Using the ThinkRF Real-Time Spectrum Analyzer with MATLAB
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 (v4.2.0 or higher) 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). It could also be downloaded independently, added to the MATLAB API and applied it by run the loadDLL.m file

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 and its sub-folder is added to the MATLAB path. To make sure the path is present, run the following code in MATLAB:

```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.

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

Before using the examples provided, make sure to add this folder and the sub-folders to the MATLAB path as well.
API Overview

The API is built as a class called rtsaInterface() with properties and methods to perform setup and data acquisition tasks. Listed below is an overview of the class data, methods and structs available in rtsaInterface.m, convenience for quick referencing.

- **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
    - readPersist
  - **VRT Context Packet Structs**
    - vrt_header
    - vrt_context
    - vrt_gnss_geolocn

- **Methods**
  - **Construct/Destruct**
    - rtsaInterface()
    - delete()
  - **Connect/Disconnect**
    - connect() / disconnect()
  - **SCPI Command**
    - querySCPI()
    - getIDN()
    - getTemperatures()
    - getSCPIError()
    - setSCPI()
    - systemReset()
    - systemFlush()
  - **GNSS Related**
    - enableGNSS()
    - getGNSSStatus()
    - getGNSSPosition()
    - getGNSSFixSource()
    - setGNSSAntennaDelay()
    - setGNSSAntennaDelay()
    - setGNSSConstellation()
    - getGNSSConstellation()
    - setReferencePPS()
    - getReferencePPS()
    - readGNSSContextPkt()
  - **Trace Config and Data Capture**
    - getReadAccess()
    - setTraceAttenuation()
    - getTraceAttenuation()
    - traceConfigure()
    - setTraceLevelTrigger()
    - traceCapture()
    - traceCaptureAndContext()
    - captureTraceSpectrum()
    - captureTraceSpectrumAndContext()
    - streamStart()
    - streamStop()
    - streamTraceCaptureAndContext()
    - computeUsableTraceSpectrum
  - **Manual Sweep and Capture**
    - manualSweepStart()
    - manualSweepStop()
    - manualSweepTraceCaptureAndContext()
  - **Sweep-Device**
    - sweepDeviceSetup()
    - captureSweepSpectrum()
    - sweepAndPeakFind()
  - **Other Data Capture**
    - armTraceCapture()
    - readIFPacket()
    - readVRTPacket()
The properties are mostly intended as a storage space in the API storing relevant data, but they can be accessed to get more information about the session and the acquired data. See rtsaInterface.m for the description of each component listed.

Quick start

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

Make sure to add the API, include and examples folders as well as their sub-folders to the MATLAB path before starting.

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 acquisition:

   Configure the trace setup:
   
   Interface.getReadAccess()
   Interface.traceConfigure()

   Set frequency level trigger if needed:
   
   Interface.setLevelTrigger()

4. For stream acquisition:

   Initiate streaming:
   
   Interface.streamStart()

Acquire time-domain data only:
   
   Interface.traceCapture();
   
   or

Acquire time-domain data with context:
   
   Interface.streamTraceCaptureAndContext()

   Then to compute PSD data::
4. For sweep-device acquisition:

- Configure the sweep-device:
  - `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.
Usage Notes

This section lists any special notes or warnings that a user should be aware of when using the API.

1. When performing a sweep-device over a very large span (typically 7GHz or higher), usage of small RBWs under 10Hz could result in a memory allocation failure as your computer might not have enough RAM space to handle the allocation.

2. Usage of some sweep-device's RBWs (such as 2kHz) for `captureSweepSpectrum()` could result in a slow sweep (when compared to 1kHz) due to the FFT function usage that is not optimized for non-power of 2 FFT length. Hence, when RBW values that give raise to FFT length at or closer to power of 2, the sweep data processing will be more optimal.

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 examples 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.

![Connect/Disconnect Example](image)

**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.

![Spectrum Graph](image)

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 the Start/Stop button to work, a pause has been inserted. When a very large block of data is used, this might affect the Stop button reaction due to the time it takes to process the acquired large data. Do not press 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.

**Note:** For the Start/Stop button to work, a pause has been inserted. When a very large block of data is used, this might affect the Stop button reaction due to the time it takes to process the acquired large data. Do not press the Stop button many times during this intensive process.

The "Bandwidth" field reflects the actual sweep start/stop range set, which is adjusted to the tuning resolution; therefore, the 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.

Where applicable, only capture methods are mentioned for each example listed here for quick reference.

**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:

```plaintext
Error using calllib
Library was not found
...
```
**readIQ.m**
This example configures the device, 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 for the capture.

**readIQandGNSSData.m**
This example configures a RTSA with GNSS module to, performs a trace 'block' capture (read one or multiple packets of I/Q data in succession) along with GNSS VRT data coming back and displays them in a time-domain plot. It will throw errors if test with a non-GNSS RTSA model.

It uses many GNSS configuration methods along with `readGNSSContextPkt()` and `traceCaptureAndContext()`.

**readDataAndLog.m**
This script connects to a RTSA device and configures it for a 'block' read operation. After the read operation, the Data gets written into a log file.

It uses `traceCapture()` method for capture.

**captureTraceSpectrumExample.m**
This script connects to the RTSA device and configures it for a trace 'block' capture. It will then compute the spectral data and return spectral data of the usable bandwidth range for display.

It uses `traceCaptureSpectrum()` method for the capture.

**captureTraceSpectrumAndGNSS.m**
This simple example configures the device and performs trace 'block' capture in a loop, follows with computing the power spectral density data, some signal processing and plot the results. If a VRT context is detected during the read, it will display the context.

It uses `streamTraceCaptureAndContext()` method for the capture.

**captureStreamTraceExample.m**
This script connects to the RTSA device and configures it for a trace 'stream' capture, follows with computing the power spectral density data, some signal processing and plots the results. If a VRT context is detected during the read, it will display the context.

It uses `traceCaptureSpectrum()` method for the capture.

**manualSweepCaptureExample.m**
This script connects to the RTSA device and configures it for a manual 'sweep' capture of I/Q data and plots the results. If a VRT context is detected during the read, it will display some of the context.

It uses `manualSweepTraceCaptureAndContext()` method for the capture.
sweepDeviceExample.m

This examples connects to the device, performs a sweep-device capture, does some data processing and displays the data and results.

It uses sweepDeviceSetup() and sweepAndPeakFind() for the sweep capturing.
# Document Revision History

This section summarizes document revision history.

<table>
<thead>
<tr>
<th>Document Version</th>
<th>Release Date</th>
<th>Revisions and Notes</th>
</tr>
</thead>
<tbody>
<tr>
<td>v1.0</td>
<td>July 02, 2019</td>
<td>First release for RTSA MATLAB API Reference Guide for API v4.0.0</td>
</tr>
</tbody>
</table>
| v2.0             | Nov 26, 2019  | - Updated to include new changes mentioned in API v4.1.0 release  
                     - Removed rftsInterface section as its content repeat that of rtsaInterface.m |
| v3.0             | Feb 18, 2020  | - Updated to include new changes mentioned in API v4.2.0 release  
                     - Inserted stream acquisition steps in Quick start and its example captureStreamTraceExample.m under Script Examples |
| v3.1             | April 23, 2020| Added Usage Notes section |
| v3.2             | Dec 04, 2020  | Added new manual sweep functions:  
                     a) manualSweepStart()  
                     b) manualSweepStop()  
                     c) manualSweepTraceCaptureAndContext()  
                     Updated Quick start to include these functions usage. |

---

## Contact Us

ThinkRF Support website provides online documents for resolving technical issues with ThinkRF products at [http://www.thinkrf.com/resources](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/](https://www.thinkrf.com/support/) or by calling +1.613.369.5104.

©2017-2020 ThinkRF Corporation, Ottawa, Canada, thinkrf.com

Trade names are trademarks of the owners.

These specifications are preliminary, non-warranted, and subject to change without notice.