API Reference Manual
75-0048

libtrf - thinkRF devices API

API Reference Manual

Thu May 25 2023

thinkRF
monitor. detect. analyze

Contents

Chapter 1

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 standalone, distributed or remote deployment.

thinkRF RTSA devices provide a low-level interface via SCPI and VITA49 protocols. Whilst useful in some scenarios, for many applications a higher-level interface is required that allows users to obtain high performance from the RTSA without being exposed to low-level protocol and programming details. Furthermore, future thinkRF RTSA and other devices will increasingly make use of other interfacing technologies where open protocols are less accessible or inconvenience to end-users.

To enhance user experience and provide a future-proof path forwards for users of thinkRF RTSA current and future equipment and other devices, *libtrf* API has been developed. The libtrf library provides a one-stop programmer-friendly, cross-platform, expressive and powerful common interface to thinkRF devices with optimal performance. In addition, it provides common and advanced signal processing functions handy for end-users who wish to perform further signal analysis of the captured signals.

The libtrf API is provided as a dynamic library (.dll/.lib for windows, .so for linux environments) with 'C' bindings. thinkRF currently provides a python binding based on the 'C' shared library interface, called pyRF4.

1.1 Software and System Requirements

To use *libtrf* in your application, the following is required:

- Operating System:
  - Windows 10/11 64-bit operating system (use libtrf.dll, libtrf.lib)
  - Linux x86_64 (Intel/AMD) or aarch64 (ARM v8+). Ubuntu 20.04LTS or later is recommended (use libtrf.so)
- Access to an RTSA product. If don't have a unit, you may request access to a demo unit with support@thinkrf.com.

1.1.1 Supported RTSA Product List

libtrf version 1.4.0 or higher supports the following RTSA products and their associated models (-408, -408P, -418, -427):

- R5500
- R5550

- R5700
- R5750

For prior-generation thinkRF receivers, the legacy APIs are available. The legacy APIs, however, are not recommended for future development.

1.2 Using the API

The API provides a 'flat' C-language interface to all functionality. The functions are published via a single header file called libtrf.h and are implemented via a single self-contained shared library binary libtrf.[so|dll]. Refer to Software and System Requirements section for specifications & recommendation.

Example of how to link with libtrf on Windows and Linux are provided in the included 'Examples' directory. In particular, see the 'README.txt' file in that directory.

1.2.1 Using libtrf on Windows

To use libtrf directly in your application, you should include the 'libtrf.h' header file. Linking your C/C++ application with the supplied 'libtrf.lib' will provide the required stub functions (for the libtrf.h header) to access the shared library functions and will provide for start-time. Normally, it is recommended that libtrf.dll be placed in the same directory as the your application binary to facilitate runtime loading.

1.2.2 Using libtrf on Linux

To use libtrf directly in your application, you should include the 'libtrf.h' header file. The linker option '-ltrf' should be specified to cause the linker to reference libtrf.so - which should be referencable via the LD_LIBRARY_PATH environment variable. Note that for linking from the same directory as your binary file, executing 'export LD_↩ LIBRARY_PATH=$LD_LIBRARY_PATH:.' is typically sufficient.

1.2.3 Using Other Language Bindings

Currently, only a python binding called pyrf4 is developed for python users. Its capabilities might be lagging behind those of libtrf; however, it is an open-source code and thus, is free for python end-users to modify or update as see fit. See https://thinkrf.com/software-apis/ or reach out to support@thinkrf.com for further information.

1.3 Quick Start with Examples

The easiest way to get started with the API is to take a look at the rich set of examples provided in the "Examples" folder included with the API release package. The examples include different capture modes (spectrum sweeping, IQ capture, finite-time capture, capture with file, etc) along with build instructions for Windows and Linux environments.

The example code is heavily commented and it is recommended that you make use of this code to understand the functionality of the libtrf library calls and how they interact.

1.3.1 Building Example Code on Windows

1. Assuming Visual Studio is installed; in the windows start area, navigate to the install directory for your version of Visual Studio (ie. Visual Studio 2017). From there select the 'x64 Native Tools Command Prompt'.

2. In the command window that is launched, navigate to the 'examples' directory of your libtrf installation. To build any of the examples, enter (for example):
   cl CaptureToFile.cpp ../bin/x64/libtrf.lib -I ../source/inc /link /out:example.exe
   This will generate example.exe in your current directory.
3. To enable run-time linking with the libtrf shared library, libtrf.dll should be copied into the current directory:
   copy "..\bin\x64\libtrf.dll" .
4. Type example.exe to run the example application code.

1.3.2 Building Example Code on Linux

1. Assuming that gcc/g++ is installed (version 10 or later recommended). To build any of the examples, the LD_↩ LIBRARY_PATH environment variable should be correctly set to reference libtrf. For example, if your installation is in your home directory then in a shell you can type the following to correctly set LD_LIBRARY_PATH for build/execution with libtrf:
   export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:∼/libtrf/bin/_arch_/
   Note that this command is required every time you open a new shell to build with libtrf.
2. Build the example using the command line (for example):
   g++ CaptureToFile.cpp -o example -I ../source/inc -L ../bin/linux/x86_64 -ltrf -lpthread
3. Type ./example to run the example application code.

1.4 Contact Us

thinkRF Support website provides online documents for resolving technical issues with thinkRF products at https://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://support.thinkrf.com/

Chapter 2

libtrf

The distribution of libtrf comprises this programmers' Reference manual, a C header file and architecture-specific binary release files of the shared library.

The release package (libtrf_<version>.zip / libtrf_<version>.tar.gz) unpacks into the following structure beneath the root 'libtrf' directory:

- doc/
  - Release Notes pdf - *plaintext release notes*
  - Reference Manual pdf - *this document*
- examples/ - *source cpp code examples directory*
- inc/
  - libtrf.h - *library header file*
- bin/
  - win/x64/ - *windows binary files*
    - libtrf.lib
    - libtrf.dll
  - linux/
    - x86_64/ - *64-bit x86 architecture binaries*
      - libtrf.so
    - aarch64/ - *64-bit ARMv8+ binaries*
      - libtrf.so

2.1 Use of JSON for Parameters

A design challenge for any library with the scope of libtrf is to provide a means of accessing parameters from a set that may be modified over time for different devices, streams, processor types and in subsequent releases of the library.

To prevent an explosion of library calls to manage this parameter space, libtrf instead uses the well-known JSON (JaveScript Object Notation) open standard format (see https://datatracker.ietf.org/doc/html/rfc8259 - RFC8259 for details).

The library reports parameter information using JSON notation and allows parameters to be examined and modified by constructing and supplying JSON strings. The library provides utility functions for the creation and management

of JSON strings where this is not supported by user code. Also, syntactic-sugar is provided in the form of 'canned' functions, which allow easy encoding of common parameters and ensure that typically-used parameters are included directly where needed.

2.2 Devices, Streams, Processors

To make best use of libtrf, it is useful to understand the core abstractions that the library is based on. This model comprises three primary classes of entities; devices, streams and processors. Collectively these are referred to as libtrf 'entities'.

Entities are typically created when needed by a user, manipulated by setting and reading parameters, and may be connected to produce user-desired processing structures.

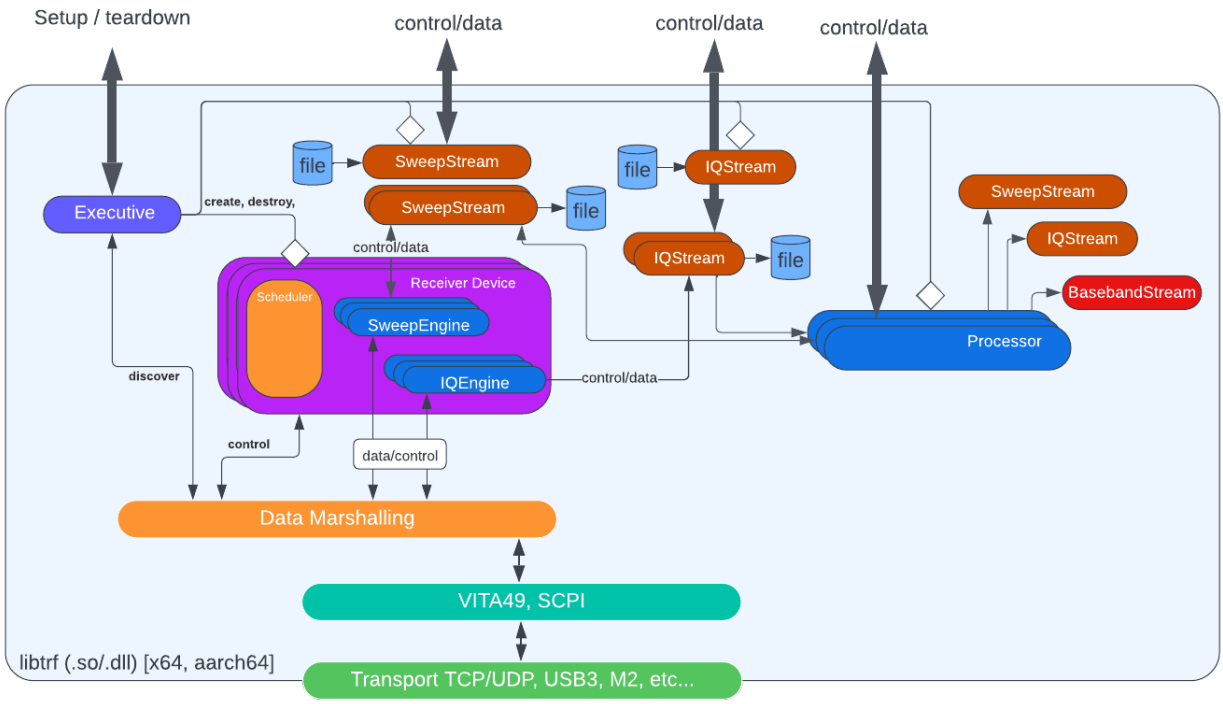

Figure 2.1 libtrf architecture

2.2.1 Handles

All entities within libtrf are referred to using a 'handle' (type trfHandle). The value of a handle is created when its matching entity is created, and its value becomes invalid when that entity is destroyed.

2.2.2 Devices

A device provides some network-controllable (or in the near furute USB) functionality. A device may be a source of data of different kinds (ie. spectrum information, digitized radio (IQ) data). Also, it is possible that a device does not source radio data directly and implements other functionality (such as an up-converter). A device could also consume radio data (ie. a transmitter).

In the current version of libtrf, the only device supported is an RTSA (ie. receiver); however, the design of the library permits other device types that will be progressively support and added in subsequent releases.

2.2.3 Streams

A stream represents a time-based sequence of data. A user may access a stream to recover sampled data from a receiver device. Also, streams may write to and read from files. A stream may contain any type of data. The libtrf library recognizes three types of streams: Spectrum, IQ and Baseband.

As well as a conduit of data, streams have an important control function. A receiver device can only provide data through a stream. Obtaining data involves creating a suitable stream to hold that data, and attaching it to a source device. If the attach is successful, the receiver device will continue to produce data into the stream until the stream is detached from the receiver device. Note that functionality is provided to allow streams to auto-detach once a certain amount of data (time, frames) has been obtained, as well as for them to continue indefinitely.

A high-performance receiver can produce data at very high rates. In particular the R5xx0-family support an internal sampling rate of 125MHz complex samples per second, representing a raw uncompressed data rate of 480MB/sec. For this reason, streams incorporate buffering to allow for client applications to achieve uninterrupted processing in the face of operating system delays etcetera. At the same time, this buffering is constrained to ensure that libtrf maintains a reasonable memory footprint even under high processing loads. The depth of buffering in a stream defaults to a small value (10 frames) and may be programmed by the user in terms of frames (TRFBufferFrames - most suitable for spectrum data) or time (TRFBufferSec - most suitable for IQ or baseband data) to suit application and target system requirements.

In the case that the buffer size is exceeded, a stream will discard the oldest data up to some 'discard proportion' limit set by the parameter TRFDiscardProportion. If set to zero (default), the stream will discard only the oldest data required to bring the buffer size back within limits. Values up to 1.0 allow the proportion of the current buffer to be discarded in the case of an overrun to be controlled, up to 100% of the buffer. This allows a user to, for example minimize the number of discontinuities in received data by performing large/rare discards rather than small/frequent discards when required.

An additional property of streams is that they may be set as 'non-discardable' (see trfAllowDiscarding()). In this case, if a stream buffer becomes full, the upstream source will be blocked until data is retrieved from the stream. If the user knows that they will not be obtaining data directly from a stream, setting the 'discardable' flag on the stream will be required to ensure that data continues to flow (trfAllowDiscarding()). Additionally, streams that are sourced directly from a file are set as non-discardable by default to allow a user to guarantee that they see all data from that file and block reading from the file in case they are unable to keep up with the data source. All streams sourced from a receiver are set as 'discardable'.

Stream functionality and control are illustrated in several key scenarios in the set of examples included with the release. These examples provide a rich illustration of the use of stream features.

Stream status return values

The functions used to retrieve data from streams (trfGetNext___) return three special information values, describing the state of the stream when no data is returned:

- trfNotStarted - This indicates that no data has yet been retrieved from the stream, and data is expected.
- trfWaiting - Some data has been received, however the next frame is not yet available.
- trfExhausted - No more data is expected from the stream without a configuration change. This occurs when a stream has been detached from its source (device, file) and its internal buffering is now empty.

The occurrence of these return values is not an error, and instead can prompt user code into responding appropriately to the conditions they indicate. Note that the timeout parameter specified in the trfGetNext___ calls can be set large (ie. 5000 msec). This has no impact on performance as the call will return as soon as data is available, however it will minimize the return of 'nuisance' trfNotStarted, trfWaiting values if this is important to user code. Note that a timeout of 0 will return immediately if no data is available, allowing a non-blocking design approach to be used.

Spectrum Streams

A spectrum stream consists of a sequence of 'frames' of spectrum data. Each frame represents a power-spectrum between a start and stop frequency, regularly sampled. The metadata describing a spectrum frame also details how it was created; specifically its resolution bandwidth and window type. Additional parameters include a timestamp and sequence number. Spectrum frames may also contain additional spectrum data (such as min, max, average) that may be added by processors.

Spectra are created through a process of successive steps of retuning and sampling. It should be noted that the amount of memory consumed by the reconstruction process may become large if narrow resolution bandwidths and high spans are requested. An upper bound on memory requirements for a particular span and resolution is (reciever devices):

$$UpperBound_{bytes} = (\frac{Span_{MHz}}{38} + 1) \times (3 \times \frac{62500}{RBW_{KHz}}) \times 8$$

IQ Streams

An IQ stream consists of a sequence of arbitrary-sized frames consisting of a contiguous sequence of IQ samples. The metadata included describes the number of samples, the usable IF bandwidth, the sample rate, frame sequence number and timestamp.

Baseband Streams

Through the actions of a demodulator (processor), IQ data may be processed to yeild baseband (ie. audio) data. The baseband frame type provides a sequence of contiguous real samples. The metadata in a baseband frame describes its size, usable bandwidth, sample rate, sequence number and timestamp.

2.2.4 Processors

A processor encapsulates functionality processing a stream (typically IQ or Spectrum) to produce an output stream (IQ, Spectrum, Baseband) and optionally output parameters. Examples in the current version of libtrf include AM and FM demodulators, IQ-to-spectrum conversion and spectrum characterization functionality.

AMDemodulator Processor

See examples AMDemodulatorExample.cpp

The AMDemodulator processor implements a conventional Dual-sideband, Upper-sideband (USB) and Lower-sideband (LSB) AM demodulator. The demodulator operates on an input IQ stream and produces an output baseband stream. The parameters of the demodulation are controlled by the TRFAMSubtype parameter and TRFOutputSampleRate.

FMDemodulator Processor

See examples FMDemodulatorExample.cpp

The FMDemodulator processor implements a conventional FM demodulator. The demodulator operates on an input IQ stream and produces an output baseband stream. The output sample rate is the only control parameter, set by TRFOutputSampleRate.

IQToSpectrum Processor

See examples IQToSpectrumExample.cpp

This processor takes an IQ stream as input, and produces a spectrum stream as output. The conversion from IQ to spectrum data is controlled by the following set of parameters:

- TRFFollowIQ - if set 'true' then one spectrum frame will be produced for each input IQ frame. this effectively sets TRFOverlap to zero and TRFRBWHz is ignored.
- TRFWindow - selects the window function to use in spectrum reconstruction.
- TRFOverlap - sets the degree of overlap between successive spectra from the input stream. A value of 0.5 implies 50% overlap of the input data stream in successive output spectra. Default is zero.

The processor produces the following read-only parameters:

- TRFMatchIQ - the sequence number of the most recently-processed input IQ frame.
- TRFOutputSize - the expected spectrum output size.

SpectrumCharacterization Processor

See examples SCProcessorExample.cpp and SCProcessorFromFileExample.cpp

This processor implements the commonly used functionality of producing max, average and mimimum spectrum traces. The output spectrum stream thus consists of the input spectrum, plus 3 'passenger' spectra for the average, minimum and maximum. The operation of this processor is controlled by the following parameters:

- TRFAverageCount - Sets the number of spectra over which an average is to be computed. Note that the processor stores this number of spectra internally to perform the sliding-window average, and therefore large numbers and/or large spectra may incur a large memory footprint.
- TRFClearAverageTrace - Setting this to 'true' will cause the average trace to be reset.
- TRFClearMaxHoldTrace - Setting this to 'true' will cause the max-hold trace to be reset.
- TRFClearMinHoldTrace - Setting this to 'true' will cause the min-hold trace to be reset.

The output spectrum frame will hold 3 'passenger' spectra. The indexes of these are contained in the output spectrum 'pJSONInfo' field. This field will also contain the current state of the TRFClearAverageTrace, TRFClearMaxHoldTrace, TRFClearMinHoldTrace valid when the output frame was created. These are included to aid in downstream synchronization.

2.3 Memory considerations

High-bandwidth receivers such as the thinkRF devices that libtrf supports can generate large data rates and hence care should be taken to manage the storage requirements and lifetime of data to ensure that memory constraints on any particular platform and for any particular application are respected.

2.3.1 IQ capture

Capturing IQ data will consume memory at the capture sampling rate, where each sample is 8 bytes (2 32-bit floating point numbers representing a complex sample). Inside libtrf, IQ data is buffered within streams up to a default value of 10 frames which minimizes overhead whilst minimizing the possibility of frame loss for most applications. As a worst-case for R5xx0 devices this represents 64k $*$ 8 bytes $*$ 10 frames or 5MBytes. This limit can be changed using either the 'TRFBufferFrames' or 'TRFBufferSeconds' parameters of IQ streams.

2.3.2 Spectrum capture

libtrf can produce very fine resolution spectra. It is easily possible for a user to create data structures with a very large memory footprint. For example, a 27GHz spectrum at 10kHz resolution bandwidth with a Nuttall window is represented by approximately 21MB of memory. The memory requirement is directly proportional to the spectrum width, and inversely proportional to the resolution bandwidth chosen. In this case a 10-frame buffer ('TRFBufferFrames' parameter) occupies 210MB of memory. The libtrf functions do not artificially constrain the user from specifying values that may require excessive amounts of memory. The user should therefore be aware of memory requirements.

2.3.3 SpectrumCharacterization processor

Of particular note for memory usage is the SpectrumCharacterization processor. Internally this processor maintains a list of past spectra equal to the size of the averaging window. With narrow/wide spectra, this internal structure can very rapidly take up memory. In the case of our 27GHz/10kHz spectrum above, an averaging window of 100 frames would consume 2100MB. Care should therefore be taken in setting the TRFAverageCount parameter of this processor.

2.4 libtrf Structure

The libtrf library calls are divided into error handling and diagnostics, device functions, stream functions, processor functions and parameter manipulation.

2.4.1 Error Handling and Diagnostic Functions

The following functions provide support for debugging.

- trfGetTextForStatus() - Interpret a libtrf status code as text.
- trfGetNextStatus() - Get any additional status codes relating to the last error returned.
- trfGetDetailedStatus() - Returns a string providing more detailed, potentially nested error information.

2.4.2 Device Handling Functions

The following provide means of discovering, interrogating and managing access to devices:

- trfAddDeviceAddress() - Add a (remote) device to the device addressing table.
- trfEnumerateDevices() - Enumerate all devices, returning a count. Any directly-connected devices will be automatically returned in this list.
- trfExamineDevice() - Return all information known about an enumerated device.
- trfOpenDevice() - Open an enumerated device, returning a device handle.
- trfResetDeviceConnection() - Reset the network connection with a device. This provides a means of recovering a device in the case of a network or other system error.
- trfClose() - Close any libtrf handle (device, stream, processor).

2.5 Stream and Stream Data Handling Functions

Receiver devices deliver data in streams of different kinds. These family of functions provide means of creating, destroying and obtaining data from streams.

- trfCreateIQStream() - Create an unattached IQ stream.
- trfCreateSpectrumStream() - Create an unattached spectrum stream. Note: Instances of BasebandStream are only supported as output of Processor instances and hence are only obtained from querying a processor instance.

2.5.1 Stream Frame Handling Functions

- trfGetNextSpectrum() - Get the next spectrum (frame) from the specified stream.
- trfFreeSpectrum() - Dispose of a spectrum frame returned by trfGetNextSpectrum.
- trfGetNextIQ() - Get the next IQ (frame) from the specified stream.
- trfFreeIQ() - Free an IQ frame returned by trfGetNextIQ.
- trfGetNextBaseband() - Get the next baseband frame from the specified stream.
- trfFreeBaseband() - Dispose of a baseband frame returned by trfGetNextBaseband

2.5.2 Streams and Files Functions

- trfSetStreamOutputFile() - Attach a file to a stream to receive its contents or detach the stream from the current file.
- trfAttachStreamToFile() - Source a stream from the specified file (non-discarding by default).

2.6 Stream Attach/Detach and State Functions

- trfAttachStreamToDevice() - Attach a specified stream to a given device. This operation if successful will start the flow of data into the stream.
- trfDetachStream() - Detach a stream from any entity it is attached to, halting the flow of data.
- trfFlushStream() - Flush all data currently in the stream.
- trfIsAttached() - Determine if the stream is attached to a data source and hence is expected to continue to deliver data.
- trfAllowDiscarding() - Controls the discarding behaviour of a stream. By default all streams from receiver devices are 'discarding' - that is if the stream buffer size overruns its limits (see TRFBufferFrames, TRFBufferSec) then the oldest frames will be discarded (subject to the discarding policy set by TRFDiscardProportion) to return the buffer within size limits. Conversely, streams sourced from files are set as 'non-discarding' which will cause the file reading process to block if frames are not consumed from the stream sufficiently quickly. In the case a user knows that a file-sourced stream will not be read directly (ie. it is sourcing a processor), then the user should set that stream to be 'discarding' to allow the read process to proceed without blocking. For an example of usage, see the SpectrumCharacterizationProcessorExampleFromFile.cpp example code.

2.6.1 Special File Handling Functions

- trfConvertBasebandFileToWavFile() - Converts a captured baseband data file into an AIFF 'wav' file suitable for use with typical audio-handling utilities.

2.7 Processor Handling Functions

- trfGetProcessorTypes() - Returns a description of the set of available processor type names.
- trfCreateProcessor() - Given a valid processor type name, creates an instance of that processor type.
- trfAttachProcessorToStream() - Attach the specified processor to a source stream. Note that the stream type must be that accepted by the processor or an error will be returned.
- trfDetachProcessor() - Detaches the specified processor from its source stream.
- trfGetProcessorOutputStream() - Return the output stream (if any) supported by the processor. Different processor types will support different output stream types.

2.7.1 Parameter Handling (JSON) Functions

Parameter management is performed by passing and obtaining strings from libtrf entities using the JSON notation. The following calls provide for the creation, examination and application of JSON parameter sets. All parameters may be read/written using their 'TRFxxx' parameter name as defined in libtrf.h.

A user is not required to use these functions as libtrf will recognize and generate valid JSON strings in UTF-8 encoding, captured as conventional C (ie. char $*$, null-terminated) strings, allocated on the heap (ie. using 'new char[]'). Use of stack-based storage will result in undefined behaviour. All strings returned by libtrf functions are heap-allocated and may be conveniently disposed of using the trfDisposeString function.

- trfGetParameterInfo() - Return all information known about the public parameters of a libtrf entity (device, stream, processor).
- trfGetParameters() - Return the values of all entity public parameters.
- trfSetParameters() - Apply a JSON parameter set to a libtrf entity (device, stream, processor).
- trfReadParameterInfoElement() - Extract a nested JSON description of a particular parameter from a parameter info string.
- trfReadParameterAsJSON() - Extract a JSON parameter as a new JSON string.
- trfDisposeString() - Utility function to dispose of JSON (or other) strings returned by libtrf functions. Note that this function supersedes trfDisposeJSON from earlier versions of libtrf.

All parameter names recognized by libtrf entities are listed in libtrf.h and are of the form 'TRF<name>'. The following functions allow the creation of JSON strings from parameter names and values:

- trfAddNumericParameter() - Add a named numeric parameter to a JSON parameter set string.
- trfAddTextParameter() - Add a named textual parameter to a JSON parameter set string.
- trfAddBooleanParameter() - Add a named boolean parameter to a JSON parameter set string.

The following functions allow the examination of JSON strings, extracting named parameters:

- trfReadNumericParameter() - Extract a named numeric parameter from a JSON parameter set string.
- trfReadTextParameter() - Extract a named textual parameter from a JSON parameter set string.
- trfReadBooleanParameter() - Extract a named boolean parameter from a JSON parameter set string.

Commonly-used Parameter Functions

To aid with creating self-documenting code, the following 'syntactic-sugar' functions are provided that encapsulate the setting of particular named parameters. This functionality can be also be realized using trfAdd<...> and trfRead<...> calls above using the appropriate parameter names. The following functions are provided for ease of programming when encoding and examining commonly-used parameters.

- trfAddFCentre()
- trfAddIFBWHz()
- trfAddFMinMax()
- trfAddRBWHz()
- trfAddWindowType()
- trfReadFCentre()
- trfReadIFBWHz()
- trfReadFMinMax()
- trfReadRBWHz()
- trfReadWindowType()
- trfReadSampleRateHz()
- trfReadRefLeveldBm()

2.8 The Rest of this document

The remainder of this document provides a detailed description of all the functions, structures and definitions provided by libtrf. It also documents the example code provided with libtrf and provides detailed build instructions for windows and linux environments.

Chapter 3

Data Structure Index

3.1 Data Structures

Here are the data structures with brief descriptions:

Chapter 4

File Index

4.1 File List

Here is a list of all files with brief descriptions:

Chapter 5

Data Structure Documentation

5.1 _complex32 Struct Reference

Complex type consisting of 32-bit float Real (In-phase) and Imaginary (Quadrature) components.

`#include <libtrf.h>`

Data Fields

- _float32 fI

  32-bit floating point (IEEE-754) In-phase (I) sample value

- _float32 fQ

  32-bit floating point (IEEE-754) Quadrature (Q) sample value

5.1.1 Detailed Description

Complex type consisting of 32-bit float Real (In-phase) and Imaginary (Quadrature) components.

5.1.2 Field Documentation

fI

`_float32 _complex32::fI`

32-bit floating point (IEEE-754) In-phase (I) sample value

fQ

`_float32 _complex32::fQ`

32-bit floating point (IEEE-754) Quadrature (Q) sample value

5.2 _trfBasebandFrame Struct Reference

Baseband sampled data (ie. audio) frame. Basedband data can be of any type representable by an array of _float32 values. Multiple channels are supported.

`#include <libtrf.h>`

Data Fields

- _uint32 uSequenceNumber

  Frame sequence number (monotonic unit-step)
- _int64 iTimestampNanosec

  Frame origination timestamp.
- _float32 fSampleRateHz

  Sample rate in Hz.
- _float32 fUsableBWHz

  Usable bandwidth of sampled data (is less than fSampleRateHz/2)
- _uint32 uChannels

  Channels in pData. Channel0 at offset 0, 1 at offset uSamples etc...
- _uint32 uSamples

  Samples in frame.
- const _float32 ** ppData

  ppData[0] - channel0, ppData[1] - channel 1 etc... Frame data (range +/-1.0f). Array terminates with a NULL pointer.
- const char * pJSONInfo

  Any additional information for this frame (ie. from processing)

5.2.1 Detailed Description

Baseband sampled data (ie. audio) frame. Basedband data can be of any type representable by an array of _float32 values. Multiple channels are supported.

5.2.2 Field Documentation

fSampleRateHz

`_float32 _trfBasebandFrame::fSampleRateHz`

Sample rate in Hz.

fUsableBWHz

`_float32 _trfBasebandFrame::fUsableBWHz`

Usable bandwidth of sampled data (is less than fSampleRateHz/2)

iTimestampNanosec

```
_int64 _trfBasebandFrame::iTimestampNanosec
```

Frame origination timestamp.

pJSONInfo

```
const char* _trfBasebandFrame::pJSONInfo
```

Any additional information for this frame (ie. from processing)

ppData

```
const _float32** _trfBasebandFrame::ppData
```

ppData[0] - channel0, ppData[1] - channel 1 etc... Frame data (range +/-1.0f). Array terminates with a NULL pointer.

uChannels

```
_uint32 _trfBasebandFrame::uChannels
```

Channels in pData. Channel0 at offset 0, 1 at offset uSamples etc...

uSamples

```
_uint32 _trfBasebandFrame::uSamples
```

Samples in frame.

uSequenceNumber

```
_uint32 _trfBasebandFrame::uSequenceNumber
```

Frame sequence number (monotonic unit-step)

5.3 _trfIQFrame Struct Reference

IQ data frame. This structure contains the actual data and all descriptive metadata.

```
#include <libtrf.h>
```

Data Fields

- trfStatus eFlags

  Any warning/error for this frame only.

- _uint32 uSequenceNumber

  Frame sequence number (monotonic unit-step)
- _int64 iTimestampNanosec

  Frame origination timestamp.
- _float64 fCentreHz

  Centre frequency.
- _float32 fIFBWHz

  Usable IF Bandwidth - assumed symmetrical around fCentreHz.
- _float32 fSampleRateHz

  Sample rate in Hz - fSampleRateHz is always greater than fIFBWHz.
- bool bDiscontinuity

  'true' if frame is not contiguous with previous frame. Will not be set for the first frame of a new stream when first attached. If detached and re-attached the first frame will have the discontinuity flag set.
- bool bSubOptimalDRFlag

  'true' if a low or high excursion outside target dynamic range was detected in this frame
- _uint32 uSamples

  Complex IQ samples in ∗pData.
- const _complex32 ∗ pData

  IQ Frame data as calibrated sqrt(mW)
- const char ∗ pJSONInfo

  Any additional information for this frame (ie. from processing)

5.3.1 Detailed Description

IQ data frame. This structure contains the actual data and all descriptive metadata.

5.3.2 Field Documentation

bDiscontinuity

`bool _trfIQFrame::bDiscontinuity`

'true' if frame is not contiguous with previous frame. Will not be set for the first frame of a new stream when first attached. If detached and re-attached the first frame will have the discontinuity flag set.

bSubOptimalDRFlag

`bool _trfIQFrame::bSubOptimalDRFlag`

'true' if a low or high excursion outside target dynamic range was detected in this frame

eFlags

```
trfStatus _trfIQFrame::eFlags
```

Any warning/error for this frame only.

fCentreHz

```
_float64 _trfIQFrame::fCentreHz
```

Centre frequency.

fIFBWHz

```
_float32 _trfIQFrame::fIFBWHz
```

Usable IF Bandwidth - assumed symmetrical around fCentreHz.

fSampleRateHz

```
_float32 _trfIQFrame::fSampleRateHz
```

Sample rate in Hz - fSampleRateHz is always greater than fIFBWHz.

iTimestampNanosec

```
_int64 _trfIQFrame::iTimestampNanosec
```

Frame origination timestamp.

pData

```
const _complex32* _trfIQFrame::pData
```

IQ Frame data as calibrated sqrt(mW)

pJSONInfo

```
const char* _trfIQFrame::pJSONInfo
```

Any additional information for this frame (ie. from processing)

uSamples

```
_uint32 _trfIQFrame::uSamples
```

Complex IQ samples in ∗pData.

uSequenceNumber

_uint32 _trfIQFrame::uSequenceNumber

Frame sequence number (monotonic unit-step)

5.4 _trfSpectrumFrame Struct Reference

Spectrum data frame. This structure contains the actual spectral data and all descriptive metadata.

`#include <libtrf.h>`

Data Fields

- trfStatus eFlags

  Any warning/error for this frame only.
- _uint32 uSequenceNumber

  Frame sequence number (monotonic unit-step)
- _int64 iTimestampNanosec

  Timestamp of first sample used in spectrum reconstruction.
- _int64 iDurationNanosec

  Delta from first-sample to last-sample timestamp used in spectrum reconstruction. Note that this is not the same as the frame rate which can be derived from a stream by comparing the timestamps of successive trfSpectrumFrame instances in the stream.
- _float64 fMinHz

  Centre frequency of first bin in sweep.
- _float64 fMaxHz

  Centre frequency of last bin in sweep.
- _float32 fRBWHz

  Resolution Bandwidth in Hz.
- _uint32 uPoints

  Number of data points in ∗pData.
- const _float32 ∗ pData

  Frame data as calibrated dBm.
- const char ∗ pJSONInfo

  Any additional information for this frame (ie. from processing)
- _uint32 uAdditionalSpectra

  Number of entries in pAdditionalSpectra.
- const _float32 ∗∗ pAdditionalSpectra

  Array of uAdditionalSpectra entries.

5.4.1 Detailed Description

Spectrum data frame. This structure contains the actual spectral data and all descriptive metadata.

5.4.2 Field Documentation

eFlags

```
trfStatus _trfSpectrumFrame::eFlags
```

Any warning/error for this frame only.

fMaxHz

```
_float64 _trfSpectrumFrame::fMaxHz
```

Centre frequency of last bin in sweep.

fMinHz

```
_float64 _trfSpectrumFrame::fMinHz
```

Centre frequency of first bin in sweep.

fRBWHz

```
_float32 _trfSpectrumFrame::fRBWHz
```

Resolution Bandwidth in Hz.

iDurationNanosec

```
_int64 _trfSpectrumFrame::iDurationNanosec
```

Delta from first-sample to last-sample timestamp used in spectrum reconstruction. Note that this is not the same as the frame rate which can be derived from a stream by comparing the timestamps of successive trfSpectrumFrame instances in the stream.

iTimestampNanosec

```
_int64 _trfSpectrumFrame::iTimestampNanosec
```

Timestamp of first sample used in spectrum reconstruction.

pAdditionalSpectra

```
const _float32** _trfSpectrumFrame::pAdditionalSpectra
```

Array of uAdditionalSpectra entries.

pData

```
const _float32* _trfSpectrumFrame::pData
```

Frame data as calibrated dBm.

pJSONInfo

```
const char* _trfSpectrumFrame::pJSONInfo
```

Any additional information for this frame (ie. from processing)

uAdditionalSpectra

```
_uint32 _trfSpectrumFrame::uAdditionalSpectra
```

Number of entries in pAdditionalSpectra.

uPoints

```
_uint32 _trfSpectrumFrame::uPoints
```

Number of data points in ∗pData.

uSequenceNumber

```
_uint32 _trfSpectrumFrame::uSequenceNumber
```

Frame sequence number (monotonic unit-step)

Chapter 6

File Documentation

6.1 libtrf_doxy.txt File Reference

6.2 libtrf.h File Reference

Data Structures

- struct _complex32

  Complex type consisting of 32-bit float Real (In-phase) and Imaginary (Quadrature) components.
- struct _trfIQFrame

  IQ data frame. This structure contains the actual data and all descriptive metadata.
- struct _trfSpectrumFrame

  Spectrum data frame. This structure contains the actual spectral data and all descriptive metadata.
- struct _trfBasebandFrame

  Baseband sampled data (ie. audio) frame. Basedband data can be of any type representable by an array of _float32 values. Multiple channels are supported.

Macros

- #define trfInvalidHandle (0xffffffff)

  Invalid handle value for any libtrf entity. As good practice, all unused or uninitialized handle values should as good practice be set to this value - ie.
- #define TRFDefault "default"

  Generic JSON parameter names for libtrf entities; devices, streams, processors. These parameter names are used in conjunction with JSON string construction/ examination functions to allow parameters to be set for and obtained from libtrf entities. In the following documentation 'RO' denotes 'read-only', and RW denotes read/write. Setting a parameter for an entity that does not support that parameter is an error and will typically return trfParameterSetError when calling trfSetParameters with a JSON string including an unsupported parameter. Detailed information can be obtained using trfGetDetailedStatus. Note that these names may change across future releases and therefore it is highly recommended that the following symbolic names TRF___ are used instead of their textual equivalent. RO - Read-only, RW - Read and Write.
- #define TRFMin "min"

  RO minimum value tag.
- #define TRFMax "max"

  RO maximum value tag.

- #define TRFType "type"

  RO type of unit tag (user readable)
- #define TRFUnits "units"

  RO units of value tag.
- #define TRFAddress "address"

  RO network address tag.
- #define TRFDevice "device"

  RO device information section.
- #define TRFDeviceType "type"

  RO parameter - device textual type.
- #define TRFDeviceVersion "version"

  RO parameter - device textual version.
- #define TRFDeviceSerialNum "serial"

  RO parameter - device textual serial number.
- #define TRFDeviceFirmware "firmware"

  RO parameter - device textual firmware version.
- #define TRFDeviceConnection "connection"

  RO parameter - device textual connection information.
- #define TRFHasFlatteningInfo "iqequalization"

  RO receiver flag that equalization data is available (flattening is possible)
- #define TRFSequence "Sequence"

  RO sequence number parameter (internal)
- #define TRFTimestamp "Timestamp"

  RO timestamp parameter (internal)
- #define TRFTimeResolution "TimeResolution"

  RO time resolution parameter (internal)
- #define TRFBandwidthHz "BandwidthHz"

  RO generic bandwidth parameter.
- #define TRFFilename "filename"

  RW parameter - filename parameter (stream)
- #define TRFLooping "LoopingFlag"

  RW set to cause file to loop.
- #define TRFSweepActive "SweepActive"

  RO parameter - sweep is active flag (Rx)
- #define TRFIQActive "IQActive"

  RO parameter - IQ capture is active flag (Rx)
- #define TRFBufferSec "bufferSec"

  RW parameter - user-settable maximum buffering duration in seconds. If set this will override TRFBufferFrames.
- #define TRFBufferFrames "bufferFrames"

  RW parameter - user-settable maximum stream buffering in frames. If set this will override TRFBufferSec.
- #define TRFDiscardProportion "bufferDiscard"

  RW parameter - On overflow (TRFBufferSec, TRFBufferFrames) this sets the proportion of the buffer that will be discarded. The oldest frames will be discarded first. Setting to zero will minimize the number of frames discarded. Setting to 1.0 will discard the whole buffer. All other values 0,1 are acceptable. This allows a user to manage the tradeoff between discontinuities and data latency.
- #define TRFFMinHz "FMinHz"

  RW parameter - Min extent of sweep range.
- #define TRFFMaxHz "FMaxHz"

RW parameter - Max extent of sweep range.

- #define TRFRBWHz "RBWHz"

  RW parameter - Resolution Bandwidth in Hz.
- #define TRFWindow "WindowFn"

  RW parameter - Window function for spectrum reconstruction.
- #define TRFFCentreHz "FCentreHz"

  RW parameter - IQ Capture centre frequency.
- #define TRFSampleRateHz "SampleRateHz"

  RO parameter - IQ Capture sample rate.
- #define TRFIFBWHz "IFBWHz"

  RW parameter - IF Bandwidth for IQ capture. Note that in the current implementation of libtrf, IF bandwidths narrower than 100kHz are not supported. Additionally IF bandwidths of less than 1MHz below 50MHz FCentre exhibit significant distortion and their use is not recommended. This limitation will be addressed in planned future releases.
- #define TRFRefLevel "RefdBm"

  RW parameter - User-supplied expected max signal dBm.
- #define TRFUserCalibrationOffset "UserCaldB"

  RW parameter - User-supplied correction-factor for both Spectrum and calibrated-IQ.
- #define TRFFlattenFlag "flatten"

  RW parameter - apply spectrum equalization if available.
- #define TRFDurationFrames "captureFrames"

  RW parameter - Spectrum Stream specified capture duration.
- #define TRFDurationSec "captureSec"

  RW parameter - IQ Stream specified capture duration.
- #define TRFMaxContiguousSec "maxContiguousSec"

  RO parameter - IQ Stream max limit of continuous streaming. Note -only valid after attaching the stream to a receiver device.
- #define TRFIsContiguous "contiguous"

  RO parameter - Indicates this IQ stream is guaranteed contiguous.
- #define TRFFramesExpected "framesExpected"

  RO parameter - Number of frames expected in this stream, or 0 if configured for continuous capture.
- #define TRFSubOptimalFlag "SubOptimalDR"

  RO parameter - Dynamic range is non-optimal flag.
- #define TRFFullAdaptFlag "adaptFull"

  RW parameter - Full adapt flag - if set on stream will cause DR adaption before IQ streaming (~500msec)
- #define TRFStepAdaptFlag "adaptStep"

  RW parameter - Step adapt flag - if set on stream will cause DR adaption before IQ streaming.
- #define TRFMeasuredAvgdBm "avgdBm"

  RO parameter - Measured average power - only valid if TRFRefLevelAdapt is true.
- #define TRFdBFSD "dBFSD"

  RO parameter - Proportion of dynamic range utilized - expressed in dB relative to full-scale.
- #define TRFFlowControl "flowControl"

  RW parameter - If set, ensures tight flow control with source if it is a receiver for spectra and IQ data.
- #define TRFAttenuation "attenuation"

  RO parameter - Attenuation used in dB.
- #define TRFPacketDurationMsec "PacketDuration"

  RO parameter - Packet duration in secondds.
- #define TRFFilterRatio "IFFilterRatio"

  RO parameter - IF Filter ratio applied.

- #define TRFNTP "NTPServers"

  RW parameter - NTP servers list.
- #define TRFSCPIQueryTimeout "SCPIQueryTimeout"

  RW parameter - SCPI transaction timeout.
- #define TRFRxSampleRate "RxSampleRate"

  RO parameter - native 'base' sample rate of a receiver.
- #define TRFGNSSValid "GNSSValid"

  RO parameter - true if GNSS data is valid, false otherwise.
- #define TRFLatitude "latitude"

  RO parameter - Latitude returned by GNSS.
- #define TRFLongitude "longitude"

  RO parameter - Longitude returned by GNSS.
- #define TRFAltitude "altitude"

  RO parameter - Altitude returned by GNSS.
- #define TRFGNSSRxTime "GNSSPosnEpoch"

  RO parameter - Last position report in UTC Msec.
- #define TRFGNSSTimestamp "GNSSTimeNanosec"

  RO parameter - Most recent GNSS UTC timestamp in nanosec.
- #define TRFGNSSADelay "GNSSAntDelay"

  RW parameter - GNSS Antenna delay parameter.
- #define TRFGNSSConstellation "GNSSCons"

  RO parameter - GNSS Constellation information.
- #define TRFGNSSDynamicMode "GNSSDynamic"

  RW parameter - GNSS Dynamic mode (textual)
- #define TRFGNSSSpeedOverGround "GNSSSpeed"

  RO parameter - GNSS Speed over ground in m/sec.
- #define TRFGNSSHeadingDegT "GNSSHeading"

  RO parameter - GNSS Heading in degrees true.
- #define TRFGNSSTrackDegT "GNSSTrack"

  RO parameter - GNSS Track in degrees true.
- #define TRFGNSSMagVarDegT "GNSSMagVar"

  RO parameter - GNSS Magnetic variation in degrees E+,W-.
- #define TRFDeviceTemperature "deviceTemperature"

  RO parameter - Reported device internal temperatures.
- #define TRFDeviceLockStatus "deviceClockLock"

  RO parameter - Reported device clock lock statuses.
- #define TRFPLLSource "pllSource"

  RW parameter - PLL clock source (internal, external, gnss)
- #define TRFPPSSource "ppsSource"

  RW parameter - PPS clock source (external, gnss)
- #define TRFPPSSource "ppsSource"

  RW parameter - PPS clock source (external, gnss)
- #define TRFTimeSync "timeSync"

  RW parameter - clock reference source (disable,"ntp,once","ntp,continuous")
- #define TRFCurrentDate "Date"

  RW parameter - current date <year>,<month>,<date>
- #define TRFCurrentTime "Time"

RW parameter - current time <hour>,<minute>,<second>[,<millisecond>].

- #define TRFAMSubtype "type"

  RW parameter - type of AM demodulation to perform; [DSB, LSB, USB].
- #define TRFFramesProduced "frames"

  RO parameter - count of frames produced.
- #define TRFOutputSampleRate "OutputSampleRate"

  RW parameter - requested baseband output sample rate.
- #define TRFFollowIQ "FollowIQ"

  RW parameter - if 'true', then one spectrum frame will be produced for each input IQ frame with RBW dictated by the window function and input IQ frame size.
- #define TRFOverlap "FFTOverlap"

  RW parameter - sets the proportion 0.0-1.0 of overlap in successive output spectra.
- #define TRFMatchIQ "MatchIQ"

  RO parameter - Most recently processed input IQ frame sequence number.
- #define TRFOutputSize "OutputSize"

  RO parameter - Size of output spectra expected with current parameters.
- #define TRFRecentFrames "RecentFrames"

  RO parameter - Number of frames processed since last trfGetParameters call.
- #define TRFAverageCount "Average"

  RW parameter - number of spectra over which to create 'average' spectrum.
- #define TRFClearAverageTrace "ClearAverageTrace"

  RW parameter - On the next frame, clear the average trace.
- #define TRFClearMaxHoldTrace "ClearMaxHoldTrace"

  RW parameter - On the next frame, clear the max-hold trace.
- #define TRFClearMinHoldTrace "ClearMinHoldTrace"

  RW parameter - On the next frame, clear the min-hold trace.

Typedefs

- typedef unsigned short _uint16

  Basic types used in the API.
- typedef unsigned int _uint32

  Unsigned 32-bit integer.
- typedef unsigned int _uint

  Unsigned integer.
- typedef unsigned long long int _uint64

  Signed 64-bit integer.
- typedef float _float32

  32-bit floating point (IEEE-754) value
- typedef double _float64

  64-bit floating point (IEEE-754) value
- typedef _uint32 trfHandle

  Device or processor handle type.
- typedef enum _trfStatus trfStatus

  General operation status reporting.
- typedef struct _trfIQFrame trfIQFrame

  IQ data frame. This structure contains the actual data and all descriptive metadata.

- typedef struct _trfSpectrumFrame trfSpectrumFrame

  Spectrum data frame. This structure contains the actual spectral data and all descriptive metadata.

- typedef struct _trfBasebandFrame trfBasebandFrame

  Baseband sampled data (ie. audio) frame. Basedband data can be of any type representable by an array of _float32 values. Multiple channels are supported.

Enumerations

- enum _trfStatus {
  trfOk = 0 , trfWaiting = 1 , trfExhausted = 2 , trfNotStarted = 3 ,
  trfDiscontinuousWithPreviousFrame = 4 , trfDetached = 5 , trfUnimplemented = -1 , trfAPINotInitialized = -2 ,
  trfDeviceAddressAlreadyKnown = -3 , trfUnallocatedUserData = -4 , trfDeviceIndexOutOfRange = -5 ,
  trfNoDeviceInformation = -6 ,
  trfCannotOpenDevice = -7 , trfBadDeviceHandle = -8 , trfBadUnitHandle = -9 , trfBadParameterName = -10 ,
  trfNoParametersSpecified = -11 , tffBadParameter = -12 , trfInvalidParameter = -13 , trfBadReceiverHandle = -14 ,
  trfIQStreamStartFailure = -15 , trfMemoryAllocationError = -16 , trfInvalidStreamHandle = -17 , trfNoAssociatedRxOrFile = -18 ,
  trfNotAnIQStream = -19 , trfNoDataSpecified = -20 , trfInvalidHandleSpecified = -21 , trfBadStreamShutdown = -22 ,
  trfUnsupportedParameter = -23 , trfCannotCloseStream = -24 , trfNotASpectrumStream = -25 ,
  trfCannotRestartStream = -26 ,
  trfSweepStartFailure = -27 , trfFrameTypeMismatch = -28 , trfFileOpenFailed = -29 , trfFileTypeIsNotIQ = -30 ,
  trfFileTypeIsNotSpectrum = -31 , trfNoAssociatedSource = -32 , trfUnknownParameter = -33 , trfParameterSetError = -34 ,
  trfDeviceHandleIsInvalid = -35 , trfSCPISendFailed = -36 , trfSCPIQueryFailed = -37 , trfBadStreamHandle = -38 ,
  trfCannotAttachIQStream = -39 , trfCannotAttachSpectrumStream = -40 , trfCannotAttachStream = -41 ,
  trfCannotDetachIQStream = -42 ,
  trfCannotDetachSpectrumStream = -43 , trfCannotDetachStream = -44 , trfConnectionResetFailed = -45 ,
  trfBadFrequencyParameters = -46 ,
  trfBadRBWParameter = -47 , trfBadReferenceLevelParameter = -48 , trfSpectrumStreamCreateFailure = -49 ,
  trfBufferSecOutOfRange = -50 ,
  trfBufferFramesOutOfRange = -51 , trfBufferDiscardProportionOutOfRange = -52 , trfFrequencyRangeIsInvalid = -53 , trfBadGNSSDynamicMode = -54 ,
  trfBadPLLSource = -55 , trfPLLSourceWithNoGNSS = -56 , trfFCentreInvalid = -57 , trfIFBWInvalid = -58 ,
  trfFMinInvalid = -59 , trfFMaxInvalid = -60 , trfResolutionBWInvalid = -61 , trfFCentreOutOfRange = -62 ,
  trfIFBWOutOfRange = -63 , trfReferenceLevelOutOfRange = -64 , trfResolutionBWOutOfRange = -65 ,
  trfUnknownWindowType = -66 ,
  trfStartFrequencyOutOfRange = -67 , trfStopFrequencyOutOfRange = -68 , trfDeviceUnreachable = -69 ,
  trfInvalidFilename = -70 ,
  trfCannotOpenOutputFile = -71 , trfUninitializedUserData = -72 , trfBadProcessorHandle = -73 ,
  trfCannotCreateProcessor = -74 ,
  trfProcessorAttachFailed = -75 , trfProcessorDetachFailed = -76 , trfProcessorUnattached = -77 ,
  trfNotABasebandStream = -78 ,
  trfFileTypeIsNotBaseband = -79 , trfBasebandStreamStartFailure = -80 , trfSampleRateInvalid = -81 ,
  trfCannotObtainOutputStream = -82 ,
  trfCannotConnectOutputStream = -83 , trfBadFilename = -84 , trfWindowTypeInvalid = -85 , trfDeviceCommunicationsLost = -86 ,
  trfUnknownStreamType = -87 , trfStreamHandleAlreadyOpen = -88 , trfInvalidStreamSource = -89 ,
  trfBadPPSSource = -90 ,
  trfLastErrorSentinel = -1000 }

  General operation status reporting.

Functions

- _TRFAPI char * trfGetVersion (void)

  Return a C-string describing this version of the library.
- _TRFAPI const char * trfGetTextForStatus (trfStatus eStatus)

  Return textual (UTF-8, ISO-Latin-1) status description matching with the passed status code.
- _TRFAPI trfStatus trfGetNextStatus (void)

  Return any additional (descriptive) error codes arising from the last issued libtrf function on the calling thread. To return all error status a user can iterate on this call until it returns trfOk to indicate no further error information is available.
- _TRFAPI char * trfGetDetailedStatus (void)

  Return textual (UTF-8, ISO-Latin-1) error details for the last significant status on the calling thread. Clears error string for this thread on on call.
- _TRFAPI trfStatus trfAddDeviceAddress (const char *pRemoteAddress)

  Add a potential device address for use by trfEnumerateDevices. Note that local devices that may be discovered directly (LAN, USB3 etc...) will not require their addresses to be added through this call, however it is not an error if they are.
- _TRFAPI trfStatus trfEnumerateDevices (_uint *puDevices, bool bAutoDiscover)

  Enumerate all accessible devices and return a count. Device enumeration may be initiated by a user at any time to enumerate newly connected devices or update the list of devices.
- _TRFAPI trfStatus trfExamineDevice (_uint uDeviceIndex, char **ppJSON)

  Return known information about a discovered device (uDeviceIndex in the range [0 .. (uDevices-1)] into user-allocated trfDeviceInfo structure.
- _TRFAPI trfStatus trfOpenDevice (_uint uDeviceIndex, trfHandle *pDevice)

  *Given a device index, attempt to open it. On success sets the device handle *pHandle.*
- _TRFAPI trfStatus trfResetDeviceConnection (trfHandle cDevice)

  If a communications error is suspected, issuing this call will cause a connection shutdown and re-open to the device without loosing any operational state. On success the device will continue with programmed operation. If the return value is not 'trfOk' then it is likely communications are unrecoverable and the device handle should be closed (see trfClose).
- _TRFAPI trfStatus trfClose (trfHandle *pHandle)

  *Close any open handle (device, processor, stream) On successful return, *pHandle==kInvalidHandle.*
- _TRFAPI trfStatus trfCreateIQStream (trfHandle *pStream, _float32 fCentreHz, _float32 fIFBWHz, _float32 fRefLeveldBm)

  Create an IQ stream with the given parameters. This entity will be inactive until successfully attached to a device capable of servicing the stream. Note that in the current implementation of libtrf, IF bandwidths narrower than 100kHz are not supported. Additionally IF bandwidths of less than 1MHz below 50MHz FCentre exhibit significant distortion and their use is not recommended. This limitation will be addressed in planned future releases.
- _TRFAPI trfStatus trfGetNextIQ (trfHandle cStream, trfIQFrame **ppFrame, _uint32 uTimeoutMsec)

  *Obtain the next IQ frame (if available) from the stream. If no frame is yet available trfStatus will be 'trfWaiting'. If the stream has been halted then the status 'trfExhausted' may be returned to indicate there are no more buffered frames to retrieve. If the retrieval succeeds, trfStatus will be trfOk and *ppFrame will point to the returned IQ frame. The frame should be disposed of using trfFreeIQ. The trfExhausted return indicates that capture for this 'attach' is complete and the stream has been detached.*
- _TRFAPI trfStatus trfFreeIQ (trfIQFrame **ppFrame)

  Free the given IQ frame, which will no longer be valid after this call (NULL). The user should ensure that frames are disposed when no longer required to avoid STORAGE_CAPACITY_VIOLATION errors.
- _TRFAPI trfStatus trfCreateSpectrumStream (trfHandle *pStream, _float32 fMinHz, _float32 fMaxHz, _float32 fRBWHz, const char *pWindowType, _float32 fRefLeveldBm)

  Create a spectrum stream with the given parameters. This entity will be inactive until successfully attached to a device capable of servicing the stream.
- _TRFAPI trfStatus trfGetNextSpectrum (trfHandle cStream, trfSpectrumFrame **ppFrame, _uint32 uTimeout↩ Msec)

Obtain the next frame (if available) from the stream. If no frame is yet available trfStatus will be 'WAITING' and ∗pp↩ Frame will be NULL otherwise trfStatus will be trfOk and ∗ppFrame will point to the returned data frame. The frame should be disposed of using trfFreeSpectrum.

- _TRFAPI trfStatus trfFreeSpectrum (trfSpectrumFrame ∗∗ppFrame)

  Free the given spectrum frame, which will no longer be valid after this call. The user should ensure that frames are disposed when no longer required to avoid STORAGE_CAPACITY_VIOLATION errors.

- _TRFAPI trfStatus trfGetNextBaseband (trfHandle cStream, trfBasebandFrame ∗∗ppFrame, _uint32 u↩ TimeoutMsec)

  Obtain the next Baseband frame (if available) from the stream. If no frame is yet available trfStatus will be 'trfWaiting'. If the stream has been halted then the status 'trfExhausted' may be returned to indicate there are no more buffered frames to retrieve. If the retrieval succeeds, trfStatus will be trfOk and ∗ppFrame will point to the returned Baseband frame. The frame should be disposed of using trfFreeBaseband. The trfExhausted return indicates that capture for this 'attach' is complete and the stream has been detached.

- _TRFAPI trfStatus trfFreeBaseband (trfBasebandFrame ∗∗ppFrame)

  Free the given IQ frame, which will no longer be valid after this call (NULL). The user should ensure that frames are disposed when no longer required to avoid STORAGE_CAPACITY_VIOLATION errors.

- _TRFAPI trfStatus trfAttachStreamToDevice (trfHandle cDevice, trfHandle cStream)

  Attach the given stream to the specified device. A stream may only be attached to one device at a time. If the device supports the stream, the call will return trfOk and the stream will start to produce data, subject to resource availability on the specified device.

- _TRFAPI trfStatus trfDetachStream (trfHandle cStream)

  Detach the given stream from any device to which it is connected. This will stop any capture associated with the stream - however the stream will still contain any buffered data that has been previously delivered.

- _TRFAPI trfStatus trfFlushStream (trfHandle cStream)

  Remove all frames currently present in a stream. If the stream is attached and delivering data, this will potentially create a break in the data. In the case that the stream is detached, this will remove any outstanding frames. After this call any call to trfGetNextXXX will return trfExhausted.

- _TRFAPI trfStatus trfIsAttached (trfHandle cStream)

  Test if the specified stream is attached. Note that a finite capture stream will detach itself after capture is completed. This change will occur at some point in time after the completion of capture, and hence it is valid for this function to return trfOk whilst an associated trfGetNextXXXFrame has returned trfExhausted (finite capture).

- _TRFAPI trfStatus trfAllowDiscarding (trfHandle cStream, bool bAllowDiscarding)

  By default, the data contained in streams will be discarded to ensure buffer size constraints are respected. In certain cases - such as streaming from file - it is necessary to control the flow of data to keep a processing chain synchronized and not arbitrarily drop frames. This call allows streams that would normally contain non-discardable frames to allow discarding to occur. By setting this flag on (say) an IQ stream from a file that is being processed downstream, the user will not have to loop to keep this stream emptied to allow the processor to proceed - otherwise the stream would block waiting for the stream buffer to be emptied.

- _TRFAPI trfStatus trfSetStreamOutputFile (trfHandle cStream, const char ∗pFilePath)

  Attaches the specified output file - or if the file exists and is not empty, a numbered sibling file to the specified stream. The self-contained output file may be replayed using the trfCreateFileSourceStream function. The output file shall have the extension '.data' (ThinkRF Spectrum, IQ, Baseband), The binary file shall each be accompanied by a metadata file (.json). If pFilePath is NULL, any file writing for the stream shall be immediately halted.

- _TRFAPI trfStatus trfAttachStreamToFile (trfHandle cStream, const char ∗pFilePath)

  Attaches the given stream to an input file. Assumes that the file is valid and of the correct type for the stream - otherwise the call will fail. Playback from the stream is controllable through the following parameters that may be set for the stream: TRFLooping - boolean: if set true then the file will be played continuously by looping back to the start once the file has been completely delivered. Function parameters:

- _TRFAPI trfStatus trfConvertBasebandFileToWavFile (const char ∗pSourceFilePath, const char ∗p↩ DestinationFilePath)

  Stand-alone function to convert a previously created Baseband stream file and produce a WAV file equivalent output file.

- _TRFAPI trfStatus trfGetProcessorTypes (char ∗∗ppJSON)

Returns a JSON-formatted array names 'processortypes' of the available processor types.

- _TRFAPI trfStatus trfCreateProcessor (trfHandle *pProcessor, const char *pType)

  Create a processor entity of the given type. Returns a handle to the newly created processor.
- _TRFAPI trfStatus trfAttachProcessorToStream (trfHandle hProcessor, trfHandle hSourceStream)

  Attach a processor to a source stream.
- _TRFAPI trfStatus trfDetachProcessor (trfHandle hProcessor)

  Detach a processor from its current source stream.
- _TRFAPI trfStatus trfGetProcessorOutputStream (trfHandle cProcessor, trfHandle *pStream)

  Return the output stream from the given processor.
- _TRFAPI trfStatus trfGetParameterInfo (trfHandle cEntity, char **ppJSON)

  Obtains all available parameters for the specified device along with the valid ranges, type and any associated information through ppJSON.
- _TRFAPI trfStatus trfReadParameterInfoElement (char **ppJSON, const char *pParameter, const char *p↩ Element, _float32 *pfValue)

  Read a sub-element from a JSON element as a _float32 value. Useful for obtaining 'min', 'max' and 'default' values from parameter information.
- _TRFAPI trfStatus trfReadParameterAsJSON (char **pJSON, const char *pParameter, char **pp↩ JSONExtract)

  Read a sub-element from a JSON element as another JSON string. Useful for extracting substructure from a JSON string.
- _TRFAPI trfStatus trfGetParameters (trfHandle cEntity, char **ppJSON)

  Obtain the current values of device/processor parameters and make accessible through ppJSON.
- _TRFAPI trfStatus trfSetParameters (trfHandle cEntity, char **ppJSON)

  Set the current value of a device/stream/processor parameter.
- _TRFAPI void trfDisposeString (char **ppString)

  *Dispose of a previously allocated ppJSON string. Note that only strings that are returned via a parameter of the form 'char **' should be disposed of using this call. This is provided for syntactic convenience and is equivalent to: if (ppJSON != nullptr) { if (*ppJSON != nullptr) { delete [](*ppJSON); *ppJSON = nullptr; } }.*
- _TRFAPI trfStatus trfAddNumericParameter (char **ppJSON, const char *pParameterName, _float32 fValue)

  For parameters not named above - add parameter by name Add named numeric parameter to ppJSON.
- _TRFAPI trfStatus trfAddTextParameter (char **ppJSON, const char *pParameterName, const char *pValue)

  Add named textual parameter to ppJSON.
- _TRFAPI trfStatus trfAddBooleanParameter (char **ppJSON, const char *pParameterName, bool bValue)

  Add named boolean parameter to ppJSON.
- _TRFAPI trfStatus trfReadNumericParameter (char **ppJSON, const char *pParameterName, _float32 *pf↩ Value)

  Read named numeric parameter from ppJSON.
- _TRFAPI trfStatus trfReadTextParameter (char **ppJSON, const char *pParameterName, char **ppValue)

  Read named textual parameter from ppJSON.
- _TRFAPI trfStatus trfReadBooleanParameter (char **ppJSON, const char *pParameterName, bool *pbValue)

  Read named boolean parameter from ppJSON.
- _TRFAPI trfStatus trfAddFCentre (char **ppJSON, _float32 fFCentreHz)

  Add Centre frequency parameter to ppJSON. Centre frequency parameter is required by IQ Streams.
- _TRFAPI trfStatus trfAddIFBWHz (char **ppJSON, _float32 fFIFBWHz)

  Add IF Bandwidth (IFBW) parameter to ppJSON. IFBW is required by IQ Streams.
- _TRFAPI trfStatus trfAddFMinMax (char **ppJSON, _float32 fFMinHz, _float32 fFMaxHz)

  Add min/max frequency range to ppJSON. The min and max frequencies specified are required by Spectrum streams.
- _TRFAPI trfStatus trfAddRBWHz (char **ppJSON, _float32 fRBWHz)

Add resolution bandwidth parameter to ppJSON. RBW parameter is required by spectrum streams.

- _TRFAPI trfStatus trfAddWindowType (char **ppJSON, const char *pWindow)

  Add window type specification to ppJSON. A window type is optional for spectrum streams.

- _TRFAPI trfStatus trfAddRefLeveldBm (char **ppJSON, _float32 fRefLeveldBm)

  Add reference level to ppJSON The reference level is the strongest expected signal. Setting the reference level instructs a receiver to optimize its signal path, attenuation, gain settings to maximize available dynamic range for this expected signal level. A reference level is optional but recommended for IQ and Spectrum streams.

- _TRFAPI trfStatus trfAddIQCaptureSec (char **ppJSON, _float32 fIQCaptureSec)

  Add capture duration parameter ppJSON The duration is expressed in seconds and for an IQ stream represents the amount of data in seconds to be captured. After this capture period has expired from attaching to a device, the stream will automatically detach from the device and report 'trfExhausted' once all captured data has been retrieved. All devices will attempt to meet fIQCaptureSec at best-effort. In the case that the captured time will fit in internal buffering, the captured IQ sequence is guaranteed to be contiguous. If a stream duration is short enough to be fully buffered, the read-only stream parameter 'contiguous' will be present (and reads as 'true' (boolean)). A stream for which the 'contiguous' flag is not present may contain discontinuities. The read-only parameter 'maxContiguousSec' (_float32) gives the maximum capture length that can fit in device buffering and thus for which the 'contiguous' flag will be set with the current capture parameters.

- _TRFAPI trfStatus trfReadFCentre (char **ppJSON, _float32 *pfFCentreHz)

  Read Centre frequency parameter from ppJSON.

- _TRFAPI trfStatus trfReadIFBWHz (char **ppJSON, _float32 *pfFIFBWHz)

  Read IFBW parameter from ppJSON.

- _TRFAPI trfStatus trfReadFMinMax (char **ppJSON, _float32 *pfFMinHz, _float32 *pfFMaxHz)

  Read min/max frequency range from ppJSON.

- _TRFAPI trfStatus trfReadRBWHz (char **ppJSON, _float32 *pfRBWHz)

  Read resolution bandwidth parameter from ppJSON.

- _TRFAPI trfStatus trfReadWindowType (char **ppJSON, char **ppWindow)

  Read window type parameter from ppJSON.

- _TRFAPI trfStatus trfReadSampleRateHz (char **ppJSON, _float32 *pfSampleRateHz)

  Read sample rate parameter from ppJSON.

- _TRFAPI trfStatus trfReadRefLeveldBm (char **ppJSON, _float32 *pfRefLeveldBm)

  Read reference level parameter from ppJSON.

6.2.1 Macro Definition Documentation

TRFAddress

```
#define TRFAddress "address"
```

RO network address tag.

TRFAltitude

```
#define TRFAltitude "altitude"
```

RO parameter - Altitude returned by GNSS.

TRFAMSubtype

```
#define TRFAMSubtype "type"
```

RW parameter - type of AM demodulation to perform; [DSB, LSB, USB].

TRFAttenuation

```
#define TRFAttenuation "attenuation"
```

RO parameter - Attenuation used in dB.

TRFAverageCount

```
#define TRFAverageCount "Average"
```

RW parameter - number of spectra over which to create 'average' spectrum.

TRFBandwidthHz

```
#define TRFBandwidthHz "BandwidthHz"
```

RO generic bandwidth parameter.

TRFBufferFrames

```
#define TRFBufferFrames "bufferFrames"
```

RW parameter - user-settable maximum stream buffering in frames. If set this will override TRFBufferSec.

TRFBufferSec

```
#define TRFBufferSec "bufferSec"
```

RW parameter - user-settable maximum buffering duration in seconds. If set this will override TRFBufferFrames.

TRFClearAverageTrace

```
#define TRFClearAverageTrace "ClearAverageTrace"
```

RW parameter - On the next frame, clear the average trace.

TRFClearMaxHoldTrace

```
#define TRFClearMaxHoldTrace "ClearMaxHoldTrace"
```

RW parameter - On the next frame, clear the max-hold trace.

TRFClearMinHoldTrace

```
#define TRFClearMinHoldTrace "ClearMinHoldTrace"
```

RW parameter - On the next frame, clear the min-hold trace.

TRFCurrentDate

```
#define TRFCurrentDate "Date"
```

RW parameter - current date <year>,<month>,<date>

TRFCurrentTime

```
#define TRFCurrentTime "Time"
```

RW parameter - current time <hour>,<minute>,<second>[,<millisecond>].

TRFdBFSD

```
#define TRFdBFSD "dBFSD"
```

RO parameter - Proportion of dynamic range utilized - expressed in dB relative to full-scale.

TRFDefault

```
#define TRFDefault "default"
```

Generic JSON parameter names for libtrf entities; devices, streams, processors. These parameter names are used in conjunction with JSON string construction/ examination functions to allow parameters to be set for and obtained from libtrf entities. In the following documentation 'RO' denotes 'read-only', and RW denotes read/write. Setting a parameter for an entity that does not support that parameter is an error and will typically return trfParameterSetError when calling trfSetParameters with a JSON string including an unsupported parameter. Detailed information can be obtained using trfGetDetailedStatus. Note that these names may change across future releases and therefore it is highly recommended that the following symbolic names TRF___ are used instead of their textual equivalent. RO - Read-only, RW - Read and Write.

RO default parameter value tag

TRFDevice

```
#define TRFDevice "device"
```

RO device information section.

TRFDeviceConnection

```
#define TRFDeviceConnection "connection"
```

RO parameter - device textual connection information.

TRFDeviceFirmware

```
#define TRFDeviceFirmware "firmware"
```

RO parameter - device textual firmware version.

TRFDeviceLockStatus

```
#define TRFDeviceLockStatus "deviceClockLock"
```

RO parameter - Reported device clock lock statuses.

TRFDeviceSerialNum

```
#define TRFDeviceSerialNum "serial"
```

RO parameter - device textual serial number.

TRFDeviceTemperature

```
#define TRFDeviceTemperature "deviceTemperature"
```

RO parameter - Reported device internal temperatures.

TRFDeviceType

```
#define TRFDeviceType "type"
```

RO parameter - device textual type.

TRFDeviceVersion

```
#define TRFDeviceVersion "version"
```

RO parameter - device textual version.

TRFDiscardProportion

```
#define TRFDiscardProportion "bufferDiscard"
```

RW parameter - On overflow (TRFBufferSec, TRFBufferFrames) this sets the proportion of the buffer that will be discarded. The oldest frames will be discarded first. Setting to zero will minimize the number of frames discarded. Setting to 1.0 will discard the whole buffer. All other values 0,1 are acceptable. This allows a user to manage the tradeoff between discontinuities and data latency.

TRFDurationFrames

```
#define TRFDurationFrames "captureFrames"
```

RW parameter - Spectrum Stream specified capture duration.

TRFDurationSec

```
#define TRFDurationSec "captureSec"
```

RW parameter - IQ Stream specified capture duration.

TRFFCentreHz

```
#define TRFFCentreHz "FCentreHz"
```

RW parameter - IQ Capture centre frequency.

TRFFilename

```
#define TRFFilename "filename"
```

RW parameter - filename parameter (stream)

TRFFilterRatio

```
#define TRFFilterRatio "IFFilterRatio"
```

RO parameter - IF Filter ratio applied.

TRFFlattenFlag

```
#define TRFFlattenFlag "flatten"
```

RW parameter - apply spectrum equalization if available.

TRFFlowControl

```
#define TRFFlowControl "flowControl"
```

RW parameter - If set, ensures tight flow control with source if it is a receiver for spectra and IQ data.

TRFFMaxHz

```
#define TRFFMaxHz "FMaxHz"
```

RW parameter - Max extent of sweep range.

TRFFMinHz

```
#define TRFFMinHz "FMinHz"
```

RW parameter - Min extent of sweep range.

TRFFollowIQ

```
#define TRFFollowIQ "FollowIQ"
```

RW parameter - if 'true', then one spectrum frame will be produced for each input IQ frame with RBW dictated by the window function and input IQ frame size.

TRFFramesExpected

```
#define TRFFramesExpected "framesExpected"
```

RO parameter - Number of frames expected in this stream, or 0 if configured for continuous capture.

TRFFramesProduced

```
#define TRFFramesProduced "frames"
```

RO parameter - count of frames produced.

TRFFullAdaptFlag

```
#define TRFFullAdaptFlag "adaptFull"
```

RW parameter - Full adapt flag - if set on stream will cause DR adaption before IQ streaming (∼500msec)

TRFGNSSADelay

```
#define TRFGNSSADelay "GNSSAntDelay"
```

RW parameter - GNSS Antenna delay parameter.

TRFGNSSConstellation

```
#define TRFGNSSConstellation "GNSSCons"
```

RO parameter - GNSS Constellation information.

TRFGNSSDynamicMode

```
#define TRFGNSSDynamicMode "GNSSDynamic"
```

RW parameter - GNSS Dynamic mode (textual)

TRFGNSSHeadingDegT

```
#define TRFGNSSHeadingDegT "GNSSHeading"
```

RO parameter - GNSS Heading in degrees true.

TRFGNSSMagVarDegT

```
#define TRFGNSSMagVarDegT "GNSSMagVar"
```

RO parameter - GNSS Magnetic variation in degrees E+,W-.

TRFGNSSRxTime

```
#define TRFGNSSRxTime "GNSSPosnEpoch"
```

RO parameter - Last position report in UTC Msec.

TRFGNSSSpeedOverGround

```
#define TRFGNSSSpeedOverGround "GNSSSpeed"
```

RO parameter - GNSS Speed over ground in m/sec.

TRFGNSSTimestamp

```
#define TRFGNSSTimestamp "GNSSTimeNanosec"
```

RO parameter - Most recent GNSS UTC timestamp in nanosec.

TRFGNSSTrackDegT

```
#define TRFGNSSTrackDegT "GNSSTrack"
```

RO parameter - GNSS Track in degrees true.

TRFGNSSValid

```
#define TRFGNSSValid "GNSSValid"
```

RO parameter - true if GNSS data is valid, false otherwise.

TRFHasFlatteningInfo

```
#define TRFHasFlatteningInfo "iqequalization"
```

RO receiver flag that equalization data is available (flattening is possible)

TRFIFBWHz

```
#define TRFIFBWHz "IFBWHz"
```

RW parameter - IF Bandwidth for IQ capture. Note that in the current implementation of libtrf, IF bandwidths narrower than 100kHz are not supported. Additionally IF bandwidths of less than 1MHz below 50MHz FCentre exhibit significant distortion and their use is not recommended. This limitation will be addressed in planned future releases.

trfInvalidHandle

```
#define trfInvalidHandle (0xffffffff)
```

Invalid handle value for any libtrf entity. As good practice, all unused or uninitialized handle values should as good practice be set to this value - ie.

```
trfHandle hNewDevice(trfInvalidHandle);
```

TRFIQActive

```
#define TRFIQActive "IQActive"
```

RO parameter - IQ capture is active flag (Rx)

TRFIsContiguous

```
#define TRFIsContiguous "contiguous"
```

RO parameter - Indicates this IQ stream is guaranteed contiguous.

TRFLatitude

```
#define TRFLatitude "latitude"
```

RO parameter - Latitude returned by GNSS.

TRFLongitude

```
#define TRFLongitude "longitude"
```

RO parameter - Longitude returned by GNSS.

TRFLooping

```
#define TRFLooping "LoopingFlag"
```

RW set to cause file to loop.

TRFMatchIQ

```
#define TRFMatchIQ "MatchIQ"
```

RO parameter - Most recently processed input IQ frame sequence number.

TRFMax

```
#define TRFMax "max"
```

RO maximum value tag.

TRFMaxContiguousSec

```
#define TRFMaxContiguousSec "maxContiguousSec"
```

RO parameter - IQ Stream max limit of continuous streaming. Note -only valid after attaching the stream to a receiver device.

TRFMeasuredAvgdBm

```
#define TRFMeasuredAvgdBm "avgdBm"
```

RO parameter - Measured average power - only valid if TRFRefLevelAdapt is true.

TRFMin

```
#define TRFMin "min"
```

RO minimum value tag.

TRFNTP

```
#define TRFNTP "NTPServers"
```

RW parameter - NTP servers list.

TRFOutputSampleRate

```
#define TRFOutputSampleRate "OutputSampleRate"
```

RW parameter - requested baseband output sample rate.

TRFOutputSize

```
#define TRFOutputSize "OutputSize"
```

RO parameter - Size of output spectra expected with current parameters.

TRFOverlap

```
#define TRFOverlap "FFTOverlap"
```

RW parameter - sets the proportion 0.0-1.0 of overlap in successive output spectra.

TRFPacketDurationMsec

```
#define TRFPacketDurationMsec "PacketDuration"
```

RO parameter - Packet duration in secondds.

TRFPLLSource

```
#define TRFPLLSource "pllSource"
```

RW parameter - PLL clock source (internal, external, gnss)

TRFPPSSource [1/2]

```
#define TRFPPSSource "ppsSource"
```

RW parameter - PPS clock source (external, gnss)

TRFPPSSource [2/2]

```
#define TRFPPSSource "ppsSource"
```

RW parameter - PPS clock source (external, gnss)

TRFRBWHz

```
#define TRFRBWHz "RBWHz"
```

RW parameter - Resolution Bandwidth in Hz.

TRFRecentFrames

```
#define TRFRecentFrames "RecentFrames"
```

RO parameter - Number of frames processed since last trfGetParameters call.

TRFRefLevel

```
#define TRFRefLevel "RefdBm"
```

RW parameter - User-supplied expected max signal dBm.

TRFRxSampleRate

```
#define TRFRxSampleRate "RxSampleRate"
```

RO parameter - native 'base' sample rate of a receiver.

TRFSampleRateHz

```
#define TRFSampleRateHz "SampleRateHz"
```

RO parameter - IQ Capture sample rate.

TRFSCPIQueryTimeout

```
#define TRFSCPIQueryTimeout "SCPIQueryTimeout"
```

RW parameter - SCPI transaction timeout.

TRFSequence

```
#define TRFSequence "Sequence"
```

RO sequence number parameter (internal)

TRFStepAdaptFlag

```
#define TRFStepAdaptFlag "adaptStep"
```

RW parameter - Step adapt flag - if set on stream will cause DR adaption before IQ streaming.

TRFSubOptimalFlag

```
#define TRFSubOptimalFlag "SubOptimalDR"
```

RO parameter - Dynamic range is non-optimal flag.

TRFSweepActive

```
#define TRFSweepActive "SweepActive"
```

RO parameter - sweep is active flag (Rx)

TRFTimeResolution

```
#define TRFTimeResolution "TimeResolution"
```

RO time resolution parameter (internal)

TRFTimestamp

```
#define TRFTimestamp "Timestamp"
```

RO timestamp parameter (internal)

TRFTimeSync

```
#define TRFTimeSync "timeSync"
```

RW parameter - clock reference source (disable,"ntp,once","ntp,continuous")

TRFType

```
#define TRFType "type"
```

RO type of unit tag (user readable)

TRFUnits

```
#define TRFUnits "units"
```

RO units of value tag.

TRFUserCalibrationOffset

```
#define TRFUserCalibrationOffset "UserCaldB"
```

RW parameter - User-supplied correction-factor for both Spectrum and calibrated-IQ.

TRFWindow

```
#define TRFWindow "WindowFn"
```

RW parameter - Window function for spectrum reconstruction.

6.2.2 Typedef Documentation

_float32

```
typedef float _float32
```

32-bit floating point (IEEE-754) value

_float64

```
typedef double _float64
```

64-bit floating point (IEEE-754) value

_uint

```
typedef unsigned int _uint
```

Unsigned integer.

_uint16

```
typedef unsigned short _uint16
```

Basic types used in the API.

Unsigned 16-bit integer

_uint32

```
typedef unsigned int _uint32
```

Unsigned 32-bit integer.

_uint64

```
typedef unsigned long long int _uint64
```

Signed 64-bit integer.

trfBasebandFrame

```
typedef struct _trfBasebandFrame trfBasebandFrame
```

Baseband sampled data (ie. audio) frame. Basedband data can be of any type representable by an array of _float32 values. Multiple channels are supported.

trfHandle

```
typedef _uint32 trfHandle
```

Device or processor handle type.

Handle for accessing libtrf resource (device, processor etc...)

trfIQFrame

```
typedef struct _trfIQFrame trfIQFrame
```

IQ data frame. This structure contains the actual data and all descriptive metadata.

trfSpectrumFrame

```
typedef struct _trfSpectrumFrame trfSpectrumFrame
```

Spectrum data frame. This structure contains the actual spectral data and all descriptive metadata.

trfStatus

```
typedef enum _trfStatus trfStatus
```

General operation status reporting.

6.2.3 Enumeration Type Documentation

_trfStatus

enum _trfStatus

General operation status reporting.

Enumerator

trfOk	No errors, no warnings - operation has completed successfully.
trfWaiting	Warnings or status (greater than 0) No error - data delivery is underway but momentarily unavailable
trfExhausted	No more data is available.
trfNotStarted	Data is expected but is not yet available.
trfDiscontinuousWithPreviousFrame	Indicator that a discontinuity is detected in IQ or baseband data.
trfDetached	Indicates that a stream is in a 'detached' state.
trfUnimplemented	Errors codes (less than 0) Function or behaviour is not currently available
trfAPINotInitialized	API initialization is required before executing this function.
trfDeviceAddressAlreadyKnown	An already-known device address has been added.
trfUnallocatedUserData	Expected user-allocated storage has not been provided.
trfDeviceIndexOutOfRange	Provided device index is not in the range of enumerated devices.
trfNoDeviceInformation	No information is available for the specified device.
trfCannotOpenDevice	Unable to open the specified device.
trfBadDeviceHandle	The given device handle is invalid or corrupted.
trfBadUnitHandle	The given entity handle is invalid or corrupted.
trfBadParameterName	A parameter name specified is incorrect.
trfNoParametersSpecified	No parameters where parameters were expected have been specified.
tffBadParameter	A parameter has been specified which is out of range.
trfInvalidParameter	An invalid parameter has been specified.
trfBadReceiverHandle	The given receiver handle is invalid or corrupted.
trfIQStreamStartFailure	The IQ stream has failed to start.
trfMemoryAllocationError	An internal memory allocation has failed.
trfInvalidStreamHandle	The given stream handle is invalid or corrupted.
trfNoAssociatedRxOrFile	The specified stream has no associated data source.
trfNotAnIQStream	An IQ-specific call is made on a non IQ-stream entity handle.
trfNoDataSpecified	No data has been specified where data was expected.
trfInvalidHandleSpecified	The given entity handle is invalid or corrupted.
trfBadStreamShutdown	An error was encountered during stream shutdown.
trfUnsupportedParameter	A parameter was specified which is not supported.
trfCannotCloseStream	An error occurred preventing the specified stream being closed.
trfNotASpectrumStream	A stream handle for a non-spectrum stream was specified.
trfCannotRestartStream	A failure has occurred preventing the stream from restarting.
trfSweepStartFailure	A failure has occurred preventing the spectrum stream from starting.
trfFrameTypeMismatch	An incorrect frame type has been specified.
trfFileOpenFailed	A failure has occurred when attempting to open a file.
trfFileTypeIsNotIQ	The file specified was expected to contain IQ data.
trfFileTypeIsNotSpectrum	The file specified was expected to contain spectra.

Enumerator

trfNoAssociatedSource	No source is associated with this entity.
trfUnknownParameter	The parameter specified is not known.
trfParameterSetError	An error was encountered when setting a parameter.
trfDeviceHandleIsInvalid	An invalid or corrupted device handle was specified.
trfSCPISendFailed	An error in sending SCPI data has occurred.
trfSCPIQueryFailed	An error in making a SCPI query has occurred.
trfBadStreamHandle	An invalid or corrupted stream handle was specified.
trfCannotAttachIQStream	A failure occurred when attempting to attach an IQ stream.
trfCannotAttachSpectrumStream	A failure occurred when attempting to attach a spectrum stream.
trfCannotAttachStream	A failure occurred when attempting to attach a stream.
trfCannotDetachIQStream	A failure occurred when attempting to detach an IQ stream.
trfCannotDetachSpectrumStream	A failure occurred when attempting to detach a spectrum stream.
trfCannotDetachStream	A failure occurred when attempting to detach a stream.
trfConnectionResetFailed	Failed to reset the connection to the specified device.
trfBadFrequencyParameters	'Impossible' frequency parameters specified (ie. FStart >= FEnd)
trfBadRBWParameter	'Impossible' RBW parameter specified
trfBadReferenceLevelParameter	'Impossible' reference level parameter specified
trfSpectrumStreamCreateFailure	General failure to create a spectrum stream.
trfBufferSecOutOfRange	Buffer seconds parameter specified is invalid.
trfBufferFramesOutOfRange	Buffer frames parameter specified is invalid.
trfBufferDiscardProportionOutOfRange	Buffer discard propotion parameter specified is out of range (valid is 0.0 to 1.0)
trfFrequencyRangeIsInvalid	Invalid frequency range specified.
trfBadGNSSDynamicMode	Invalid GNSS dynamic mode specified.
trfBadPLLSource	Invalid PLL Source parameter specified.
trfPLLSourceWithNoGNSS	PLL source specified for a non-GNSS enabled device.
trfFCentreInvalid	Centre frequency specified is invalid.
trfIFBWInvalid	IFBW specified is invalid.
trfFMinInvalid	Minimum frequency specified is invalid.
trfFMaxInvalid	Maximum frequency specified is invalid.
trfResolutionBWInvalid	Resolution bandwidth specified is invalid.
trfFCentreOutOfRange	Centre frequency specified is out of range for this device.
trfIFBWOutOfRange	IFBW specified is out of range for this device.
trfReferenceLevelOutOfRange	Reference level is out-of-range for this device.
trfResolutionBWOutOfRange	Resolution bandwidth is out-of-range for this device.
trfUnknownWindowType	Window type specified is not supported by this device.
trfStartFrequencyOutOfRange	Minimum frequency specified is not supported by this device.
trfStopFrequencyOutOfRange	Maximum frequency specified is not supported by this device.
trfDeviceUnreachable	Unable to connect with device through network address.
trfInvalidFilename	Filename passed does not specify a valid file path.
trfCannotOpenOutputFile	Failed to open specified output file for writing.
trfUninitializedUserData	Expected data was found uninitialized.

Enumerator

trfBadProcessorHandle	Unknown processor handle specified.
trfCannotCreateProcessor	Failed to create processor.
trfProcessorAttachFailed	Attempt to attach processor to stream failed (wrong stream type?)
trfProcessorDetachFailed	Attempt to detach processor from source stream failed.
trfProcessorUnattached	Attempt to detach processor when not attached.
trfNotABasebandStream	Stream handle given is not of a baseband stream.
trfFileTypeIsNotBaseband	Stream file given is not a baseband file.
trfBasebandStreamStartFailure	Failed to start baseband file strema from file.
trfSampleRateInvalid	Invalid sample rate specified.
trfCannotObtainOutputStream	Unable to obtain processor output stream handle.
trfCannotConnectOutputStream	Failed to connect output stream to processor.
trfBadFilename	Filename / file path is invalid.
trfWindowTypeInvalid	Window type specified is not known.
trfDeviceCommunicationsLost	Device is not responding over network.
trfUnknownStreamType	Stream type is not known - internal error (should never happen)
trfStreamHandleAlreadyOpen	Stream handle is already open - attempt to re-open failed.
trfInvalidStreamSource	Stream source is invalid inside a connection operation.
trfBadPPSSource	Invalid PPS Source parameter specified.
trfLastErrorSentinel	

6.2.4 Function Documentation

trfAddBooleanParameter()

```
_TRFAPI trfStatus trfAddBooleanParameter (
            char ** ppJSON,
            const char * pParameterName,
            bool bValue )
```

Add named boolean parameter to ppJSON.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
pParameterName	Name of parameter to add.
bValue	Boolean value to add.

Returns

error/warning information (on success == trfOk)

trfAddDeviceAddress()

```
_TRFAPI trfStatus trfAddDeviceAddress (
            const char * pRemoteAddress )
```

Add a potential device address for use by trfEnumerateDevices. Note that local devices that may be discovered directly (LAN, USB3 etc...) will not require their addresses to be added through this call, however it is not an error if they are.

Parameters

pRemoteAddress	IPv4/IPv6 address for remote device.

Returns

error/warning information (on success == trfOk)

Example:

```
eStatus = trfAddDeviceAddress("10.126.110.114");
if (eStatus != trfOk) {
    cout « "trfAddDeviceAddress failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
```

trfAddFCentre()

```
_TRFAPI trfStatus trfAddFCentre (
            char ** ppJSON,
            _float32 fFCentreHz )
```

Add Centre frequency parameter to ppJSON. Centre frequency parameter is required by IQ Streams.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
fFCentreHz	Centre frequency to add to JSON parameter string.

Returns

error/warning information (on success == trfOk)

trfAddFMinMax()

```
_TRFAPI trfStatus trfAddFMinMax (
            char ** ppJSON,
            _float32 fFMinHz,
            _float32 fFMaxHz )
```

Add min/max frequency range to ppJSON. The min and max frequencies specified are required by Spectrum streams.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
fFMinHz	Minimum extent frequency to add to JSON parameter string.
fFMaxHz	Maximum extent frequency to add to JSON parameter string.

Returns

error/warning information (on success == trfOk)

trfAddIFBWHz()

```
_TRFAPI trfStatus trfAddIFBWHz (
            char ** ppJSON,
            _float32 fFIFBWHz )
```

Add IF Bandwidth (IFBW) parameter to ppJSON. IFBW is required by IQ Streams.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
fFIFBWHz	IF Bandwidth parameter to add to JSON parameter string.

Returns

error/warning information (on success == trfOk)

trfAddIQCaptureSec()

```
_TRFAPI trfStatus trfAddIQCaptureSec (
            char ** ppJSON,
            _float32 fIQCaptureSec )
```

Add capture duration parameter ppJSON The duration is expressed in seconds and for an IQ stream represents the amount of data in seconds to be captured. After this capture period has expired from attaching to a device, the stream will automatically detach from the device and report 'trfExhausted' once all captured data has been retrieved. All devices will attempt to meet fIQCaptureSec at best-effort. In the case that the captured time will fit in internal buffering, the captured IQ sequence is guaranteed to be contiguous. If a stream duration is short enough to be fully buffered, the read-only stream parameter 'contiguous' will be present (and reads as 'true' (boolean)). A stream for which the 'contiguous' flag is not present may contain discontinuities. The read-only parameter 'maxContiguousSec' (_float32) gives the maximum capture length that can fit in device buffering and thus for which the 'contiguous' flag will be set with the current capture parameters.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
fIQCaptureSec	Number of seconds of data to capture.

Returns

error/warning information (on success == trfOk)

trfAddNumericParameter()

```
_TRFAPI trfStatus trfAddNumericParameter (
            char ** ppJSON,
            const char * pParameterName,
            _float32 fValue )
```

For parameters not named above - add parameter by name Add named numeric parameter to ppJSON.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
pParameterName	Name of parameter to add.
fValue	Numeric value to add.

Returns

error/warning information (on success == trfOk)

trfAddRBWHz()

```
_TRFAPI trfStatus trfAddRBWHz (
            char ** ppJSON,
            _float32 fRBWHz )
```

Add resolution bandwidth parameter to ppJSON. RBW parameter is required by spectrum streams.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
fRBWHz	Resolution Bandwidth parameter to add to JSON parameter string.

Returns

error/warning information (on success == trfOk)

trfAddRefLeveldBm()

```
_TRFAPI trfStatus trfAddRefLeveldBm (
            char ** ppJSON,
            _float32 fRefLeveldBm )
```

Add reference level to ppJSON The reference level is the strongest expected signal. Setting the reference level instructs a receiver to optimize its signal path, attenuation, gain settings to maximize available dynamic range for this expected signal level. A reference level is optional but recommended for IQ and Spectrum streams.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
fRefLeveldBm	Level in dBm of strongest expected signal.

Returns

error/warning information (on success == trfOk)

trfAddTextParameter()

```
_TRFAPI trfStatus trfAddTextParameter (
            char ** ppJSON,
            const char * pParameterName,
            const char * pValue )
```

Add named textual parameter to ppJSON.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
pParameterName	Name of parameter to add.
pValue	Textual value to add (zero-terminated string).

Returns

error/warning information (on success == trfOk)

trfAddWindowType()

```
_TRFAPI trfStatus trfAddWindowType (
            char ** ppJSON,
            const char * pWindow )
```

Add window type specification to ppJSON. A window type is optional for spectrum streams.

Parameters

ppJSON	Pointer to JSON string - accumulating parameters.
pWindow	Textual window type to add to JSON parameter string.

Returns

error/warning information (on success == trfOk)

trfAllowDiscarding()

```
_TRFAPI trfStatus trfAllowDiscarding (
            trfHandle cStream,
            bool bAllowDiscarding )
```

By default, the data contained in streams will be discarded to ensure buffer size constraints are respected. In certain cases - such as streaming from file - it is necessary to control the flow of data to keep a processing chain synchronized and not arbitrarily drop frames. This call allows streams that would normally contain non-discardable frames to allow discarding to occur. By setting this flag on (say) an IQ stream from a file that is being processed downstream, the user will not have to loop to keep this stream emptied to allow the processor to proceed - otherwise the stream would block waiting for the stream buffer to be emptied.

Parameters

cStream	Stream handle
bAllowDiscarding	If set true, then this ensures normal discarding behaviour is followed.

Returns

error/warning information. Expect trfOk.

Example:

```
// 1. Setup IQ stream
trfHandle hIQStream(trfInvalidHandle);
trfCreateIQStream(&hIQStream, 0.0f, 0.0f, 0.0f);
trfAllowDiscarding(hIQStream, true);  // Stream not serviced - prevents blocking
char *pJSON(nullptr);
trfAddBooleanParameter(&pJSON, TRFLooping, true);
trfSetParameters(hIQStream, &pJSON);
```

trfAttachProcessorToStream()

```
_TRFAPI trfStatus trfAttachProcessorToStream (
            trfHandle hProcessor,
            trfHandle hSourceStream )
```

Attach a processor to a source stream.

Parameters

hProcessor	Processor handle
hSourceStream	Stream handle

Returns

error/warning information (on success == trfOk)

Example:

```
// Attach hProcessor to its source IQ stream
eStatus = trfAttachProcessorToStream(hProcessor, hIQStream);
if (eStatus != trfOk) {
    cout « "trfAttachProcessorToStream failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
```

trfAttachStreamToDevice()

```
_TRFAPI trfStatus trfAttachStreamToDevice (
            trfHandle cDevice,
            trfHandle cStream )
```

Attach the given stream to the specified device. A stream may only be attached to one device at a time. If the device supports the stream, the call will return trfOk and the stream will start to produce data, subject to resource availability on the specified device.

Parameters

cDevice	Device handle
cStream	Stream handle

Returns

error/warning information (on success == trfOk)

Example:

```
eStatus = trfAttachStreamToDevice(hDevice, hStream);
if (eStatus != trfOk) {
    cout « "trfAttachStreamToDevice failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    char *pFullError(trfGetDetailedStatus());
    cout « "error info: " « std::string(pFullError) « endl;
    trfDisposeString(&pFullError);
    trfStatus eStatus(trfOk);
    while ((eStatus = trfGetNextStatus()) != trfOk) {
        cout « "Error code " « (int)eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    }
    trfClose(&hStream);
    trfClose(&hDevice);
    return false;
}
```

trfAttachStreamToFile()

```
_TRFAPI trfStatus trfAttachStreamToFile (
            trfHandle cStream,
            const char * pFilePath )
```

Attaches the given stream to an input file. Assumes that the file is valid and of the correct type for the stream - otherwise the call will fail. Playback from the stream is controllable through the following parameters that may be set for the stream: TRFLooping - boolean: if set true then the file will be played continuously by looping back to the start once the file has been completely delivered. Function parameters:

Parameters

cStream	Handle to stream to attach to file
∗*pFilePath*	Fully qualified input path for json/data file.

Returns

error/warning information (on success == trfOk)

Example:

```
trfHandle hStream(trfInvalidHandle);
std::cout « "Playing back Spectra from file " « sFile « std::endl;
trfStatus eStatus(trfCreateSpectrumStream(&hStream, 0.0f, 0.0f, 0.0f, nullptr, 0.0f));
if (eStatus != trfOk) {
    std::cout « "trfCreateSpectrumStream failed:" « eStatus « ": " « trfGetTextForStatus(eStatus) «
  std::endl;
    return false;
}
eStatus = trfAttachStreamToFile(hStream, sFile.c_str());
if (eStatus != trfOk) {
    std::cout « "trfAttachStreamToFile failed:" « eStatus « ": " « trfGetTextForStatus(eStatus) « std::endl;
    return false;
}

char *pJSON(nullptr);
trfGetParameters(hStream, &pJSON);
std::cout « "Stream parameters:" « std::endl « "---" « std::endl « pJSON « std::endl;
```

trfClose()

```
_TRFAPI trfStatus trfClose (
            trfHandle * pHandle )
```

Close any open handle (device, processor, stream) On successful return, ∗pHandle==kInvalidHandle.

Parameters

∗*pHandle*	Pointer to Device, processor, stream handle

Returns

error/warning information (on success == trfOk)

Example:

```
eStatus = trfClose(&pHandles[i]);
if (eStatus != trfOk) {
    cout « "trfClose [ " « i « "] failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    continue;
}
```

trfConvertBasebandFileToWavFile()

```
_TRFAPI trfStatus trfConvertBasebandFileToWavFile (
            const char * pSourceFilePath,
            const char * pDestinationFilePath )
```

Stand-alone function to convert a previously created Baseband stream file and produce a WAV file equivalent output file.

Parameters

∗pSourceFilePath	Fully qualified path to previously generated Baseband data file.
∗pDestinationFilePath	Fully qualified output path for equivalent wav file.

Returns

error/warning information (on success == trfOk)

Example:

```
const char *pBasebandFilePath(_TempFile);
eStatus = trfConvertBasebandFileToWavFile(pBasebandFilePath, pOutputFile);
if (eStatus != trfOk) {
    cout « "trfConvertBasebandFileToWavFile failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) «
  endl;
    return false;
}
```

trfCreateIQStream()

```
_TRFAPI trfStatus trfCreateIQStream (
            trfHandle * pStream,
            _float32 fCentreHz,
            _float32 fIFBWHz,
            _float32 fRefLeveldBm )
```

Create an IQ stream with the given parameters. This entity will be inactive until successfully attached to a device capable of servicing the stream. Note that in the current implementation of libtrf, IF bandwidths narrower than 100kHz are not supported. Additionally IF bandwidths of less than 1MHz below 50MHz FCentre exhibit significant distortion and their use is not recommended. This limitation will be addressed in planned future releases.

Parameters

∗pStream	Stream handle (returned on success)
fCentreHz	Centre frequency of frames to be returned by this stream
fIFBWHz	IF Bandwidth of frames to be returned by this stream
fRefLeveldBm	Reference Level in dBm of maximum expected signal

Returns

error/warning information (on success == trfOk)

Example:

```
_float32 fFCentreHz(2440.0e6f), fIFBWHz(40.0e6f);
_float32 fInitialRefLeveldBm(-50.0f);
trfHandle hStream(trfInvalidHandle);
eStatus = trfCreateIQStream(&hStream, fFCentreHz, fIFBWHz, fInitialRefLeveldBm);
if (eStatus != trfOk) {
    cout « "trfCreateIQStream failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
```

trfCreateProcessor()

```
_TRFAPI trfStatus trfCreateProcessor (
            trfHandle * pProcessor,
            const char * pType )
```

Create a processor entity of the given type. Returns a handle to the newly created processor.

Parameters

∗pProcessor	Processor handle (returned on success)
∗pType	Textual type of processor to be created

Returns

error/warning information (on success == trfOk)

Example:

```
// Instantiate an AM demodulator
trfHandle hProcessor(trfInvalidHandle);
eStatus = trfCreateProcessor(&hProcessor, "AMDemodulator");
if (eStatus != trfOk) {
    cout « "trfCreateProcessor failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
```

trfCreateSpectrumStream()

```
_TRFAPI trfStatus trfCreateSpectrumStream (
            trfHandle * pStream,
            _float32 fMinHz,
            _float32 fMaxHz,
            _float32 fRBWHz,
            const char * pWindowType,
            _float32 fRefLeveldBm )
```

Create a spectrum stream with the given parameters. This entity will be inactive until successfully attached to a device capable of servicing the stream.

Parameters

∗pStream	Stream handle (returned on success)
fMinHz	Minimum extent of swept frequency range
fMaxHz	Maximum extent of swept frequency range
fRBWHz	Resolution bandwidth of returned sweeps
pWindowType	Window type to use for spectrum reconstruction (nullptr - default)
fRefLeveldBm	Reference Level in dBm of maximum expected signal

Returns

error/warning information (on success == trfOk)

Example:

```
// Create a spectrum (sweep) stream from 1 to 2GHz with RBW of 20kHz
cout « "Creating a spectrum stream." « endl;
_float32 fFStartHz(1000.0e6f), fFStopHz(2000.0e6f), fRBWHz(20.0e3f), fRefLeveldBm(-50.0f);
const char *pWindow(nullptr);
trfHandle hStream(trfInvalidHandle);
eStatus = trfCreateSpectrumStream(&hStream, fFStartHz, fFStopHz, fRBWHz, pWindow, fRefLeveldBm);
if (eStatus != trfOk) {
    cout « "trfCreateSpectrumStream failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
```

trfDetachProcessor()

```
_TRFAPI trfStatus trfDetachProcessor (
            trfHandle hProcessor )
```

Detach a processor from its current source stream.

Parameters

hProcessor	Processor handle

Returns

error/warning information (on success == trfOk)

Example:

```
// Detach processor from its source stream
eStatus = trfDetachProcessor(hProcessor);
if (eStatus != trfOk) {
    cout « "Whups! " « eStatus « endl;
}
```

trfDetachStream()

```
_TRFAPI trfStatus trfDetachStream (
            trfHandle cStream )
```

Detach the given stream from any device to which it is connected. This will stop any capture associated with the stream - however the stream will still contain any buffered data that has been previously delivered.

Parameters

cStream	Receives stream handle on success

Returns

error/warning information (on success == trfOk)

Example:

```
// Detach stream halts capture
trfDetachStream(hStream);
if (trfIsAttached(hStream) != trfDetached) {
    cout « "Not showing stream as detached - weird" « endl;
    return false;
}
```

trfDisposeString()

```
_TRFAPI void trfDisposeString (
            char ** ppString )
```

Dispose of a previously allocated ppJSON string. Note that only strings that are returned via a parameter of the form 'char **' should be disposed of using this call. This is provided for syntactic convenience and is equivalent to: if (ppJSON != nullptr) { if (*ppJSON != nullptr) { delete [](*ppJSON); *ppJSON = nullptr; } }.

trfEnumerateDevices()

```
_TRFAPI trfStatus trfEnumerateDevices (
            _uint * puDevices,
            bool bAutoDiscover )
```

Enumerate all accessible devices and return a count. Device enumeration may be initiated by a user at any time to enumerate newly connected devices or update the list of devices.

Parameters

puDevices	Address of _uint to receive the number of devices known.
bAutoDiscover	If 'true', then attempt to find any accessible devices. Otherwise only devices explicitly added through trfAddDeviceAddress are included.

Returns

error/warning information (on success == trfOk)

Example:

```
_uint uDevices(0);
eStatus = trfEnumerateDevices(&uDevices, true);
if (eStatus != trfOk) {
    cout « "trfEnumerateDevices failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
if (0 == uDevices) {
    cout « "No ThinkRF devices enumerated" « endl;
    return false;
}
```

trfExamineDevice()

```
_TRFAPI trfStatus trfExamineDevice (
            _uint uDeviceIndex,
            char ** ppJSON )
```

Return known information about a discovered device (uDeviceIndex in the range [0 .. (uDevices-1)] into user-allocated trfDeviceInfo structure.

Parameters

uDeviceIndex	Index of device
ppJSON	Pointer to receive UTF8 JSON string

Returns

error/warning information (on success == trfOk)

Example:

```
  eStatus = trfExamineDevice(i, &pJSON);
  if (eStatus != trfOk) {
      cout « "trfExamineDevice [ " « i « "] failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) «
endl;
      continue;
  }
  cout « "Device:" « i « " information: " « endl « "---" « endl « pJSON « endl;
  trfDisposeString(&pJSON);
```

trfFlushStream()

```
_TRFAPI trfStatus trfFlushStream (
            trfHandle cStream )
```

Remove all frames currently present in a stream. If the stream is attached and delivering data, this will potentially create a break in the data. In the case that the stream is detached, this will remove any outstanding frames. After this call any call to trfGetNextXXX will return trfExhausted.

Parameters

cStream	Receives stream handle on success

Returns

error/warning information (on success == trfOk)

trfFreeBaseband()

```
_TRFAPI trfStatus trfFreeBaseband (
            trfBasebandFrame ** ppFrame )
```

Free the given IQ frame, which will no longer be valid after this call (NULL). The user should ensure that frames are disposed when no longer required to avoid STORAGE_CAPACITY_VIOLATION errors.

Parameters

ppFrame	Address of pointer to IQ frame to dispose of

Returns

error/warning information (on success == trfOk)

trfFreeIQ()

```
_TRFAPI trfStatus trfFreeIQ (
            trfIQFrame ** ppFrame )
```

Free the given IQ frame, which will no longer be valid after this call (NULL). The user should ensure that frames are disposed when no longer required to avoid STORAGE_CAPACITY_VIOLATION errors.

Parameters

ppFrame	Address of pointer to IQ frame to dispose of

Returns

error/warning information (on success == trfOk)

trfFreeSpectrum()

```
_TRFAPI trfStatus trfFreeSpectrum (
            trfSpectrumFrame ** ppFrame )
```

Free the given spectrum frame, which will no longer be valid after this call. The user should ensure that frames are disposed when no longer required to avoid STORAGE_CAPACITY_VIOLATION errors.

Parameters

ppFrame	Address of pointer to Spectrum frame to dispose of

Returns

error/warning information (on success == trfOk)

trfGetDetailedStatus()

```
_TRFAPI char * trfGetDetailedStatus (
            void  )
```

Return textual (UTF-8, ISO-Latin-1) error details for the last significant status on the calling thread. Clears error string for this thread on on call.

Returns

Error string - should be disposed of using trfDisposeString.

trfGetNextBaseband()

```
_TRFAPI trfStatus trfGetNextBaseband (
            trfHandle cStream,
            trfBasebandFrame ** ppFrame,
            _uint32 uTimeoutMsec )
```

Obtain the next Baseband frame (if available) from the stream. If no frame is yet available trfStatus will be 'trfWaiting'. If the stream has been halted then the status 'trfExhausted' may be returned to indicate there are no more buffered frames to retrieve. If the retrieval succeeds, trfStatus will be trfOk and ∗ppFrame will point to the returned Baseband frame. The frame should be disposed of using trfFreeBaseband. The trfExhausted return indicates that capture for this 'attach' is complete and the stream has been detached.

Parameters

cStream	Stream handle
ppFrame	Address of pointer to frame to recieve IQ data.
uTimeoutMsec	Milliseconds to wait for the next packet

Returns

error/warning information (on success == trfOk)

Example:

```
_uint32 uSamplesRemaining((_uint32)roundf(1.0f * kAudioRateHz));
while(uSamplesRemaining > 0) {
    trfBasebandFrame *pFrame(nullptr);
    while (true) {
        eStatus = trfGetNextBaseband(hAudioOut, &pFrame, 1000);
        if (eStatus == trfOk) {
            break;
        }
    }
    if (pFrame->pJSONInfo != nullptr) {
        cout « "Annotation JSON: " « pFrame->pJSONInfo « endl;
    }
    uSamplesRemaining = (pFrame->uSamples > uSamplesRemaining) ? 0: (uSamplesRemaining - pFrame->uSamples);
    cout « "Samples remaining:" « uSamplesRemaining « endl;
    trfFreeBaseband(&pFrame);
}
```

trfGetNextIQ()

```
_TRFAPI trfStatus trfGetNextIQ (
            trfHandle cStream,
            trfIQFrame ** ppFrame,
            _uint32 uTimeoutMsec )
```

Obtain the next IQ frame (if available) from the stream. If no frame is yet available trfStatus will be 'trfWaiting'. If the stream has been halted then the status 'trfExhausted' may be returned to indicate there are no more buffered frames to retrieve. If the retrieval succeeds, trfStatus will be trfOk and ∗ppFrame will point to the returned IQ frame.

The frame should be disposed of using trfFreeIQ. The trfExhausted return indicates that capture for this 'attach' is complete and the stream has been detached.

Parameters

cStream	Stream handle
ppFrame	Address of pointer to frame to recieve IQ data.
uTimeoutMsec	Milliseconds to wait for the next packet

Returns

error/warning information (on success == trfOk)

Example:

```
// Report the first uFramesRemaining IQFrames received and then detach
_uint uFramesRemaining(1000);
_uint32 uFramesCaptured(0), uSamplesCaptured(0);
while (uFramesRemaining > 0) {
    trfIQFrame *pIQFrame(nullptr);
    eStatus = trfGetNextIQ(hStream, &pIQFrame, 5000);   // Obtain next frame
    if (eStatus != trfOk) {
        cout « "trfGetNextIQ failed with code (possibly timed-out) " « eStatus « " - " «
  trfGetTextForStatus(eStatus) « endl;
        trfClose(&hStream);
        trfClose(&hDevice);
        return false;
    }

    // Report the received frame
    uFramesCaptured++;
    uSamplesCaptured += (_uint32)pIQFrame->uSamples;
    if (0 == (uFramesCaptured % 10)) {
        cout « uFramesCaptured « " frames captured - total " « uSamplesCaptured « " samples" « endl;
    }
    uFramesRemaining--;

    // Release the received frame
    trfFreeIQ(&pIQFrame);
}
```

trfGetNextSpectrum()

```
_TRFAPI trfStatus trfGetNextSpectrum (
            trfHandle cStream,
            trfSpectrumFrame ** ppFrame,
            _uint32 uTimeoutMsec )
```

Obtain the next frame (if available) from the stream. If no frame is yet available trfStatus will be 'WAITING' and ∗pp↩ Frame will be NULL otherwise trfStatus will be trfOk and ∗ppFrame will point to the returned data frame. The frame should be disposed of using trfFreeSpectrum.

Parameters

cStream	Stream handle
ppFrame	Address of pointer to receive frame pointer
uTimeoutMsec	Maximum time in milliseconds to wait for frame

Returns

error/warning information (on success == trfOk)

Example:

```
        _uint32 uPoints(0);
        _float32 fRBWHz(0.0f);
        bool bFirstFrame(true);
        for (_uint32 i = 0; i < 100; i++) {
            trfSpectrumFrame *pSpectrumFrame(nullptr);
            do {
                eStatus = trfGetNextSpectrum(hStream, &pSpectrumFrame, 5000);
                if (eStatus < trfOk) {
                    cout « "trfGetNextSpectrum failed: " « trfGetTextForStatus(eStatus) « ":" « (int)eStatus
« endl;
                    break;
                }
                else if (eStatus == trfNotStarted) {
                    cout « "s";
                    std::cout « trfGetVerboseInternalErrorState(100) « std::endl;
                    break;
                }
                else if (eStatus == trfWaiting) {
                    cout « "w";
                }
                else if (eStatus == trfExhausted) {
                    cout « "!";
                    break;
                }
            } while (pSpectrumFrame == nullptr);

            if (nullptr == pSpectrumFrame) {
                break;
            }

            if (uPoints != pSpectrumFrame->uPoints) {
                uPoints = pSpectrumFrame->uPoints;
                cout « "Frame points " « uPoints « endl;
            }
            if (fRBWHz != pSpectrumFrame->fRBWHz) {
                fRBWHz = pSpectrumFrame->fRBWHz;
                cout « "RBWHz " « fRBWHz « endl;
            }
            trfFreeSpectrum(&pSpectrumFrame);
            cout « (i%10) « flush;
        }
```

trfGetNextStatus()

```
_TRFAPI trfStatus trfGetNextStatus (
            void  )
```

Return any additional (descriptive) error codes arising from the last issued libtrf function on the calling thread. To return all error status a user can iterate on this call until it returns trfOk to indicate no further error information is available.

Returns

API status code (trfOk if no further status available)

trfGetParameterInfo()

```
_TRFAPI trfStatus trfGetParameterInfo (
            trfHandle cEntity,
            char ** ppJSON )
```

Obtains all available parameters for the specified device along with the valid ranges, type and any associated information through ppJSON.

Parameters

cEntity	Device/Processor/Stream handle
ppJSON	Pointer to receive UTF8 JSON string

Returns

error/warning information (on success == trfOk)

Example:

```
// Display demodulator parameter set
char *pParameterInfo(nullptr), *pParameters(nullptr);
eStatus = trfGetParameterInfo(hProcessor, &pParameterInfo);
cout « "Parameter info : " « endl « pParameterInfo « endl;
```

trfGetParameters()

```
_TRFAPI trfStatus trfGetParameters (
            trfHandle cEntity,
            char ** ppJSON )
```

Obtain the current values of device/processor parameters and make accessible through ppJSON.

Parameters

cEntity	Device/Processor/Stream handle
ppJSON	Pointer to receive UTF8 JSON string describing parameters

Returns

error/warning information (on success == trfOk)

Example:

```
trfGetParameters(hStream, &pJSON);
std::cout « "Stream parameters:" « std::endl « "---" « std::endl « pJSON « std::endl;
```

trfGetProcessorOutputStream()

```
_TRFAPI trfStatus trfGetProcessorOutputStream (
            trfHandle cProcessor,
            trfHandle * pStream )
```

Return the output stream from the given processor.

Parameters

cProcessor	Processor handle
pStream	Pointer to trfHandle to receive stream handle.

Example:

```
// Obtain baseband output stream from processor (AM Demodulator)
trfHandle hAudioOut(trfInvalidHandle);
eStatus = trfGetProcessorOutputStream(hProcessor, &hAudioOut);
if (eStatus != trfOk) {
    cout « "trfGetProcessorOutputStream failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
```

trfGetProcessorTypes()

```
_TRFAPI trfStatus trfGetProcessorTypes (
            char ** ppJSON )
```

Returns a JSON-formatted array names 'processortypes' of the available processor types.

Parameters

***ppJSON*	*pJSON contains the list of available processors on exit.

Returns

error/warning information (on success == trfOk)

Example:

```
// Get the list of available processor types
char *pTypes(nullptr);
eStatus = trfGetProcessorTypes(&pTypes);
cout « "trfGetProcessorTypes returns: " « endl « pTypes « endl;
trfDisposeString(&pTypes);
```

trfGetTextForStatus()

```
_TRFAPI const char * trfGetTextForStatus (
            trfStatus eStatus )
```

Return textual (UTF-8, ISO-Latin-1) status description matching with the passed status code.

Parameters

eStatus	API status code

Returns

Error strfing - remains owned by API.

trfGetVersion()

```
_TRFAPI char * trfGetVersion (
            void  )
```

Return a C-string describing this version of the library.

Returns

textual description of the library version. Should be disposed of using trfDisposeString using delete[].

Example:

```
char *pVersion(trfGetVersion());
cout « pVersion « endl;
trfDisposeString(&pVersion);
```

trfIsAttached()

```
_TRFAPI trfStatus trfIsAttached (
            trfHandle cStream )
```

Test if the specified stream is attached. Note that a finite capture stream will detach itself after capture is completed. This change will occur at some point in time after the completion of capture, and hence it is valid for this function to return trfOk whilst an associated trfGetNextXXXFrame has returned trfExhausted (finite capture).

Parameters

cStream	Stream handle

Returns

error/warning information. If attached returns trfOk. If detached returns the warning trfDetached. trfBadHandle may also be returned.

Example:

```
if (trfIsAttached(hStream) != trfDetached) {
    cout « "Stream is still attached to source" « endl;
    return false;
}
```

trfOpenDevice()

```
_TRFAPI trfStatus trfOpenDevice (
            _uint uDeviceIndex,
            trfHandle * pDevice )
```

Given a device index, attempt to open it. On success sets the device handle *pHandle.

Parameters

uDeviceIndex	Index of device from most recent call to trfEnumerateDevices
pDevice	Pointer to device handle to be used in future operations

Returns

error/warning information (on success == trfOk)

Example:

```
eStatus = trfOpenDevice(i, &pHandles[i]);
if (eStatus != trfOk) {
    cout « "trfOpenDevice [" « i « "] failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    continue;
}
```

trfReadBooleanParameter()

```
_TRFAPI trfStatus trfReadBooleanParameter (
            char ** ppJSON,
            const char * pParameterName,
            bool * pbValue )
```

Read named boolean parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pParameterName	Name of numeric parameter to read.
pbValue	Pointer to bool to receive value.

Returns

error/warning information (on success == trfOk)

trfReadFCentre()

```
_TRFAPI trfStatus trfReadFCentre (
            char ** ppJSON,
            _float32 * pfFCentreHz )
```

Read Centre frequency parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pfFCentreHz	Pointer to _float32 to receive value.

Returns

error/warning information (on success == trfOk)

trfReadFMinMax()

```
_TRFAPI trfStatus trfReadFMinMax (
            char ** ppJSON,
            _float32 * pfFMinHz,
            _float32 * pfFMaxHz )
```

Read min/max frequency range from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pfFMinHz	Pointer to _float32 to receive value.
pfFMaxHz	Pointer to _float32 to receive value.

Returns

error/warning information (on success == trfOk)

trfReadIFBWHz()

```
_TRFAPI trfStatus trfReadIFBWHz (
            char ** ppJSON,
            _float32 * pfFIFBWHz )
```

Read IFBW parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pfFIFBWHz	Pointer to _float32 to receive value.

Returns

error/warning information (on success == trfOk)

trfReadNumericParameter()

```
_TRFAPI trfStatus trfReadNumericParameter (
            char ** ppJSON,
            const char * pParameterName,
            _float32 * pfValue )
```

Read named numeric parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pParameterName	Name of numeric parameter to read.
pfValue	Pointer to _float32 to receive value.

Returns

error/warning information (on success == trfOk)

trfReadParameterAsJSON()

```
_TRFAPI trfStatus trfReadParameterAsJSON (
            char ** pJSON,
            const char * pParameter,
            char ** ppJSONExtract )
```

Read a sub-element from a JSON element as another JSON string. Useful for extracting substructure from a JSON string.

Parameters

pJSON	Source JSON string
pParameter	Name of parameter to extract
ppJSONExtract	Pointer to receive JSON string

Returns

error/warning information (on success == trfOk)

trfReadParameterInfoElement()

```
_TRFAPI trfStatus trfReadParameterInfoElement (
            char ** ppJSON,
            const char * pParameter,
            const char * pElement,
            _float32 * pfValue )
```

Read a sub-element from a JSON element as a _float32 value. Useful for obtaining 'min', 'max' and 'default' values from parameter information.

Parameters

ppJSON	Source JSON string
pParameter	Name of parameter to be extracted from

Parameters

pElement	Name of element (ie. "min", "max", "default")
pfValue	Pointer to _float32 to receive value

Returns

error/warning information (on success == trfOk)

trfReadRBWHz()

```
_TRFAPI trfStatus trfReadRBWHz (
            char ** ppJSON,
            _float32 * pfRBWHz )
```

Read resolution bandwidth parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pfRBWHz	Pointer to _float32 to receive value.

Returns

error/warning information (on success == trfOk)

trfReadRefLeveldBm()

```
_TRFAPI trfStatus trfReadRefLeveldBm (
            char ** ppJSON,
            _float32 * pfRefLeveldBm )
```

Read reference level parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pfRefLeveldBm	Pointer to _float32 to receive value.

Returns

error/warning information (on success == trfOk)

trfReadSampleRateHz()

```
_TRFAPI trfStatus trfReadSampleRateHz (
            char ** ppJSON,
            _float32 * pfSampleRateHz )
```

Read sample rate parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pfSampleRateHz	Pointer to _float32 to receive value.

Returns

error/warning information (on success == trfOk)

trfReadTextParameter()

```
_TRFAPI trfStatus trfReadTextParameter (
            char ** ppJSON,
            const char * pParameterName,
            char ** ppValue )
```

Read named textual parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
pParameterName	Name of textual parameter to read.
ppValue	Pointer to char pointer to receive value - allocated as []char.

Returns

error/warning information (on success == trfOk)

trfReadWindowType()

```
_TRFAPI trfStatus trfReadWindowType (
            char ** ppJSON,
            char ** ppWindow )
```

Read window type parameter from ppJSON.

Parameters

ppJSON	Pointer to JSON string containing parameter(s).
ppWindow	Pointer to char pointer to receive value - allocated as []char.

Returns

error/warning information (on success == trfOk)

trfResetDeviceConnection()

```
_TRFAPI trfStatus trfResetDeviceConnection (
            trfHandle cDevice )
```

If a communications error is suspected, issuing this call will cause a connection shutdown and re-open to the device without loosing any operational state. On success the device will continue with programmed operation. If the return value is not 'trfOk' then it is likely communications are unrecoverable and the device handle should be closed (see trfClose).

Parameters

cDevice	Device handle to be used in future operations

Returns

error/warning information (on success == trfOk)

Example:

```
  eStatus = trfResetDeviceConnection(hHandle);
  if (eStatus != trfOk) {
      cout « "trfResetDeviceConnection [" « i « "] failed:" « eStatus « " - " «
trfGetTextForStatus(eStatus) « endl;
  }
```

trfSetParameters()

```
_TRFAPI trfStatus trfSetParameters (
            trfHandle cEntity,
            char ** ppJSON )
```

Set the current value of a device/stream/processor parameter.

Parameters

cEntity	Device/Processor/Stream handle
ppJSON	Pointer to JSON string pointer, describing parameter(s) to be set. On return ∗ppJSON will be nullptr and storage will be disposed of.

Returns

error/warning information (on success == trfOk)

Example:

```
// Set AM demodulator parameters (obtain using trfGetParameterInfo)
trfAddNumericParameter(&pParameters, TRFOutputSampleRate, kAudioRateHz);//  Program output sample rate -
  44.1kHz
trfAddTextParameter(&pParameters, TRFAMSubtype, "DSB");// Program AM demodulator type
eStatus = trfSetParameters(hProcessor, &pParameters);
if (eStatus != trfOk) {
    cout « "trfSetParameters failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) « endl;
    return false;
}
trfDisposeString(&pParameters);
```

trfSetStreamOutputFile()

```
_TRFAPI trfStatus trfSetStreamOutputFile (
            trfHandle cStream,
            const char * pFilePath )
```

Attaches the specified output file - or if the file exists and is not empty, a numbered sibling file to the specified stream. The self-contained output file may be replayed using the trfCreateFileSourceStream function. The output file shall have the extension '.data' (ThinkRF Spectrum, IQ, Baseband), The binary file shall each be accompanied by a metadata file (.json). If pFilePath is NULL, any file writing for the stream shall be immediately halted.

Parameters

cStream	Stream handle
pFilePath	Fully qualified output path for data file.

Returns

error/warning information (on success == trfOk)

Example:

```
trfSetStreamOutputFile(hStream, pFilename);
if (eStatus != trfOk) {
    std::cout « "trfSetStreamOutputFile failed:" « eStatus « " - " « trfGetTextForStatus(eStatus) «
  std::endl;
    trfClose(&hStream);
    return false;
}
```

6.3 libtrf.h

Go to the documentation of this file.

```
00001 /*****************************************************************************
00002 * @file libtrf.h
00003 *
00004 * @brief ThinkRF Device common API header file
00005 *****************************************************************************
00006 * Language: C99 ISO/IEC 9899:1999
00007 * File Version: 1.0
00008 * @date 12Apr2023
00009 *****************************************************************************
00010 * Code Copyright (C) 2020-2023 ThinkRF Incorporated. All Rights Reserved.
00011 *****************************************************************************/
```

```
00012
00013 #ifndef _LIBTRF_
00014 #define _LIBTRF_
00015
00016
00017 //*** Macros
00018 //****************************************************************************
00019 // Build-control for Windows and Linux platforms
00020 //----------------------------------------------------------------------------
00021 #if defined (_WIN32) | defined (_WIN64)
00022  #ifdef _DLL
00023   #define _TRFAPI extern "C" __declspec(dllexport)
00024  #else
00025   #define _TRFAPI extern "C" __declspec(dllimport)
00026  #endif
00027 #elif defined (__GNUG__)
00028  #include <sys/stat.h>
00029  #include <fcntl.h>
00030  #ifdef _SHAREDLIB
00031   #define _TRFAPI __attribute__((visibility("default"))) extern "C"
00032  #else
00033   #define _TRFAPI extern "C"
00034  #endif
00035 #else
00036  #error "Unknown build platform"
00037 #endif
00038
00039 //*** Defines
00040 //****************************************************************************
00046 #define trfInvalidHandle        (0xffffffff)
00047
00048
00061 //----------------------------------------------------------------------------
00062 #ifndef _PARAMETERS_
00063
00064 // JSON names used for defining parameter ranges and similar
00065 #define TRFDefault                  "default"
00066 #define TRFMin                      "min"
00067 #define TRFMax                      "max"
00068 #define TRFType                     "type"
00069 #define TRFUnits                    "units"
00070 #define TRFAddress                  "address"
00071
00072 // Entity-specific JSON parameter names
00073 #define TRFDevice                   "device"
00074 #define TRFDeviceType               "type"
00075 #define TRFDeviceVersion            "version"
00076 #define TRFDeviceSerialNum          "serial"
00077 #define TRFDeviceFirmware           "firmware"
00078 #define TRFDeviceConnection         "connection"
00079 #define TRFHasFlatteningInfo        "iqequalization"
00080
00081 #define TRFSequence                 "Sequence"
00082 #define TRFTimestamp                "Timestamp"
00083 #define TRFTimeResolution           "TimeResolution"
00084 #define TRFBandwidthHz              "BandwidthHz"
00085
00086 #define TRFFilename                 "filename"
00087 #define TRFLooping                  "LoopingFlag"
00088 #define TRFSweepActive              "SweepActive"
00089 #define TRFIQActive                 "IQActive"
00090 #define TRFBufferSec                "bufferSec"
00092 #define TRFBufferFrames             "bufferFrames"
00094 #define TRFDiscardProportion        "bufferDiscard"
00100 #define TRFFMinHz                   "FMinHz"
00101 #define TRFFMaxHz                   "FMaxHz"
00102 #define TRFRBWHz                    "RBWHz"
00103 #define TRFWindow                   "WindowFn"
00104 #define TRFFCentreHz                "FCentreHz"
00105 #define TRFSampleRateHz             "SampleRateHz"
00106 #define TRFIFBWHz                   "IFBWHz"
00111 #define TRFRefLevel                 "RefdBm"
00112 #define TRFUserCalibrationOffset    "UserCaldB"
00113 #define TRFFlattenFlag              "flatten"
00114 #define TRFDurationFrames           "captureFrames"
00115 #define TRFDurationSec              "captureSec"
00116 #define TRFMaxContiguousSec         "maxContiguousSec"
00117 #define TRFIsContiguous             "contiguous"
00118 #define TRFFramesExpected           "framesExpected"
00119 #define TRFSubOptimalFlag           "SubOptimalDR"
00120 #define TRFFullAdaptFlag            "adaptFull"
```

```
00121 #define TRFStepAdaptFlag            "adaptStep"
00122 #define TRFMeasuredAvgdBm           "avgdBm"
00123 #define TRFdBFSD                    "dBFSD"
00124 #define TRFFlowControl              "flowControl"
00125
00126 #define TRFAttenuation              "attenuation"
00127 #define TRFPacketDurationMsec       "PacketDuration"
00128 #define TRFFilterRatio              "IFFilterRatio"
00129 #define TRFNTP                      "NTPServers"
00130 #define TRFSCPIQueryTimeout         "SCPIQueryTimeout"
00131
00132 #define TRFRxSampleRate             "RxSampleRate"
00133 #define TRFGNSSValid                "GNSSValid"
00134 #define TRFLatitude                 "latitude"
00135 #define TRFLongitude                "longitude"
00136 #define TRFAltitude                 "altitude"
00137 #define TRFGNSSRxTime               "GNSSPosnEpoch"
00138 #define TRFGNSSTimestamp            "GNSSTimeNanosec"
00139 #define TRFGNSSADelay               "GNSSAntDelay"
00140 #define TRFGNSSConstellation        "GNSSCons"
00141 #define TRFGNSSDynamicMode          "GNSSDynamic"
00142 #define TRFGNSSSpeedOverGround      "GNSSSpeed"
00143 #define TRFGNSSHeadingDegT          "GNSSHeading"
00144 #define TRFGNSSTrackDegT            "GNSSTrack"
00145 #define TRFGNSSMagVarDegT           "GNSSMagVar"
00146
00147 #define TRFDeviceTemperature        "deviceTemperature"
00148 #define TRFDeviceLockStatus         "deviceClockLock"
00149 #define TRFPLLSource                "pllSource"
00150 #define TRFPPSSource                "ppsSource"
00151 #define TRFPPSSource                "ppsSource"
00152 #define TRFTimeSync                 "timeSync"
00153 #define TRFCurrentDate              "Date"
00154 #define TRFCurrentTime              "Time"
00155
00156 // AM/FM Demodulator parameter names
00157 #define TRFAMSubtype                "type"
00158 #define TRFFramesProduced           "frames"
00159 #define TRFOutputSampleRate         "OutputSampleRate"
00160
00161 // IQ to Spectrum processor parameter names (re-uses TRFRBWHz, TRFWindowFn)
00162 #define TRFFollowIQ                 "FollowIQ"
00165 #define TRFOverlap                  "FFTOverlap"
00167 #define TRFMatchIQ                  "MatchIQ"
00168 #define TRFOutputSize               "OutputSize"
00169 #define TRFRecentFrames             "RecentFrames"
00170
00171 // Spectrum Characterization processor parameter names
00172 #define TRFAverageCount             "Average"
00173 #define TRFClearAverageTrace        "ClearAverageTrace"
00174 #define TRFClearMaxHoldTrace        "ClearMaxHoldTrace"
00175 #define TRFClearMinHoldTrace        "ClearMinHoldTrace"
00176
00177 #endif // _PARAMETERS_
00178
00179
00180 //*** Type definitions
00181 //*****************************************************************************
00183 //-----------------------------------------------------------------------------
00184 #ifndef _TRFTYPES_
00185
00186 #ifdef __GNUC__
00187 typedef char _int8;
00188 typedef short _int16;
00189 typedef int _int32;
00190 typedef long long int _int64;
00191 #endif
00192
00193 typedef unsigned short _uint16;
00194 typedef unsigned int _uint32;
00195 typedef unsigned int _uint;
00196 typedef unsigned long long int _uint64;
00197 typedef float _float32;
00198 typedef double _float64;
00199
00201 typedef struct {
00202     _float32 fI;
00203     _float32 fQ;
00204 }_complex32;
00205
00206 #endif //_TRFTYPES_
```

```
00207
00208
00210 //-------------------------------------
00211 typedef _uint32  trfHandle;
00212
00213
00215 //-------------------------------------
00216 typedef enum _trfStatus {
00217     trfOk = 0,
00218
00220     trfWaiting = 1,
00221     trfExhausted = 2,
00222     trfNotStarted = 3,
00223     trfDiscontinuousWithPreviousFrame = 4,
00224     trfDetached = 5,
00225     //...
00226
00228     trfUnimplemented = -1,
00229     trfAPINotInitialized = -2,
00230
00231     trfDeviceAddressAlreadyKnown = -3,
00232     trfUnallocatedUserData = -4,
00233     trfDeviceIndexOutOfRange = -5,
00234     trfNoDeviceInformation = -6,
00235     trfCannotOpenDevice = -7,
00236     trfBadDeviceHandle = -8,
00237     trfBadUnitHandle = -9,
00238     trfBadParameterName = -10,
00239     trfNoParametersSpecified = -11,
00240     tffBadParameter = -12,
00241     trfInvalidParameter = -13,
00242     trfBadReceiverHandle = -14,
00243     trfIQStreamStartFailure = -15,
00244     trfMemoryAllocationError = -16,
00245     trfInvalidStreamHandle = -17,
00246     trfNoAssociatedRxOrFile = -18,
00247     trfNotAnIQStream = -19,
00248     trfNoDataSpecified = -20,
00249     trfInvalidHandleSpecified = -21,
00250     trfBadStreamShutdown = -22,
00251     trfUnsupportedParameter = -23,
00252     trfCannotCloseStream = -24,
00253     trfNotASpectrumStream = -25,
00254     trfCannotRestartStream = -26,
00255     trfSweepStartFailure = -27,
00256     trfFrameTypeMismatch = -28,
00257     trfFileOpenFailed = -29,
00258     trfFileTypeIsNotIQ = -30,
00259     trfFileTypeIsNotSpectrum = -31,
00260     trfNoAssociatedSource = -32,
00261     trfUnknownParameter = -33,
00262     trfParameterSetError = -34,
00263     trfDeviceHandleIsInvalid = -35,
00264     trfSCPISendFailed = -36,
00265     trfSCPIQueryFailed = -37,
00266     trfBadStreamHandle = -38,
00267     trfCannotAttachIQStream = -39,
00268     trfCannotAttachSpectrumStream = -40,
00269     trfCannotAttachStream = -41,
00270     trfCannotDetachIQStream = -42,
00271     trfCannotDetachSpectrumStream = -43,
00272     trfCannotDetachStream = -44,
00273     trfConnectionResetFailed = -45,
00274     trfBadFrequencyParameters = -46,
00275     trfBadRBWParameter = -47,
00276     trfBadReferenceLevelParameter = -48,
00277     trfSpectrumStreamCreateFailure = -49,
00278     trfBufferSecOutOfRange = -50,
00279     trfBufferFramesOutOfRange = -51,
00280     trfBufferDiscardProportionOutOfRange = -52,
00281     trfFrequencyRangeIsInvalid = -53,
00282     trfBadGNSSDynamicMode = -54,
00283     trfBadPLLSource = -55,
00284     trfPLLSourceWithNoGNSS = -56,
00285     trfFCentreInvalid = -57,
00286     trfIFBWInvalid = -58,
00287     trfFMinInvalid = -59,
00288     trfFMaxInvalid = -60,
00289     trfResolutionBWInvalid = -61,
00290     trfFCentreOutOfRange = -62,
00291     trfIFBWOutOfRange = -63,
```

```
00292     trfReferenceLevelOutOfRange = -64,
00293     trfResolutionBWOutOfRange = -65,
00294     trfUnknownWindowType = -66,
00295     trfStartFrequencyOutOfRange = -67,
00296     trfStopFrequencyOutOfRange = -68,
00297     trfDeviceUnreachable = -69,
00298     trfInvalidFilename = -70,
00299     trfCannotOpenOutputFile = -71,
00300     trfUninitializedUserData = -72,
00301     trfBadProcessorHandle = -73,
00302     trfCannotCreateProcessor = -74,
00303     trfProcessorAttachFailed = -75,
00304     trfProcessorDetachFailed = -76,
00305     trfProcessorUnattached = -77,
00306     trfNotABasebandStream = -78,
00307     trfFileTypeIsNotBaseband = -79,
00308     trfBasebandStreamStartFailure = -80,
00309     trfSampleRateInvalid = -81,
00310     trfCannotObtainOutputStream = -82,
00311     trfCannotConnectOutputStream = -83,
00312     trfBadFilename = -84,
00313     trfWindowTypeInvalid = -85,
00314     trfDeviceCommunicationsLost = -86,
00315     trfUnknownStreamType = -87,
00316     trfStreamHandleAlreadyOpen = -88,
00317     trfInvalidStreamSource = -89,
00318     trfBadPPSSource = -90,
00319
00320     trfLastErrorSentinel = -1000
00321 }trfStatus;
00322
00323
00324 // trfIQFrame
00327 //------------------------------------
00328 typedef struct _trfIQFrame {
00329     trfStatus eFlags;
00330     _uint32 uSequenceNumber;
00331     _int64 iTimestampNanosec;
00332     _float64 fCentreHz;
00333     _float32 fIFBWHz;
00334     _float32 fSampleRateHz;
00335     bool bDiscontinuity;
00338     bool bSubOptimalDRFlag;
00339     _uint32 uSamples;
00340     const _complex32 *pData;
00341     const char *pJSONInfo;
00342 }trfIQFrame;
00343
00344
00345 // trfSpectrumFrame
00348 //------------------------------------
00349 typedef struct _trfSpectrumFrame {
00350     trfStatus eFlags;
00351     _uint32 uSequenceNumber;
00352     _int64 iTimestampNanosec;
00353     _int64 iDurationNanosec;
00357     _float64 fMinHz;
00358     _float64 fMaxHz;
00359     _float32 fRBWHz;
00360     _uint32 uPoints;
00361     const _float32 *pData;
00362     const char *pJSONInfo;
00363     _uint32 uAdditionalSpectra;
00364     const _float32 **pAdditionalSpectra;
00365 }trfSpectrumFrame;
00366
00367
00368 // trfBasebandFrame
00372 //------------------------------------
00373 typedef struct _trfBasebandFrame {
00374     _uint32 uSequenceNumber;
00375     _int64 iTimestampNanosec;
00376     _float32 fSampleRateHz;
00377     _float32 fUsableBWHz;
00378     _uint32 uChannels;
00379     _uint32 uSamples;
00380     const _float32 **ppData;
00382     const char *pJSONInfo;
00383 }trfBasebandFrame;
00384
00385
```

```
00386 //*** Global declarations
00387 //****************************************************************************
00388
00389 //*** Housekeeping functions
00390 //****************************************************************************
00391 // trfGetVersion
00396 //--------------------------------------
00397 _TRFAPI char *trfGetVersion(void);
00398
00399
00400 //*** Error handling functions
00401 //****************************************************************************
00402 // trfGetTextForStatus
00407 //--------------------------------------
00408 _TRFAPI const char *trfGetTextForStatus(trfStatus eStatus);
00409
00410 // trfGetNextStatus
00416 //--------------------------------------
00417 _TRFAPI trfStatus trfGetNextStatus(void);
00418
00419 // trfGetDetailedStatus
00423 //--------------------------------------
00424 _TRFAPI char *trfGetDetailedStatus(void);
00425
00426
00427 //*** Device discovery, open/close
00428 //****************************************************************************
00429 // trfAddDeviceAddress
00437 //--------------------------------------
00438 _TRFAPI trfStatus trfAddDeviceAddress(const char *pRemoteAddress);
00439
00440 // trfEnumerateDevices
00450 //--------------------------------------
00451 _TRFAPI trfStatus trfEnumerateDevices(_uint *puDevices, bool bAutoDiscover);
00452
00453 // trfExamineDevice
00460 //--------------------------------------
00461 _TRFAPI trfStatus trfExamineDevice(_uint uDeviceIndex, char **ppJSON);
00462
00463 // trfOpenDevice
00470 //--------------------------------------
00471 _TRFAPI trfStatus trfOpenDevice(_uint uDeviceIndex, trfHandle *pDevice);
00472
00473 // trfResetDeviceConnection
00482 //--------------------------------------
00483 _TRFAPI trfStatus trfResetDeviceConnection(trfHandle cDevice);
00484
00485 // trfClose
00491 //--------------------------------------
00492 _TRFAPI trfStatus trfClose(trfHandle *pHandle);
00493
00494
00495 //*** Stream Handling
00496 //****************************************************************************
00497 // trfCreateIQStream
00510 //--------------------------------------
00511 _TRFAPI trfStatus trfCreateIQStream(
00512     trfHandle *pStream,
00513     _float32 fCentreHz, _float32 fIFBWHz,
00514     _float32 fRefLeveldBm
00515 );
00516
00517 // trfGetNextIQ
00531 //--------------------------------------
00532 _TRFAPI trfStatus trfGetNextIQ(trfHandle cStream, trfIQFrame **ppFrame, _uint32 uTimeoutMsec);
00533
00534 // trfFreeIQ
00541 //--------------------------------------
00542 _TRFAPI trfStatus trfFreeIQ(trfIQFrame **ppFrame);
00543
00544 // trfCreateSpectrumStream
00555 //--------------------------------------
00556 _TRFAPI trfStatus trfCreateSpectrumStream(
00557     trfHandle *pStream,
00558     _float32 fMinHz, _float32 fMaxHz, _float32 fRBWHz,
00559     const char *pWindowType, _float32 fRefLeveldBm
00560 );
00561
00562 // trfGetNextSpectrum
00572 //--------------------------------------
00573 _TRFAPI trfStatus trfGetNextSpectrum(trfHandle cStream, trfSpectrumFrame **ppFrame, _uint32 uTimeoutMsec);
```

```
00574
00575 // trfFreeSpectrum
00582 //-------------------------------------
00583 _TRFAPI trfStatus trfFreeSpectrum(trfSpectrumFrame **ppFrame);
00584
00585 // trfGetNextBaseband
00599 //-------------------------------------
00600 _TRFAPI trfStatus trfGetNextBaseband(trfHandle cStream, trfBasebandFrame **ppFrame, _uint32 uTimeoutMsec);
00601
00602 // trfFreeBaseband
00609 //-------------------------------------
00610 _TRFAPI trfStatus trfFreeBaseband(trfBasebandFrame **ppFrame);
00611
00612
00613 //*** Stream manipulation
00614 //****************************************************************************
00615 // trfAttachStreamToDevice
00624 //-------------------------------------
00625 _TRFAPI trfStatus trfAttachStreamToDevice(trfHandle cDevice, trfHandle cStream);
00626
00627 // trfDetachStream
00634 //-------------------------------------
00635 _TRFAPI trfStatus trfDetachStream(trfHandle cStream);
00636
00637 // trfFlushStream
00644 //-------------------------------------
00645 _TRFAPI trfStatus trfFlushStream(trfHandle cStream);
00646
00647 // trfIsAttached
00657 //-------------------------------------
00658 _TRFAPI trfStatus trfIsAttached(trfHandle cStream);
00659
00660 // trfAllowDiscarding
00675 //-------------------------------------
00676 _TRFAPI trfStatus trfAllowDiscarding(trfHandle cStream, bool bAllowDiscarding);
00677
00678
00679 //*** Stream file handling
00680 //****************************************************************************
00681 // trfSetStreamOutputFile
00692 //-------------------------------------
00693 _TRFAPI trfStatus trfSetStreamOutputFile(trfHandle cStream, const char *pFilePath);
00694
00695 // trfAttachStreamToFile
00708 //-------------------------------------
00709 _TRFAPI trfStatus trfAttachStreamToFile(trfHandle cStream, const char *pFilePath);
00710
00711 // trfConvertBasebandFileToWavFile
00718 //-------------------------------------
00719 _TRFAPI trfStatus trfConvertBasebandFileToWavFile(const char *pSourceFilePath, const char
      *pDestinationFilePath);
00720
00721
00722 //*** Processor Handling
00723 //****************************************************************************
00724 // trfGetProcessorTypes
00730 //-------------------------------------
00731 _TRFAPI trfStatus trfGetProcessorTypes(char **ppJSON);
00732
00733 // trfCreateProcessor
00740 //-------------------------------------
00741 _TRFAPI trfStatus trfCreateProcessor(trfHandle *pProcessor, const char *pType);
00742
00743 // trfAttachProcessorToStream
00749 //-------------------------------------
00750 _TRFAPI trfStatus trfAttachProcessorToStream(trfHandle hProcessor, trfHandle hSourceStream);
00751
00752 // trfDetachProcessor
00757 //-------------------------------------
00758 _TRFAPI trfStatus trfDetachProcessor(trfHandle hProcessor);
00759
00760 // trfGetProcessorOutputStream
00765 //-------------------------------------
00766 _TRFAPI trfStatus trfGetProcessorOutputStream(trfHandle cProcessor, trfHandle *pStream);
00767
00768
00769 //*** Entity parameter control
00770 //****************************************************************************
00771 // trfGetParameterInfo
00778 //-------------------------------------
00779 _TRFAPI trfStatus trfGetParameterInfo(trfHandle cEntity, char **ppJSON);
```

```
00780
00781 // trfReadParameterInfoElement
00789 //-------------------------------------
00790 _TRFAPI trfStatus trfReadParameterInfoElement(
00791     char **ppJSON, const char *pParameter, const char *pElement, _float32 *pfValue
00792 );
00793
00794 // trfReadParameterAsJSON
00801 //-------------------------------------
00802 _TRFAPI trfStatus trfReadParameterAsJSON(
00803     char **pJSON, const char *pParameter, char **ppJSONExtract
00804 );
00805
00806 // trfGetParameters
00813 //-------------------------------------
00814 _TRFAPI trfStatus trfGetParameters(trfHandle cEntity, char **ppJSON);
00815
00816 // trfSetParameters
00823 //-------------------------------------
00824 _TRFAPI trfStatus trfSetParameters(trfHandle cEntity, char **ppJSON);
00825
00826 // trfDisposeString
00836 //-------------------------------------
00837 _TRFAPI void trfDisposeString(char **ppString);
00838
00839 // trfAddNumericParameter
00846 //-------------------------------------
00847 _TRFAPI trfStatus trfAddNumericParameter(char **ppJSON, const char *pParameterName, _float32 fValue);
00848
00849 // trfAddTextParameter
00855 //-------------------------------------
00856 _TRFAPI trfStatus trfAddTextParameter(char **ppJSON, const char *pParameterName, const char *pValue);
00857
00858 // trfAddBooleanParameter
00864 //-------------------------------------
00865 _TRFAPI trfStatus trfAddBooleanParameter(char **ppJSON, const char *pParameterName, bool bValue);
00866
00867 // trfReadNumericParameter
00873 //-------------------------------------
00874 _TRFAPI trfStatus trfReadNumericParameter(char **ppJSON, const char *pParameterName, _float32 *pfValue);
00875
00876 // trfReadTextParameter
00882 //-------------------------------------
00883 _TRFAPI trfStatus trfReadTextParameter(char **ppJSON, const char *pParameterName, char **ppValue);
00884
00885 // trfReadBooleanParameter
00891 //-------------------------------------
00892 _TRFAPI trfStatus trfReadBooleanParameter(char **ppJSON, const char *pParameterName, bool *pbValue);
00893
00894
00895 // Syntactic sugar for common device parameters to aid in constructing
00896 // ppJSON to be used with trfSetParameters.
00897 //*************************************
00898 // trfAddFCentre
00904 //-------------------------------------
00905 _TRFAPI trfStatus trfAddFCentre(char **ppJSON, _float32 fFCentreHz);
00906
00907 // trfAddIFBWHz
00912 //-------------------------------------
00913 _TRFAPI trfStatus trfAddIFBWHz(char **ppJSON, _float32 fFIFBWHz);
00914
00915 // trfAddFMinMax
00922 //-------------------------------------
00923 _TRFAPI trfStatus trfAddFMinMax(char **ppJSON, _float32 fFMinHz, _float32 fFMaxHz);
00924
00925 // trfAddRBWHz
00931 //-------------------------------------
00932 _TRFAPI trfStatus trfAddRBWHz(char **ppJSON, _float32 fRBWHz);
00933
00934 // trfAddWindowType
00940 //-------------------------------------
00941 _TRFAPI trfStatus trfAddWindowType(char **ppJSON, const char *pWindow);
00942
00943 // trfAddRefLeveldBm
00953 //-------------------------------------
00954 _TRFAPI trfStatus trfAddRefLeveldBm(char **ppJSON, _float32 fRefLeveldBm);
00955
00956 // trfAddIQCaptureSec
00975 //-------------------------------------
00976 _TRFAPI trfStatus trfAddIQCaptureSec(char **ppJSON, _float32 fIQCaptureSec);
00977
```

```
00978 // trfReadFCentre
00983 //-------------------------------------
00984 _TRFAPI trfStatus trfReadFCentre(char **ppJSON, _float32 *pfFCentreHz);
00985
00986 // trfReadIFBWHz
00991 //-------------------------------------
00992 _TRFAPI trfStatus trfReadIFBWHz(char **ppJSON, _float32 *pfFIFBWHz);
00993
00994 // trfReadFMinMax
01000 //-------------------------------------
01001 _TRFAPI trfStatus trfReadFMinMax(char **ppJSON, _float32 *pfFMinHz, _float32 *pfFMaxHz);
01002
01003 // trfReadRBWHz
01008 //-------------------------------------
01009 _TRFAPI trfStatus trfReadRBWHz(char **ppJSON, _float32 *pfRBWHz);
01010
01011 // trfReadWindowType
01016 //-------------------------------------
01017 _TRFAPI trfStatus trfReadWindowType(char **ppJSON, char **ppWindow);
01018
01019 // trfReadSampleRateHz
01024 //-------------------------------------
01025 _TRFAPI trfStatus trfReadSampleRateHz(char **ppJSON, _float32 *pfSampleRateHz);
01026
01027 // trfReadRefLeveldBm
01032 //-------------------------------------
01033 _TRFAPI trfStatus trfReadRefLeveldBm(char **ppJSON, _float32 *pfRefLeveldBm);
01034
01035 #endif // _LIBTRF_
```

Index