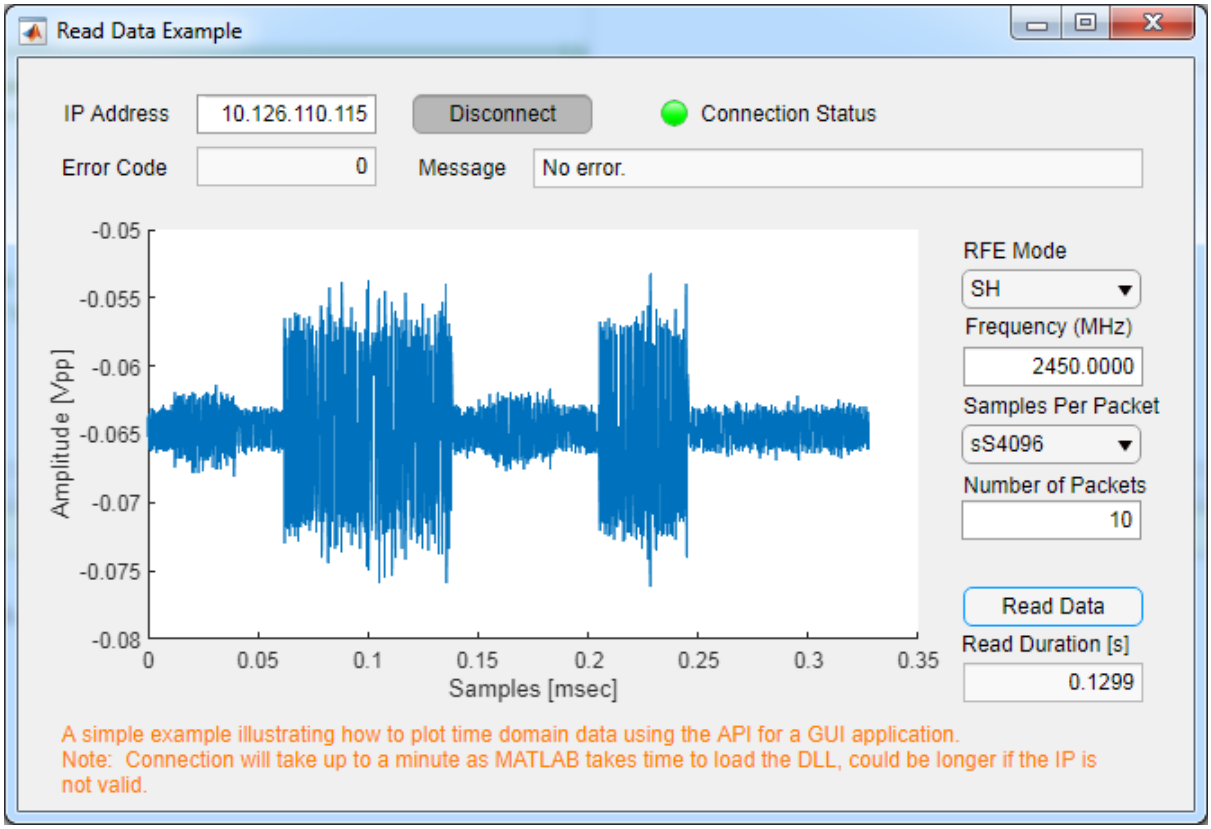
This GUI could be used in conjunction with other examples if other device settings are not available from those GUI examples.



ReadDataExample.mlapp

This example connects to a device and acquires a block of IQ Data. After the connection has been established, all drop down menus will be filled with values. Once you have made your setup, click the **Read Data** button to start the capture. The program will acquire a block of data, plot it and then pause.

The "Read Duration" indicates how long it takes to acquire that block of data. It is only for information, doesn't carry any significant meaning as it depends on your PC system and network speed.



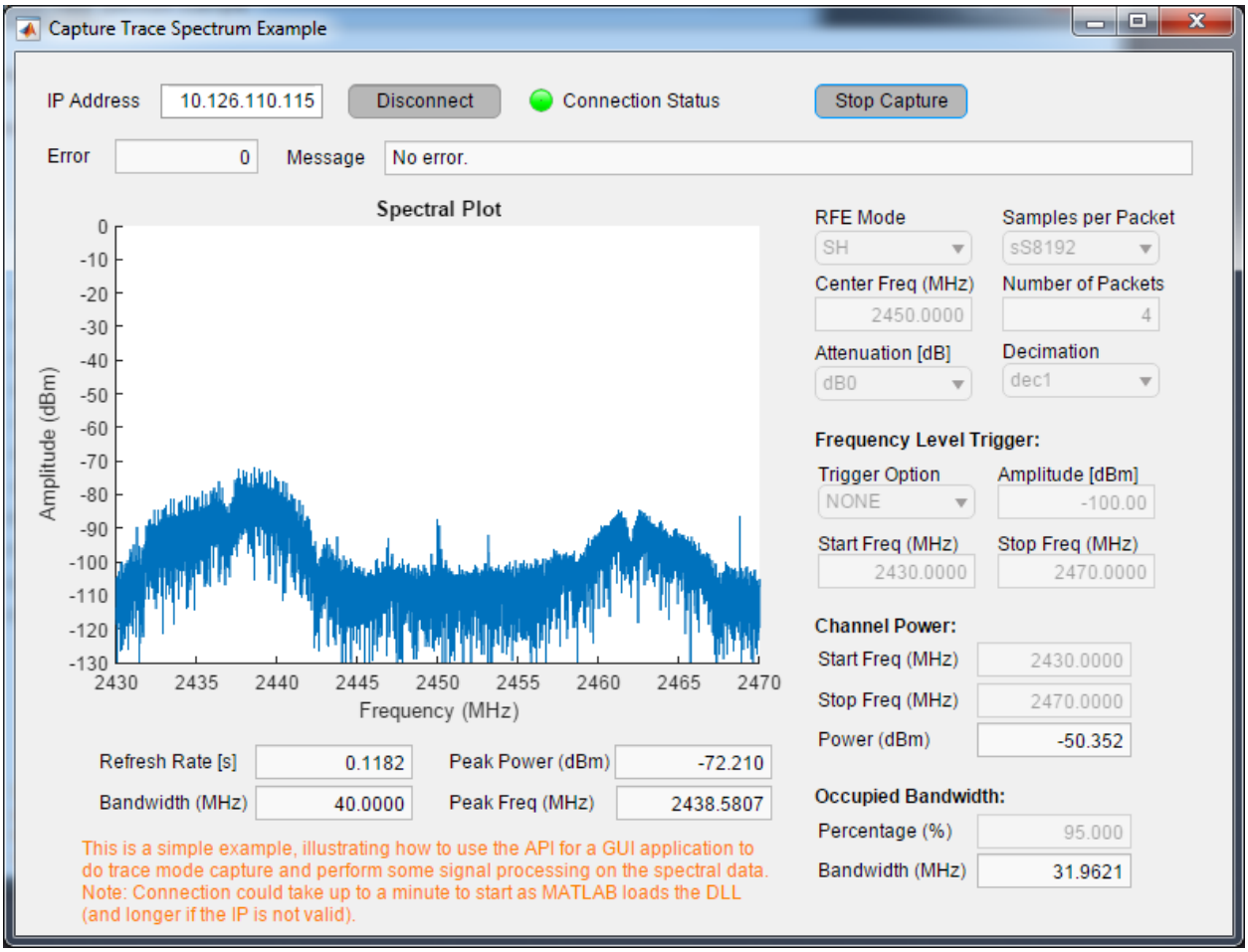
CaptureTraceSpectrumExample.mlapp

This example connects to a device, captures a block of data, performs captureTraceSpectrum() as well as other signal processing on them. After connection is established, all the Drop Down fields will be filled with values for selection. The input fields of "Frequency Level Trigger", "Channel Power", and Occupied Bandwidth" could be left as default if not used.

Once you have completed your setup, click **[Start Capture]** button to run the capture. The program will continuously update the Graph until you click **[Stop Capture]** button. No live settings changed are allowed once capture start.

Note: For this Start/Stop button to work, a pause has been inserted. When a very large block of data is used, this would affect the Stop button reaction, which takes a fair bit of time until the existing acquired block is plotted and computed. So do not press on the Stop button many times during this intensive process.

The "Refresh Rate" indicates how long it takes to acquire that block of data. It is only for information, doesn't carry any significant meaning as it depends on your PC system and network speed.

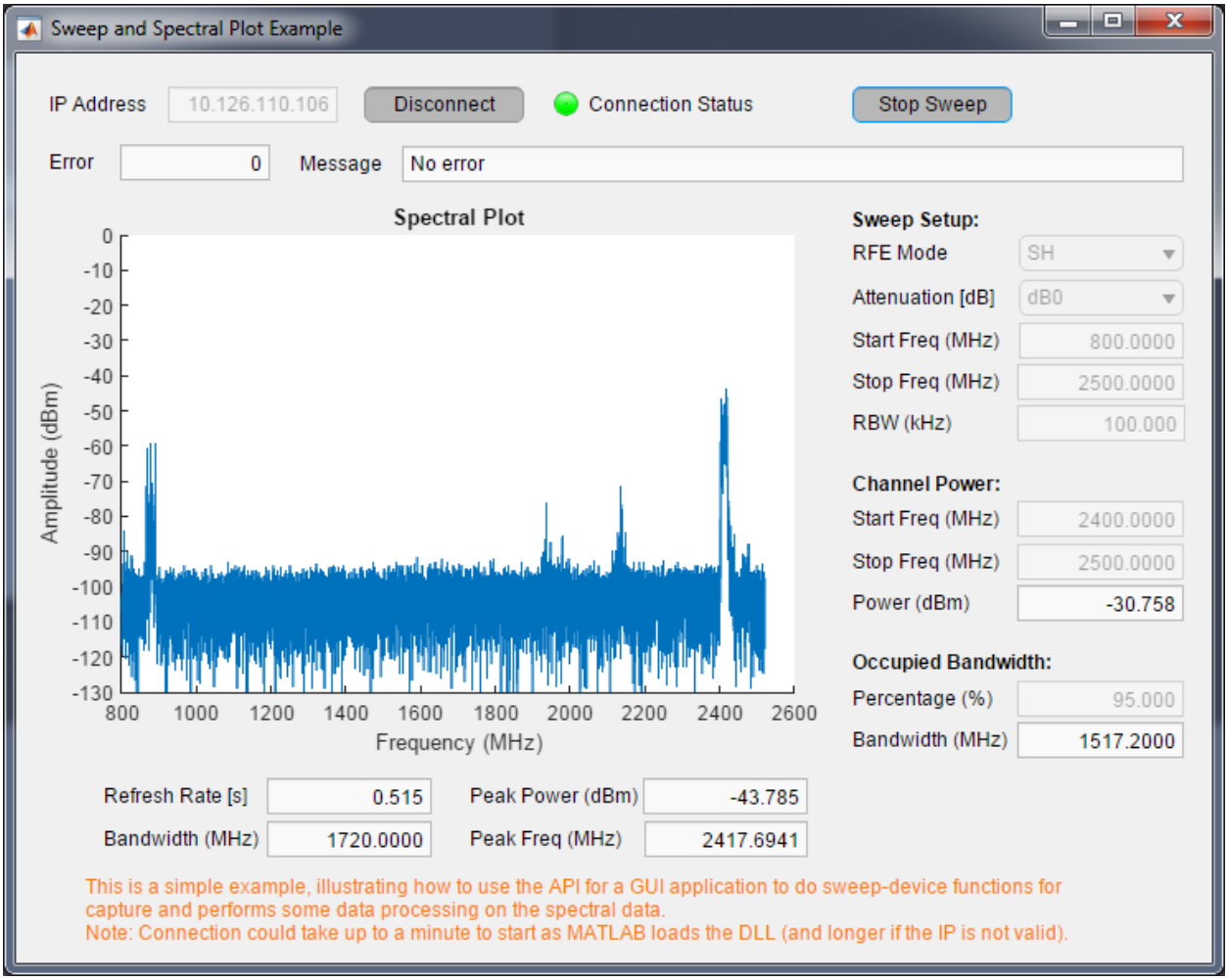


SweepDeviceExample.mlapp

This example connects to a device, performs a sweep spectral capture using sweepAndPeakFind(), follows by other data processing on the result. After connection is established, all the Drop Down fields will be filled with values for selection. The input fields of "Channel Power" and Occupied Bandwidth" could be left as default if not used.

Once the settings are set, press on **[Start Sweep]** button to start the sweep capture. The program will continuously update the Graph until you click **[Stop Capture]** button. No live settings changed are allowed once sweep capture start.

The "Bandwidth" field reflects the actual sweep start/stop range set and usually is a multiple of the RFE mode's operating bandwidth. Therefore, the "Bandwidth" value might be different than the range of the user's Start/Stop Frequencies.



Script Examples

In addition to the GUI examples, some script examples (*.m files) could be found in `..\examples\scripts` folder. They are provided to illustrate how to easily integrate the API in your own MATLAB script. Also, the script examples are supported by any MATLAB versions.

Notes:

- Before using these examples, make sure the '`IP = `' line of the script is replaced with your RTSA device's IP first.

- If running a script failed and a proper disconnect/destruct process has not been evoked, make sure "Clear Workspace" is applied first before running the script again. Otherwise, this error will occur:

```
Error using calllib
Library was not found
...
```

readIQ.m

This example configures the device and performs a trace 'block' capture (read one or multiple packets of I/Q data in succession) and displays them in a time-domain plot. It uses traceCapture() method.

readDataAndLog.m

This script connects to a RTSA device and configures it for a read Opertaion. After the read operation, the Data gets written into a log file.

captureTraceSpectrumExample.m

The readMulti Script connects to the RTSA device and configures it for multiple reads. It will then execute the read of multiple packets and display the acquired data.

sweepDeviceExample.m

Connects to the device, performs a sweep and displays the data.

Document Revision History

This section summarizes document revision history.

Document Version	Release Date	Revisions and Notes
v1.0	July 02, 2019	First release for RTSA MATLAB API Reference Guide for API v2.0.0

Contact Us

ThinkRF Support website provides online documents for resolving technical issues with ThinkRF products at http://www.thinkrf.com/resources.

For all customers who hold a valid end-user license, ThinkRF provides technical assistance 9 AM to 5 PM Eastern Time, Monday to Friday. Contact us at https://www.thinkrf.com/support/ or by calling **+1.613.369.5104**.