Location Services

Developing Bluetooth Location Services Applications

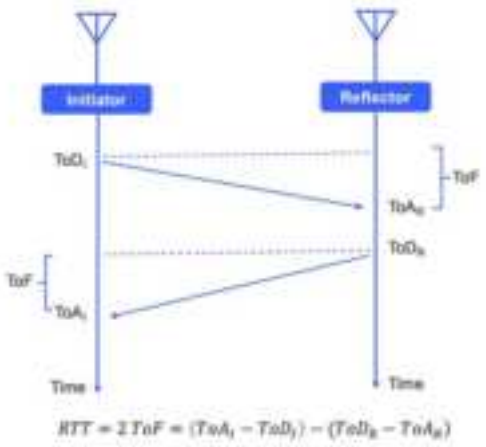
You are viewing documentation for version: 0.1 | Version History

Developing Bluetooth Location Services Applications

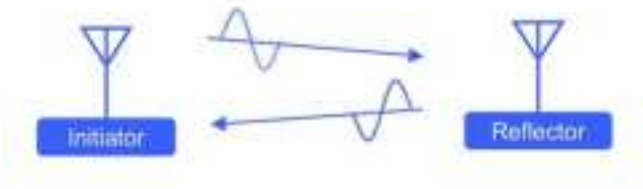
Bluetooth Location Services include applications for Channel Sounding, Direction Finding (Angle of Arrival and Angle of Departure), Triangulation, and Trilateration.

Channel Sounding is a method for estimating distances between two devices, designated as the initiator and the reflector. They exchange information across 72 RF physical channels with the initiator transmitting first, followed by the reflector's response. This alternating transmission and reception enables precise distance measurements. The technique employs Round-Trip Time (RTT) or Phase-Based Ranging (PBR) or both for accurate coordination across the 2.4 GHz spectrum. It also supports one to four antenna paths to improve measurement accuracy and reliability.

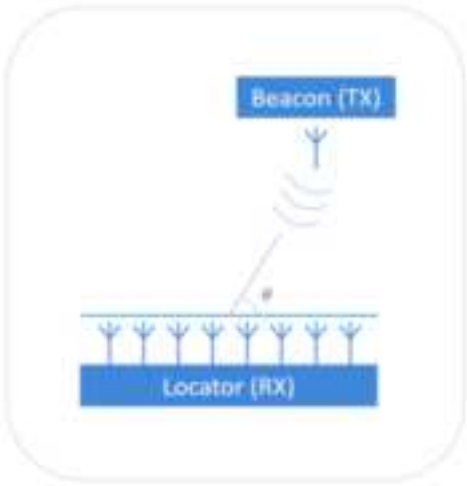
Round-Trip Time



Phase-Based Ranging



The two Direction Finding systems are Angle of Arrival (AoA) and Angle of Departure (AoD). With these technologies, the fundamental problem of locationing comes down to solving the arrival and departure angles of radio frequency signals.



The content on these pages is intended for those who want to experiment with or are already developing a Bluetooth Channel Sounding or Direction Finding application using the RTL Library provided with the Bluetooth LE SDK.

For details about this release: The silabs.com SDK Release Notes page contains alphabetically-arranged release notes for all the SDKs in the Simplicity SDK Suite. Scroll down to Bluetooth Location Services and find the corresponding version link.

For Silicon Labs' Location Services product information: See the product pages on silabs.com for Channel Sounding and Direction Finding (AoA/AoD).

For background about both solutions: The fundamentals sections are good places to start. See Channel Sounding Fundamentals or Direction Finding Fundamentals (PDF).

To get started with development: See Channel Sounding Getting Started or Direction Finding Getting Started (PDF).

If you are already in development: Proceed to the Channel Sounding Developer's Guide or go directly to the API Reference. For Direction Finding, review the linked documents.

Fundamentals

You are viewing documentation for version: 0.1 | Version History

Bluetooth LE Channel Sounding Fundamentals

Introduction

The determination of distance between two devices can be approached through a variety of methods, including the utilization of electromagnetic waves, ultrasonic waves, laser, infrared waves, and radio waves.

For radio wave-based distance estimation, several approaches exist. Among them, the commonly used approaches include:

- *Received Signal Strength Indicator (RSSI)* based
- *Time* based
- *Phase* based

Existing Bluetooth LE (BLE) solutions such as real-time tracking, access control, and proximity based services heavily rely on distance estimation using an RSSI approach and/or Angle of Arrival (AoA) and Angle of Departure (AoD) technologies.

Channel Sounding

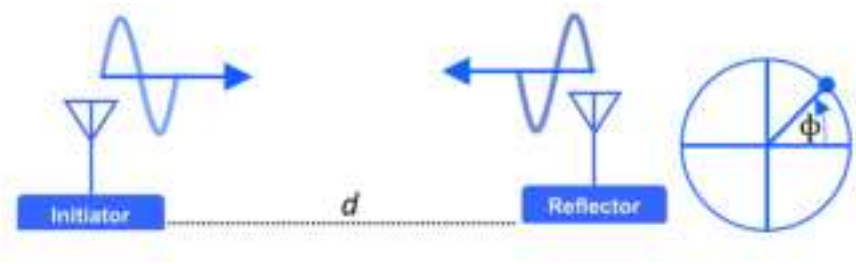
Channel Sounding (CS) is a new feature introduced by the Bluetooth SIG for distance estimation between two Bluetooth LE devices. The standard employs both phase and time based approaches which are widely adopted in other wireless technologies. Using these approaches, and the optional multiple antenna support, the expected accuracy can be significantly higher than RSSI based measurements.

The standard defines 79 RF channels with 1 MHz separation for CS. Out of these, up to 72 channels are used for the actual transmission of CS tones and packets, while the rest are excluded due to overlap with primary advertising channels.

In CS, two peer devices, known as Initiator and Reflector, exchange information that is measured to produce a distance estimate between them. In CS, the Initiator transmits first, followed by the Reflector's response. A device supporting Bluetooth LE CS supports both the phase based and round-trip time measurement approaches explained below.

Distance Estimation Based on Phase-Based Ranging (PBR)

The **Phase-Based Ranging (PBR)** approach works based on the principle that the phase shift introduced by a pure line of sight radio channel on a signal is a linear function of the distance between the transceivers and the frequency of the wave.



Phase-Based Ranging (PBR)

For a radio wave with period , it's angular speed is:

Equation 1

Where represents the time for propagation. Hence, the phase shift can be expressed in time and frequency as:

Equation 2

Replacing with in eq. 2, where is the speed of light and the distance between the devices, the phase shift with respect to distance is:

Equation 3

For a round trip distance , where is the signal wavelength, the round trip phase shift can be expressed as:

Equation 4

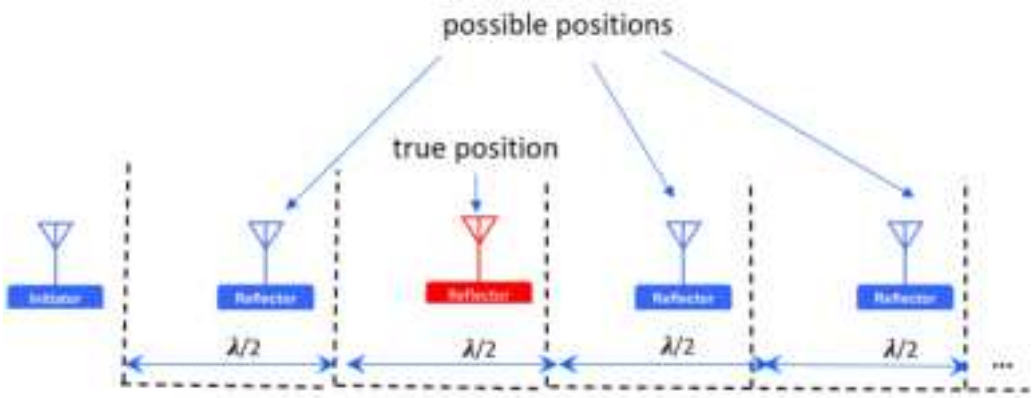
Phase measurement is circular, meaning that it wraps around 2 or after the wave propagates a total distance equal to the wavelength . For distance , the wrapped phase shift of the round trip can be expressed as:

Equation 5

Rearranging eq. 5, distance is:

Equation 6

Due to phase wrapping, round trip distance measurements are only unambiguous within half-wavelength distance. In practice, though, we want to measure longer distances than that.



Distance Ambiguity Due to Phase Wrapping

To unwrap the phase and estimate distance without ambiguity, the phase shift can be measured at two or more distinct tones [1].

Equation 7

Equation 8

Combining eq. 7 and 8, the spatial distance measured using the two tones can be represented as:

Equation 9

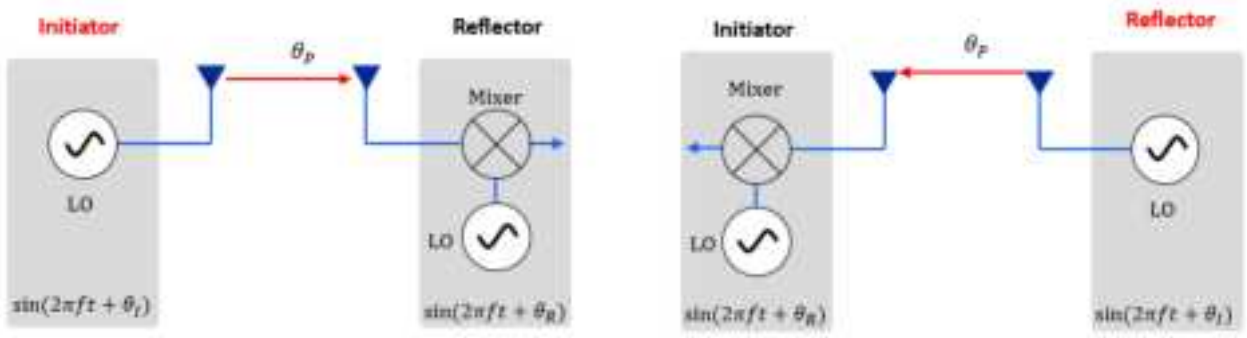
From eq. 9, the distance wrap or the maximum distance that can be measured using multiple tones PBR, can be determined as follows, where MHz is the minimum separation between two tones:

Equation 10

With a MHz separation, for instance, if only odd or even channels are used to optimize and save time, the maximum distance that can be measured without ambiguity is reduced by half, resulting in m.

Eliminating Local Oscillator Phase Offset

Note that the Reflector device does not have to lock the Local Oscillator (LO) to the incoming signal and transmit back to the Initiator. By keeping the LOs on between TX/RX, exchanging tones and doing two-way phase difference measurements, the unknown initial phases of the LOs can be canceled out, and the spatial phase rotation can be measured.



Local Oscillator Phase Offset

Let **Phase Correction Term (PCT)** be defined by the angle, if added to the internal angle of the LO, would result in a phase identical to that of the incoming signal. Then, the measured phase at the Reflector is:

Equation 11

Similarly, the measured phase at the Initiator side is:

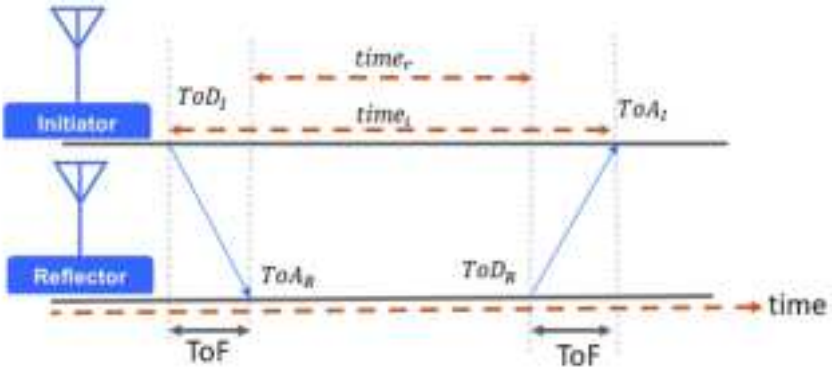
Equation 12

The round trip phase rotation is then:

Equation 13

Distance Estimation Based on Round-Trip Time (RTT)

The **Round-Trip Time (RTT)** approach allows distance estimation by measuring the round trip propagation delay. Given the same time base on both devices, sufficient precision of Time of Departure (ToD) and Time of Arrival (ToA) for the required distance accuracy, and under ideal line of sight conditions, then the distance can be calculated as shown in the equation below, where is the speed of light.



Round-Trip Time Based Ranging (RTT)

Equation 14

Equation 15

Where:

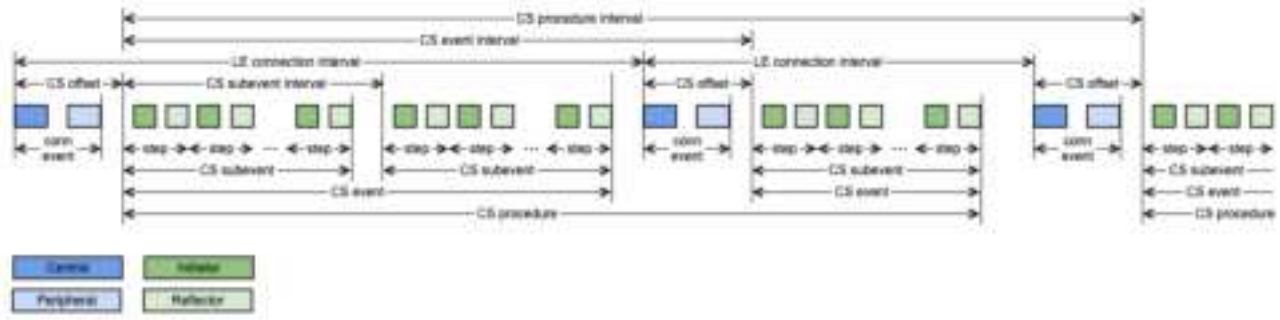
- is the round-trip time
- is speed of light
- is Time of Arrival at the Reflector
- is Time of Departure at the Initiator
- is Time of Arrival at the Initiator
- is Time of Departure at the Reflector

Bluetooth Channel Sounding Procedure

Bluetooth CS procedure consists of a series of radio operations in which the Initiator and Reflector devices exchange information that is used to either calibrate each other or measure distance between them.

For a CS procedure to start, the peer devices must first form an encrypted connection. Once an encrypted connection is in place, the CS is configured and security information is exchanged, after which the CS procedure can begin.

A CS procedure can be divided into one or more CS events. A CS procedure (and hence the first CS event in a CS procedure) is started *CS offset* from the timing of Asynchronous Connection-oriented Logical (ACL) connection event anchor point. A CS event may contain one or more subevents. A CS subevent on its turn may contain two or more CS steps. The figure below shows the relationship between CS procedures, CS events, CS subevents, and CS steps.

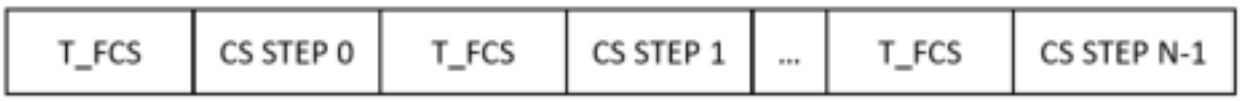


CS Procedure

It is important to note that the BLE connection roles of the devices are not strictly tied to CS roles, which allows both central and peripheral devices to assume either the Initiator or Reflector role. In the above figure, it is assumed that the central device is the Initiator and the peripheral device is the Reflector.

In a CS step, bilateral exchange between the peer devices occurs. In each step, the Initiator transmits first and the Reflector responds with one or more transmissions. These transmissions may be GFSK modulated packets, or ASK modulated tones, or both. After receiving the packets, tones, or both from the other peer device, the devices do time and/or phase measurements on their own. One of the devices sends back its measured data over a Bluetooth connection using the Ranging Service (RAS) GATT to the peer device, which then determines the actual distance estimate. Note, however, that the determination of distance measurements is outside the scope of the Bluetooth Channel Sounding specification.

CS steps require a precise timing synchronization between the Initiator and Reflector devices. Between the CS steps there is a time separation defined as *T_FCS* to allow frequency hopping. Refer to the core specification for the permitted values of *T_FCS*.



The spec defines four CS step mode types, each designated for a specific purpose as detailed below.

Mode-0 Steps

CS step mode-0 CS is used to measure and calibrate the frequency offset between the Initiator and Reflector at a given frequency.

Each subevent in a CS procedure begins with one to three mode-0 steps to provide calibration information for the remaining CS steps within that subevent. Based on the mode-0 step results, the Fractional Frequency Offset (FFO) is calculated which will be used by the Initiator device to align the timing of CS steps and transmit frequencies during non-mode 0 CS steps.

The structure of CS step mode-0 is depicted as follow.

Role/Duration	T_SY	T_RD	T_IP1	T_SY+T_GD+T_FM	T_RD
Initiator	CS_SYNC_0_I *	ramp down			
Reflector				CS_SYNC_0_R **	ramp down

CS step mode-0

* **CS_SYNC_0_I** is a CS SYNC packet sent from the Initiator to the Reflector during mode-0 steps.

** **CS_SYNC_0_R** is extended packet—CS SYNC followed by CS Tone sent by the Reflector to the Initiator during mode-0 steps.

For additional information on CS SYNC and CS extended packet formats, refer to Channel Sounding Packet Formats.

T_SY is the duration of the CS SYNC packet which depends on the PHY used.

T_RD is ramp-down time in which the transmitter is allowed to remove the transmitted energy from the RF channel. It's value is .

T_IP1 is the idle time between the end of transmission from the Initiator and the start of the transmission from the Reflector. Please refer to the core specification for the permitted values of *T_IP1*.

T_FM is the duration of the frequency measurement; and it's value is for step mode-0.

Mode-1 Steps

Mode-1 step is used to measure the round-trip time between the intiator and the Reflector. The structure of step mode-1 is depicted below.

Role/Duration	T_SY	T_RD	T_IP1	T_SY	T_RD
Initiator	CS_SYNC_1 *	ramp down			
Reflector				CS_SYNC_1	ramp down

CS step mode-1

* **CS_SYNC_1** is a CS SYNC packet exchanged between the Initiator and Reflector during step mode-1. For additional information on CS SYNC packet format, refer to Channel Sounding Packet Formats.

The duration of *T_SY* in step mode-1 depends on the PHY used and the use of the optional Sounding Sequence or Random Sequence bits described in Channel Sounding Packet Formats.

Mode-2 Steps

Mode-2 step is used to measure the phase rotation of the RF signal between the Initiator and the Reflector. The structure of mode-2 step is depicted as follow.

Role/Duration	(T_SW+T_PM)* (N_AP+1)	T_RD	T_IP2	(T_SW+T_PM)* (N_AP+1)	T_RD
Initiator	CS tone *	ramp down			
Reflector				CS tone	ramp down

CS step mode-2

* **CS tone** is Amplitude Shift Keying (ASK) modulated continuous carrier wave.

T_SW is antenna switching duration.

T_PM is the phase measurement period. Please refer to the core specification for the permitted values of *T_PM*.

T_IP2 is the idle time between the end of transmission from the Initiator and the transmission from the Reflector. Refer to the core specification for the permitted values of *T_IP2*.

N_AP is the number of antenna paths.

Mode-3 Steps

Mode-3 step is used to measure the phase rotation of the RF signal and the round-trip time between the Initiator and the Reflector. The structure of mode-3 step is depicted as follow.

Role/Duration	T_SY+T_GD+ (T_SW+T_PM)*(N_AP+1)	T_RD	T_IP2	(T_SW+T_PM)*(N_AP+1) +T_GD+T_SY	T_RD
Initiator	CS_SYNC_3_I*	ramp down			
Reflector				CS_SYNC_3_R*	ramp down

CS step mode-3

* **CS_SYNC_3_I** is extended packet—CS SYNC followed by CS Tone sent by the Initiator to the Reflector.

** **CS_SYNC_3_R** is extended packet—CS Tone followed by CS SYNC sent by the Reflector to the Initiator. For additional information on the extended packet format, see the next section, *Channel Sounding Packet Formats*.

Note that the specification requires the implementation of mode-0, mode-1, and mode-2 steps as mandatory. Mode-3, however, is optional.

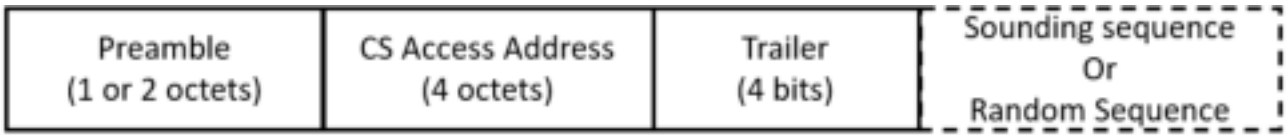
Channel Sounding Packet Format

In general, there are three packet formats defined in channel sounding:

- CS SYNC
- Extended packet—CS SYNC followed by CS Tone
- Extended packet—CS Tone followed by CS SYNC

CS SYNC Packet

CS SYNC is a specific modulated bit sequence defined for CS. The packet format of CS SYNC is similar to that of LE Uncoded PHY, except that CS SYNC packet has **no PDU, CRC or CTE fields**. The contents of CS SYNC packet format are shown in the following figure.



The duration of a CS SYNC packet with no Sounding Sequence or Random Sequence fields is when using 1M PHY, and when using 2M PHY.

The mode-0 CS SYNC packets are sent without the optional Random Sequence or Sounding Sequence.

Preamble

The same rules defined for the preamble of the LE uncoded PHY packet format applies to the preamble of CS SYNC packet. The length of the preamble is 1 octet when using 1M PHY, and 2 octets when using 2M PHY.

CS Access Address

CS Access Addresses are used for synchronization, security, and round-trip time purposes. Each CS Access Address is a sequence of bits that is cryptographically generated using a dedicated CS Deterministic Random Bit Generator (DRBG).

Trailer

The CS trailer is a sequence of 4 bits alternating between 0 and 1 bits. The trailer sequence is **1010** when the most significant bit of the Access Address is 0; otherwise, it is **0101**.

Sounding Sequence

Sounding Sequence is an optional feature that can be either 32 or 96 bits long. If present, one or more 4-bits long marker signals are added to the sounding sequence for resilience against spoofing attacks.

Random Sequence

Random Sequence is also an optional feature that can be 32, 64, 96, or 128 bits long.

Extended Packet—CS SYNC followed by CS Tone

This packet format has two mode-specific variations:

- CS_SYNC_0_R: transmitted by the Reflector in CS step mode-0.
- CS_SYNC_3_I: transmitted by the Initiator in CS step mode-3.

Extended Packet—CS Tone followed by CS SYNC

The occurrence of this packet format is during CS_SYNC_3_R, i.e packet transmission from Reflector to Initiator in mode-3.

In both extended packet formats, the CS tone and CS SYNC are transmitted on thcs e same RF frequency.

In both extended packet formats, the duration of the guard time *T_GD* is . However, the duration of the full extended packet formats is variable and depends on several factors. Please refer to the core specification for more details.

Channel Sounding Security

CS has included various security features to detect and reduce the probability of distance spoofing. The table below summarizes the different types of vulnerabilities associated with proximity-based services and the corresponding mitigation approaches that CS provides.

Vulnerability	Description
Impersonation attack: an attacker attempts to break the authentication by impersonating the legitimate transceivers.	This kind of attacks can be prevented using good cryptography and a state-of-the-art authentication/signature protocols. In CS, data exchange between Initiator and Reflector devices occurs on an encrypted connection. CS uses DRBG to randomize CS Access Address, channel list order, sounding sequence marker and marker position in CS SYNC packets, tone extension slot present information, etc. A CS device returns CS Access Address quality indication value for CS_SYNC packet received in a CS step. This information can be used for security measures.
Range extender attacks: an attacker amplifies the signal without altering the phase or frequency, making the devices appear closer to each other than they actually are.	PBR based measurement cannot prevent this kind of attack since an attacker can amplify or relay the signal beyond the ambiguity range. However, ToF measurement (i.e RTT approach) can prevent such attacks since time does not roll over. CS allows for the use of RTT and PBR ranging in the same subevent, or in the same step (i.e mode-3) to obtain two partially independent distance estimates. The discrepancy between the RTT and PBR measurement results can be used by the application for security measures.
Phase manipulation attack: an adversary manipulates the phase of the constant tone signal to reduce the estimated distance.	This kind of attack only affects phase and not the time. Thus, RTT can be used with PBR to provide additional security layer to detect this kind of attacks.
Early Detect Late Commit (EDLC): an attacker aims at decreasing the ToF, for instance, by detecting a bit in a symbol and predicting the symbol even before receiving it completely.	This kind of attack affects GFSK modulated packets, i.e mode-0, mode-1 and mode-3. As such, PBR can offer protection against such attacks within the ambiguity range. CS step mode-1 and mode-3 allow sending CS SYNC packets with an optional Sounding sequences or Random Sequence, which can provide capabilities to potentially detect whether an attacker is present. Using Random sequences, it is possible to measure how much a received GFSK modulated packet signal differs from the expected packet signal. Using Sounding sequence, it is possible to detect the position of one or more marker signals. In Addition, CS introduces **Normalized Attack Detector Metric (NADM)**, which provides a measure of how much a received GFSK modulated packet signal differs from the expected packet. Please refer to the core specification to learn more about NADM values and their corresponding indications for the likelihood of attacks.

References

[1] Zand, Pouria, et al. "A high-accuracy phase-based ranging solution with Bluetooth Low Energy (BLE)." 2019 IEEE wireless communications and networking conference (WCNC). IEEE, 2019.

[2]. Bluetooth Channel Sounding Draft Specification: https://www.bluetooth.com/specifications/specs/channel-sounding-cr-pr/.

[3]. White paper: Distance Bounding Protocol for Bluetooth Secure Access: https://www.imec-int.com/en/expertise/sensors-for-iot/secure-system-solutions/distance-bounding-white-paper.

Getting Started

You are viewing documentation for version: 0.1 | Version History

Getting Started with Silicon Labs Bluetooth Channel Sounding

Overview

Bluetooth version 6.0 introduces support for Channel Sounding (CS), a feature that enables precise distance estimation between two Bluetooth Low Energy (BLE) devices through time and phase-based measurements. Silicon Labs not only integrates this functionality into its Bluetooth stack but also offers a reference implementation of a Real-Time Locationing (RTL) library, which facilitates distance estimation and additional capabilities.

In CS, two Bluetooth devices, known as Initiator and Reflector, exchange information that is measured to produce a distance estimate between them. Silicon Labs offers comprehensive hardware and software solutions designed to streamline and simplify the evaluation of Bluetooth CS Initiator and Reflector roles.

Hardware Requirements

To run the demos, you need two setups: one for the Initiator and one for the Reflector. Each setup requires an EFR32xG24 SoC equipped with a 40 MHz crystal.

Currently, Silicon Labs offers BRD4198A, which is an EFR32MG24-based single dipole antenna radio board designed for CS.

When using BRD4198A, both reflector and initiator devices should have:

- 1x BRD4001 or BRD4002 WSTK
- 1x BRD4198A with 1x Sleeve dipole antenna and 1x antenna adapter plug

Software Requirements

To begin exploring Silicon Labs CS demos, install Simplicity Studio 5 (SSv5) along with Simplicity SDK (SiSDK) version 2024.6.1 or newer.

Bluetooth Channel Sounding Demos

SiSDK v2024.6.1 provides the following binaries that demonstrate Bluetooth CS Initiator and Reflector devices.

- **Bluetooth – SoC CS Reflector**: Demo app is a prebuilt binary version of the SoC Reflector sample project found in:

```
<simplicity_sdk>
└──app
|  └──bluetooth
|     └──example
|        └──bt_cs_soc_reflector
```

- **Bluetooth – SoC CS Initiator**: Demo app is a prebuilt binary version of the SoC Initiator sample project found in:

```
<simplicity_sdk>
└──app
|  └──bluetooth
|     └──example
|        └──bt_cs_soc_initiator
```

- **Bluetooth - NCP CS**: Demo app is a prebuilt binary version of the CS NCP target sample project found in:

```
<simplicity_sdk>
└──app
|  └──bluetooth
|     └──example
|        └──bt_cs_ncp
```

The Bluetooth - SoC CS Reflector and Initiator demos are standalone. The demo binaries run on the device and display the results directly on the devices' screen and via serial console.

The NCP demo can be used with SSv5's Channel Sounding Analyzer (CS Analyzer) tool.

Setting up the Boards

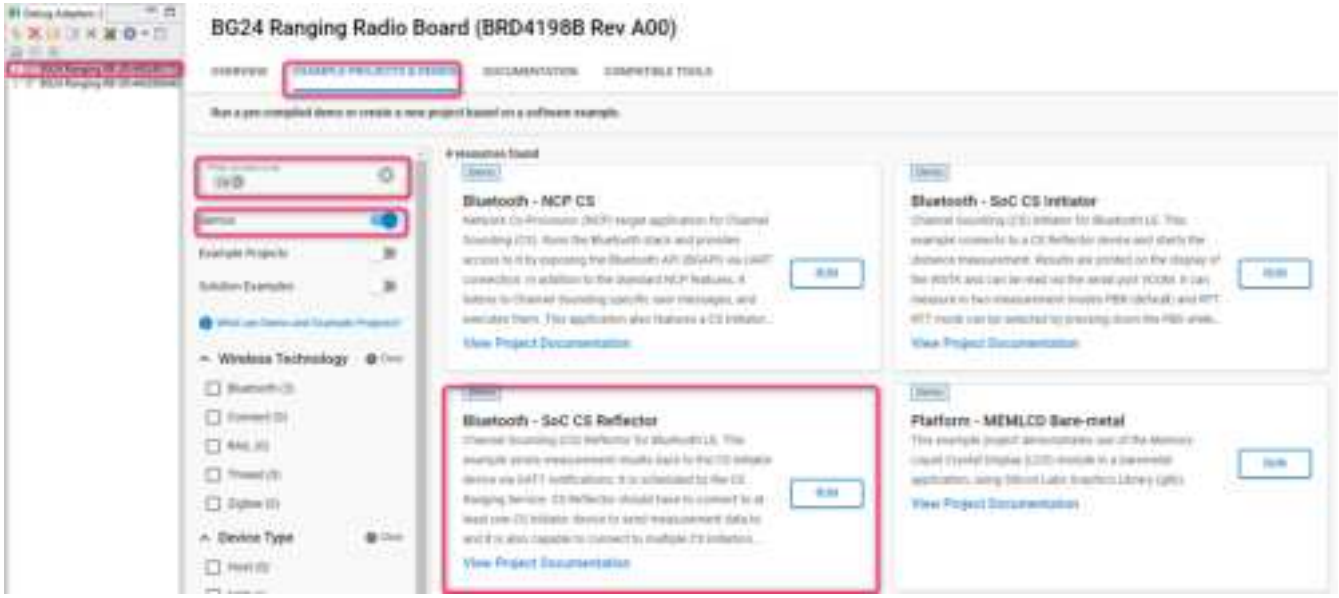
To setup the demos:

1. Ensure that the development kits are connected to a computer that has both SSv5 and SiSDK installed.
2. Ensure that the switch on the main board is set to **AEM**.

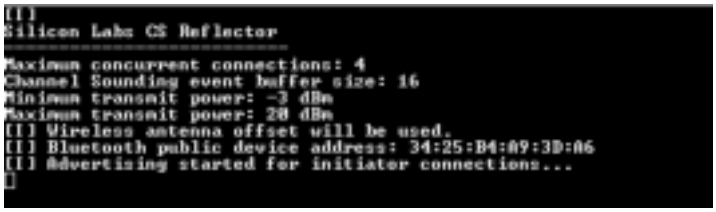
3. Start SSv5.

Running the Bluetooth – SoC CS Reflector Demo

1. In SSv5 **Launcher** perspective, select your device from the **Debug Adapters** list, and on the **EXAMPLE PROJECTS & DEMOS** tab, select the **Bluetooth - SoC CS Reflector** demo.
2. Click **RUN** to download and run the demo on your board. Note: the demo also includes a bootloader. If you build the CS Reflector application from source, a bootloader must also be flashed to the device. You can find more information about the bootloaders in UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher.

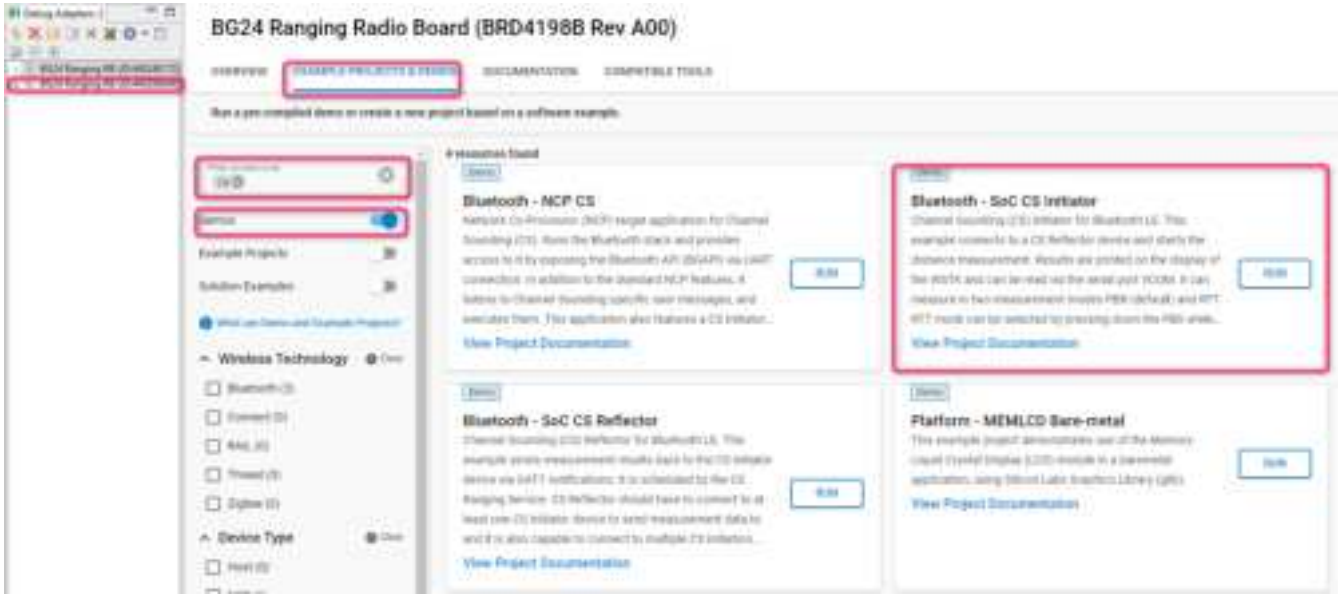


3. Open the serial monitor, and you will see a message indicating that your device has started advertising. The Reflector advertises with a device name **CS RFLCT** which will be used by the initiator to identify it and initiate a connection.



Running the Bluetooth – SoC CS Initiator Demo

1. In SSv5 **Launcher** perspective, select your device from the **Debug Adapters** list, and on the **EXAMPLE PROJECTS & DEMOS** tab, select the **Bluetooth - SoC CS Initiator** demo.
2. Click **RUN** to download and run the demo on your board. Note: the demo also includes a bootloader. If you build the CS Initiator application from source, a bootloader must also be flashed to the device. You can find more information about the bootloaders in UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher.



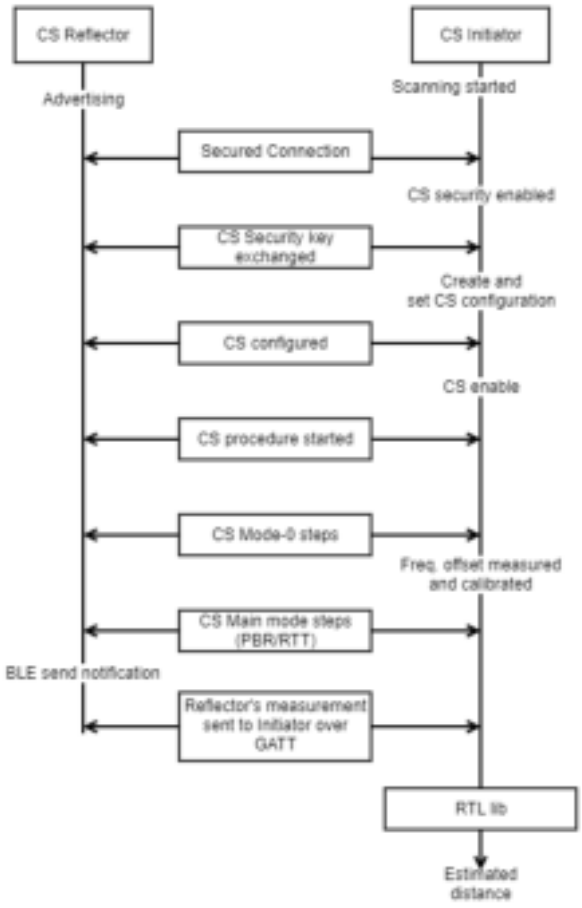
3. After flashing the firmware, the device starts scanning for a Reflector device, connects to it, and starts the CS measurements instantly. These are indicated on the LCD display.

4. By default, the Initiator uses the Phase-based Ranging (PBR) mode. Round-Trip Time (RTT) mode can be selected by pressing Button 0 while resetting the device. Refer to Channel Sounding Fundamentals to learn more about PBR and RTT based measurement methods.
5. Open the serial monitor, and you will see the logs looking like this:

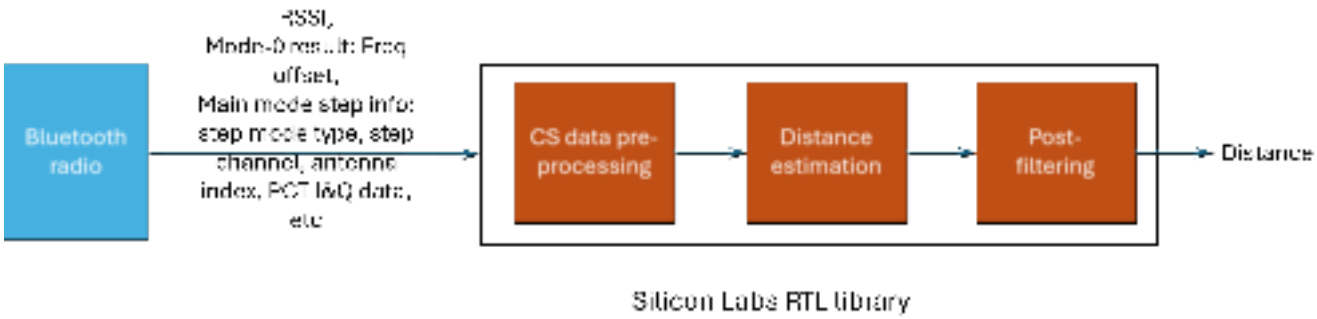
```
[I]
[I] +-[CS initiator by Silicon Labs]-------------------------+
[I] ! Press PB0 while reset to select RTT measurement mode! !
[I] +-------------------------------------------------------+
[I] Wireless antenna offset will be used.
[I] Minimum CS procedure interval: 30
[I] Maximum CS procedure interval: 30
[I] +-[ CS channel map ]-------------------------------------+
[I] 0xfc 0xff 0x7f 0xfc 0xff 0xff 0xff 0xff 0xff 0x1f
[I] +-------------------------------------------------------+
[I] RSSI will be measured.
[I] PBR measurement mode selected.
[I] +-------------------------------------------------------+
[I] peer manager: started
[I] Bluetooth public device address: 34:25:B4:A9:3D:B9
[I] peer manager: scanning for device name 'CS RFLCT' ...
[I] peer manager: connection opened: #1.
[I] [initiator] #?: create new initiator instance
[I] [initiator] #1: Tracking mode: moving objects
[I] [initiator] #1: CS channel map - channel count: 72
[I] [initiator] #1: RTL - set CS mode: PBR
[I] [initiator] #1: RTL - CS parameters set.
[I] [initiator] #1: RTL - create estimator
[I] [initiator] #1: RTL - estimator created.
```

In general, the BLE central and peripheral roles are loosely tied with the CS Reflector and Initiator roles. The CS procedure can also be started by either the Reflector or Initiator device. The transaction for the actual measurement information is always started by the Initiator device.

In this demo implementation, however, the Initiator plays the BLE central role, and is also responsible for starting the CS procedure. The figure below depicts a simple flow diagram for the communication between the Reflector and Initiator devices.



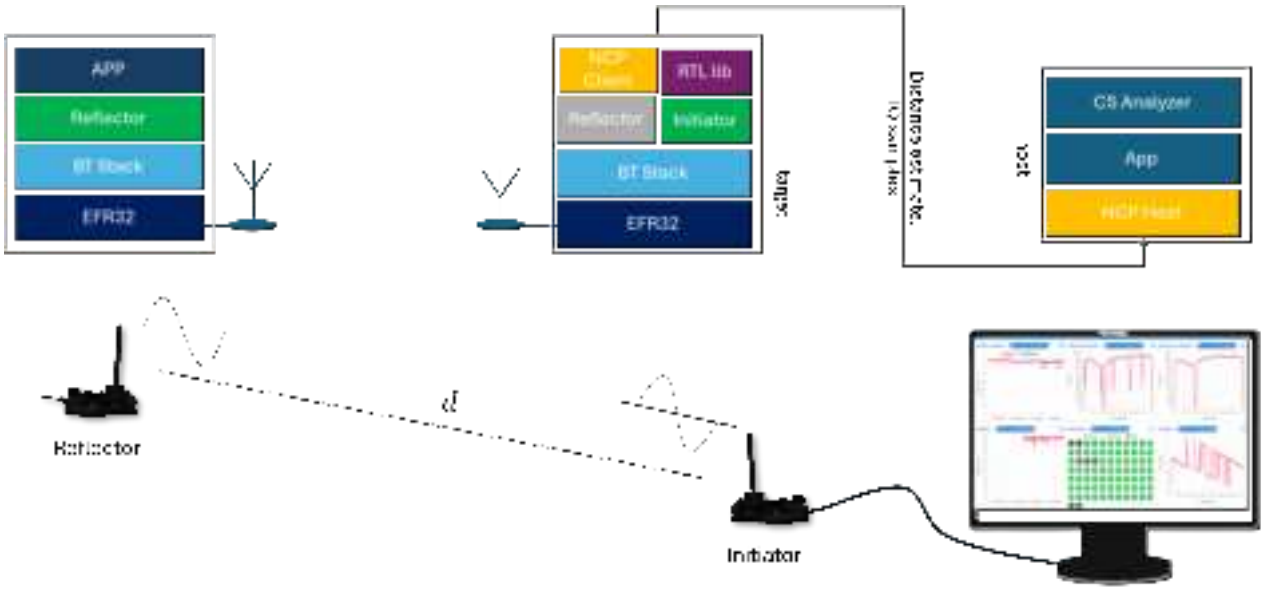
The phase and/or time measurements conducted by the Reflector and Initiator are fed to the RTL lib, which then calculates the actual distance estimation. The RTL lib takes the raw data from the results of the CS procedure, performs preprocessing, calculates an estimated distance, and then applies post-processing to yield a refined result.



Running the Bluetooth NCP Channel Sounding Demo

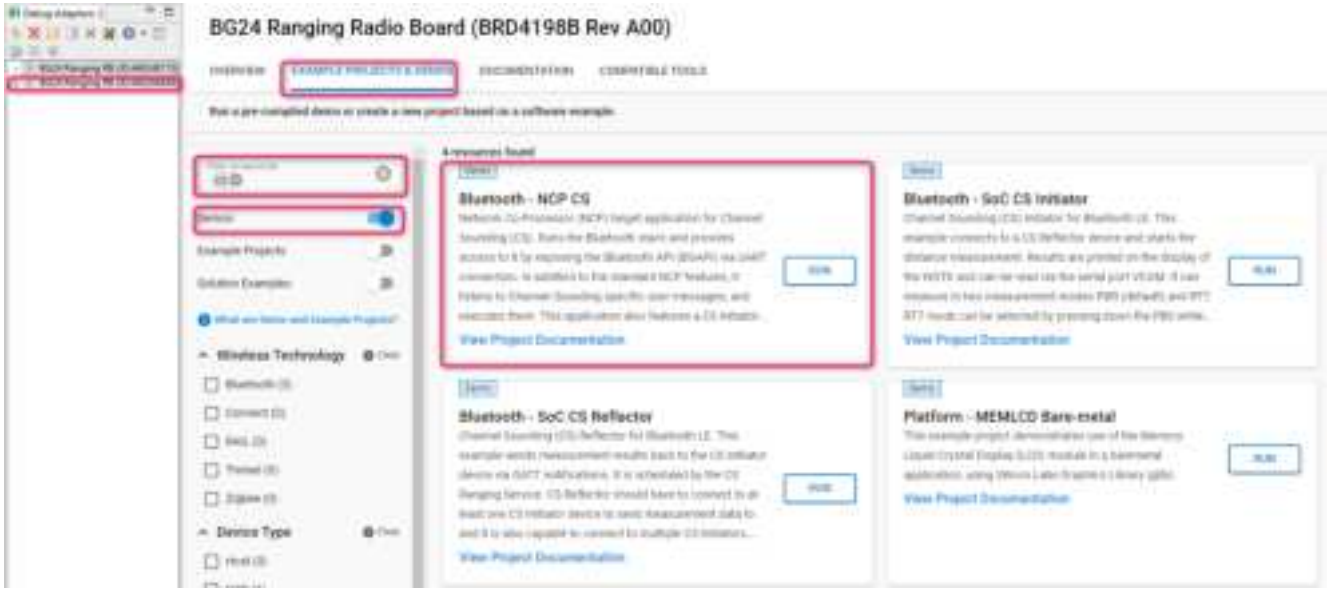
The NCP demo requires running the **Bluetooth - NCP CS** firmware on the Initiator device and the **Bluetooth - SoC CS Reflector** firmware on the Reflector device. In addition, the **CS Analyzer** is used for visualizing the measurements received by the Initiator.

The CS Analyzer tool is a Java-based application integrated into SSv5, used to visualize and debug the Silicon Labs CS solution. It is intended for demo and evaluation purposes and follows the following simple architecture.



To run the demo:

1. In SSv5 **Launcher** perspective, select your Initiator device from the **Debug Adapters** list, and on the **EXAMPLE PROJECTS & DEMOS** tab, select the **Bluetooth - NCP CS** demo.
2. Click **RUN** to download and run the demo on your board. Note: the demo also includes a bootloader. If you build the NCP CS application from source, a bootloader must also be flashed to the device. You can find more information about the bootloaders in [UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher] (https://www.silabs.com/documents/public/user-guides/ug489-gecko-bootloader-user-guide-gsdk-4.pdf).



Now, you can start the CS Analyzer.

1. In SSv5 **Launcher** perspective, go to **Tools**, and in the Tools Dialog wizard, select **Channel Sounding Analyzer**.

2. Click **OK**.
   The CS Analyzer will open in a new tab.
3. Next:
   - Select your Initiator device by its J-Link address and connect to it (**A**).
   - It will automatically start scanning the Reflector device. Select your Reflector device by its Bluetooth address and connect to it (**B**).
   - When prompted, select the CS mode (PBR or RTT) (**C**), and Enable/Disable Moving Object Tracking (**D**). By default, PBR is enabled as the CS measurement method, and moving object tracking is selected.
   - Click **OK** and a time chart displaying the distance estimate will open.

- By default, the chart visualizes CS distance estimates as well as RSSI based distance. You can disable RSSI based distance by clicking the wireless network icon (**F**).

Once you have set up development with your platform of choice and experimented with the precompiled demos, you can begin customizing example applications to suit your specific needs. Refer to the Developer's Guide to learn more about the demo projects' software architecture and the features of the CS Analyzer.

Developer's Guide

You are viewing documentation for version: 0.1 | Version History

Bluetooth LE Channel Sounding Developer's Guide

Channel Sounding (CS) is a new feature introduced as part of the Bluetooth core specification 6.0 to enable distance estimation between two Bluetooth Low Energy (LE) devices.

In CS, two devices, known as Initiator and Reflector, exchange information that is measured to provide timing and frequency synchronization, and produce accurate distance estimate between them. It achieves this using Phase Based Ranging (PBR) and Round-Trip Time (RTT) measurement approaches. You can find more information in the Fundamentals section.

Silicon Labs provides modularized software sample projects for the Initiator and Reflector that can be extended to address different use case scenarios. In addition, Silicon Labs offers a reference implementation of a Real-Time Locating (RTL) library, which facilitates distance estimation using CS.

The rest of this Developer's Guide is organized as follows:

Prerequisites: Provides all prerequisites needed to build, program, and use the CS sample projects.

Documentation: Describes where the CS documentations and API reference manual can be found.

Supported Boards: Describes Silicon Labs radio board offering with support for Bluetooth channel sounding.

Sample Applications: Describes the software architecture, configuration, building and running procedures of the provided sample projects.

Simplicity Studio Channel Sounding Analyzer: Describes how to use the Simplicity Studio Channel Sounding Analyzer tool for visualizing and debugging purposes.

Channel Sounding Performance Metrics: Describes how to use the Simplicity Studio Channel Sounding Analyzer tool for visualizing and debugging purposes.

Known Issues and Limitations: Lists the issues and limitations known at publication.

Calibration of Silicon Labs Distance Ranging: Discusses the theory and procedure of calibrating a Silicon Labs channel sounding SoC for accurate distance ranging.

Prerequisites

You are viewing documentation for version: 0.1 | Version History

Prerequisites

To get started with CS application development, you need the following:

- A computer with the latest Simplicity Studio v5.
- Simplicity SDK (SiSDK) v2024.6.1, or later.
- Two EFR32xG24-based devices equipped with a 40 MHz crystal. Refer to Supported Boards to learn about the radio boards that Silicon Labs offers for Bluetooth CS.
- A wireless Starter Kit (WSTK). The CS sample projects use the pushbutton and LCD on the WSTK to change mode and display CS results, respectively.

Documentation

You are viewing documentation for version: 0.1 | Version History

Documentation

- For background on Channel Sounding, see Fundamentals.
- For basic information on getting started, see Getting Started.
- Full API documentation can be found at docs.silabs.com, or within Simplicity Studio v5. Select your device from the Debug Adapter window, browse to the **DOCUMENTATION** tab in the Launcher perspective, and search for *Silicon Labs Bluetooth API Reference Guide*.
- The Bluetooth CS specific APIs can be found at https://docs.silabs.com/bluetooth/latest/bluetooth-stack-api/sl-bt-cs.
- The RTL CS specific APIs can be found at https://docs.silabs.com/rtl-lib/latest/rtl-lib-api/sl-rtl-cs.

Supported Boards

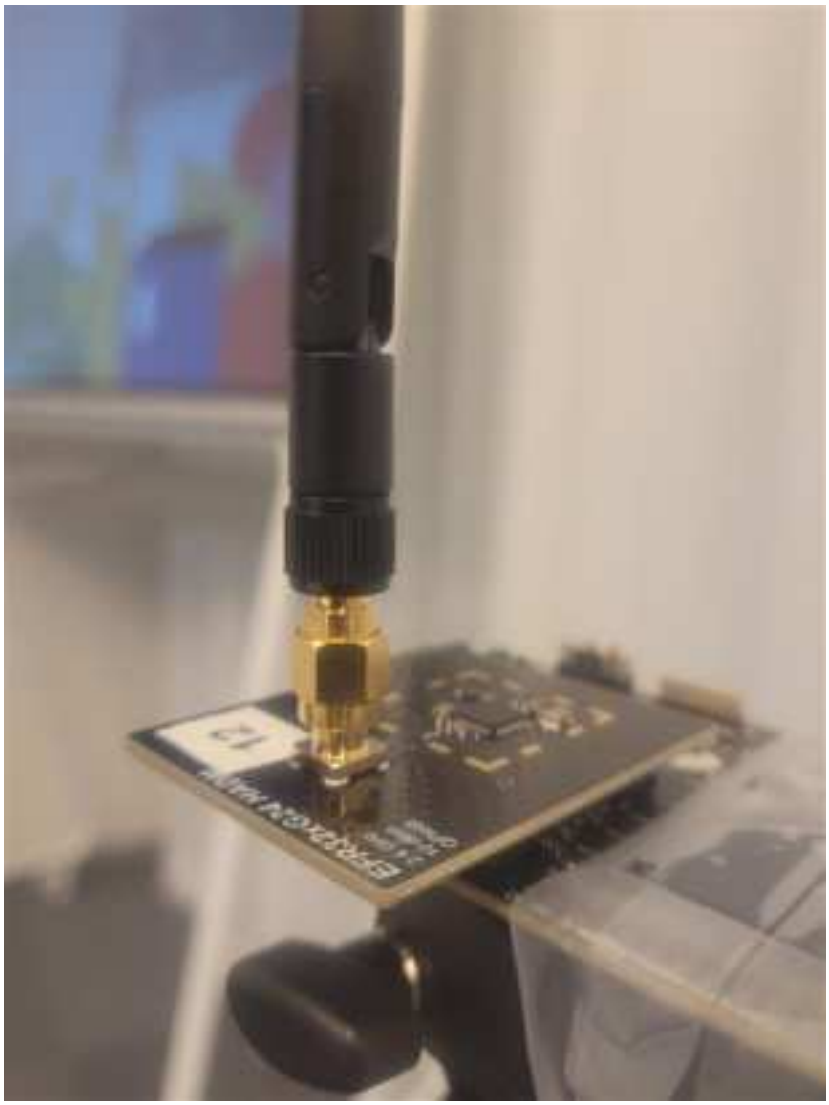
You are viewing documentation for version: 0.1 | Version History

Supported Boards

The provided sample applications are designed to run on the EFR32MG24 SoC, equipped with a 40 MHz crystal. Currently, Silicon Labs offers BRD4198A with Bluetooth CS support.

Two development setups are required for the measurement: one for the initiator and one for the reflector. Each setup contains the following:

- 1x BRD4001A or BRD4002A
- 1 x BRD4198A with 1x Sleeve dipole antenna and 1x antenna adapter plug



When assembling the BRD4198A Kit, make sure to use the provided antenna adapter plugs.

Sample Applications

Sample Applications

The sample setup consists of two types of applications, the Initiator and the Reflector. The setup uses the two-way ranging scenario. This is a connection-based measurement method where the Initiator scans for the Reflector and connects to it. After a connection is established, both the Initiator and Reflector send signals and packets to measure phase and time. The Reflector sends the measured phase and time back to the Initiator using GATT service and characteristics. The Initiator then combines all the information to calculate distance.

In general, the Simplicity SDK (SiSDK) provides sample projects for the **CS Initiator** and **CS Reflector** devices that can be run in two modes of operation—SoC Mode and NCP Mode.

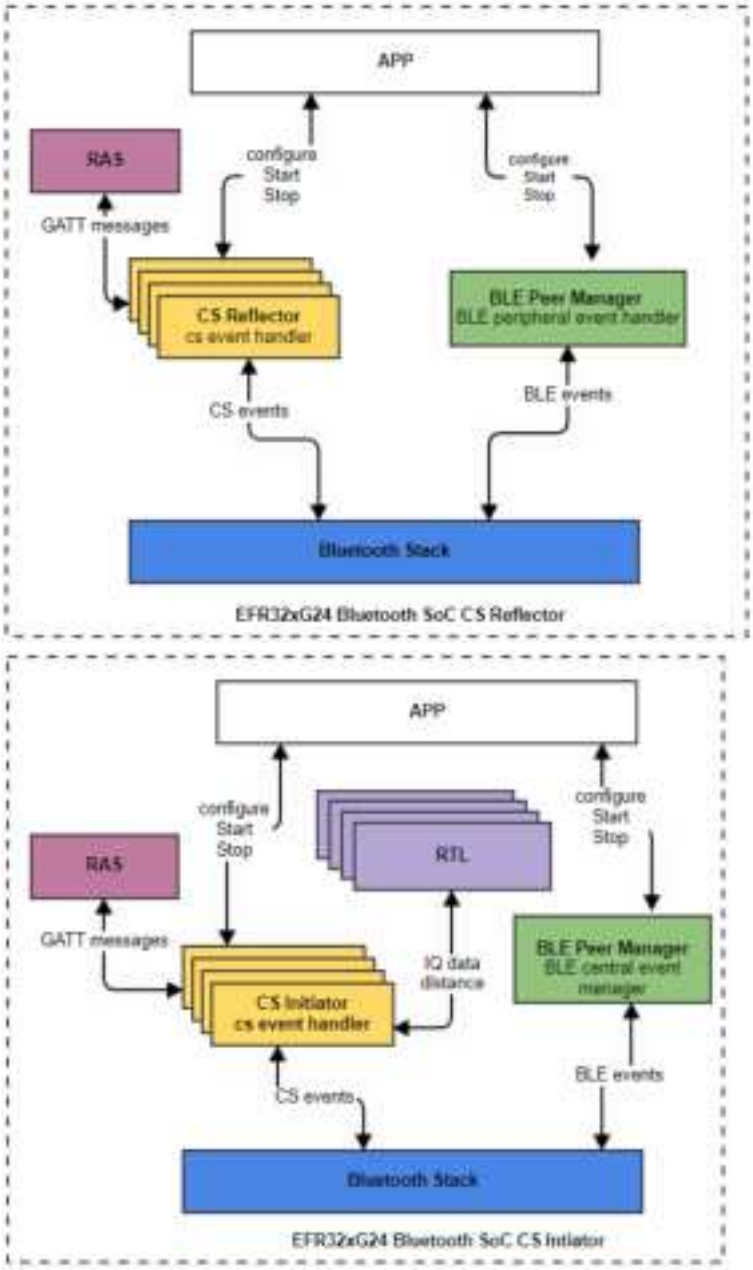
The sample projects for EFR32xG24 device can be found in SSv5.

SoC Mode Sample Applications

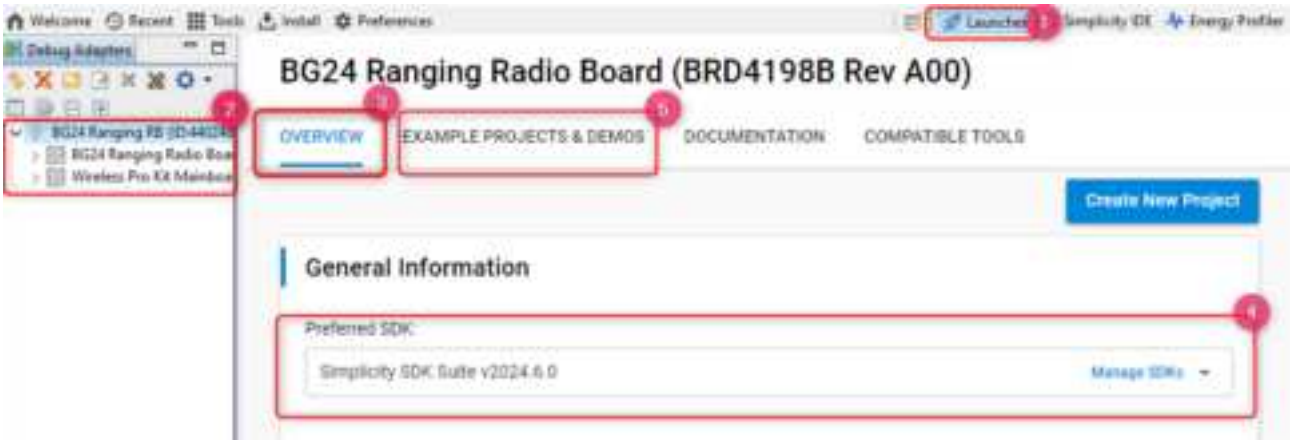
In SoC mode, the Bluetooth stack and the application (including the RTL library) run on the SoC/EFR32xG24 device. SiSDK 2024.6.0 provides the following two Studio sample projects supporting this mode of operation.

- bt_cs_soc_initiator: **Bluetooth – SoC CS Reflector**
- bt_cs_soc_reflector: **Bluetooth – SoC CS Initiator**

The applications consist of software components aimed at handling CS and general BLE activities and interfacing with the RTL library to yield distance estimation. The diagram below depicts a high-level software architecture of the SoC CS Initiator and Reflector sample applications.

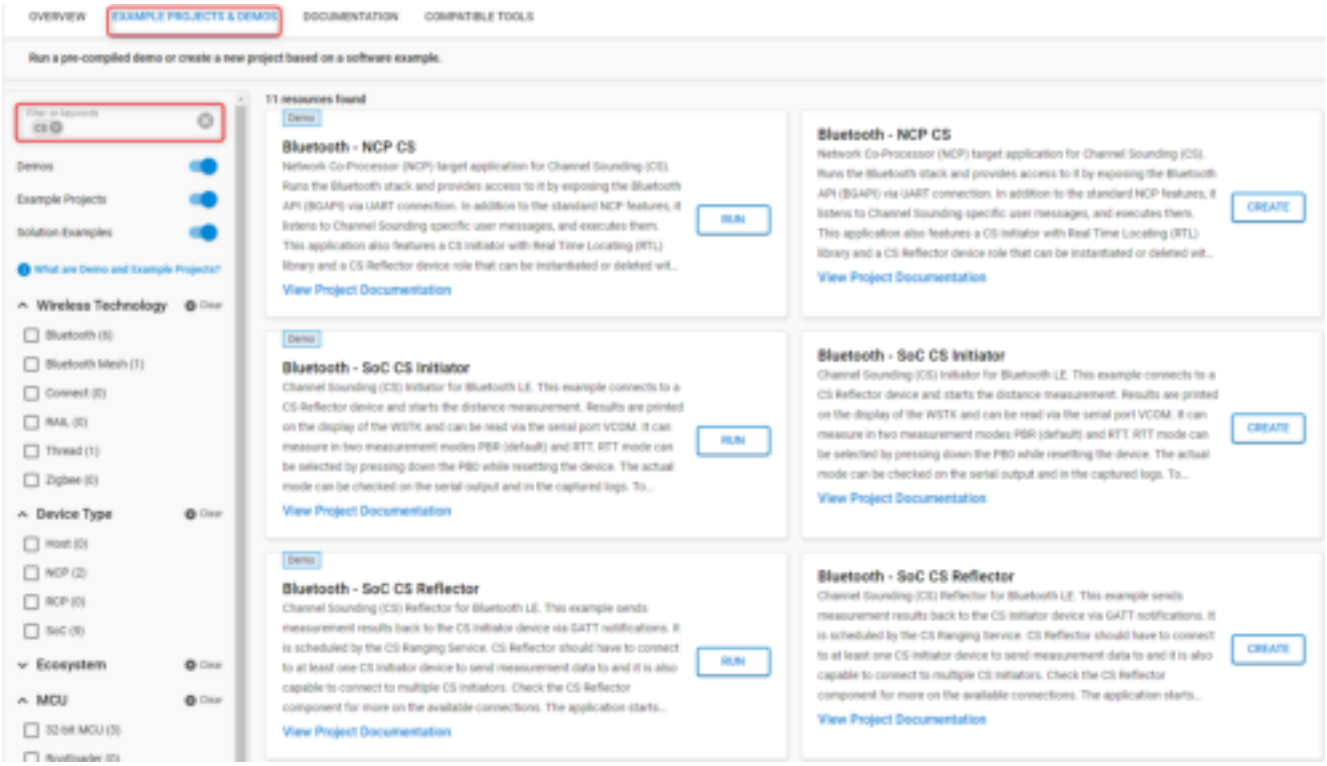


To access the projects in SSv5, in the Launcher perspective (1), select your device from the **Adapter Debug** window (2). In the **Overview** tab (3), choose the latest Simplicity SDK Suite, and then go to the **EXAMPLE PROJECTS & DEMOS** tab (5).



Once you are in the **EXAMPLE PROJECTS & DEMOS** tab, use the *CS* filtering keyword to see the sample applications. Click **CREATE** to create the project.

All the ranging applications running on the BRD4198A radio boards are also available as pre-built demos. For quick testing, you can run the demos by clicking **RUN**. This will flash the required bootloader and the corresponding sample application to your device.



Bluetooth – SoC CS Reflector

The main software components of the Bluetooth SoC CS Reflector are:

- BLE Peer Manager
- CS Reflector
- CS RAS

Project Walkthrough

The project contains the C related files under the folder *<SDK_Folder>/app/bluetooth/common/*.

The BLE Peer Manager is responsible for implementing the BLE peripheral roles of the Reflector device. These include configuring, starting, and stopping advertising. It accomplishes these tasks through the Bluetooth stack API calls: `sl_bt_advertiser_set_timing()` , `sl_bt sl_bt_legacy_advertising_start()` , and `sl_bt_advertiser_stop()` , respectively. Before advertising is started, the advertising set is created, and the advertising data is generated using `sl_bt_advertiser_create_set()` and `sl_bt_legacy_advertiser_generate_data()` , respectively. These are done in the *ble_peer_manager_peripheral.c* file.

The CS Reflector component is responsible for setting the maximum CS transmission power and enabling the CS Reflector role on the device for a specific connection by calling the `sl_bt_cs_set_default_settings()` API. This component also handles CS related events: `evt_cs_security_enable_complete` , which indicates that the CS Security Enable procedure has completed, `evt_cs_procedure_enable_complete` , which indicates the start of the CS procedure by the Initiator, and `evt_cs_result` , which is triggered by the local controller when a new CS result is available. These are done in the *cs_reflector.c* file.

The CS RAS component is responsible for parsing RAS GATT messages and creating RAS GATT responses.

Usage

Select the **Bluetooth - SoC CS Reflector** sample application, and build and flash it to a device. This sample application requires the *bootloader-apploader* type of bootloader.

After starting up, it should be advertising with the name **CS RFLCT**. By default, the Reflector allows up to 4 connections. This value can be changed in the **Connection** software component. In multiple connection use cases, it is advised to use a longer measurement interval.

Bluetooth – SoC CS Initiator

The main software components of the Bluetooth SoC CS Initiator are:

- BLE Peer Manager
- CS Initiator
- RTL library
- CS RAS

Project Walkthrough

The project contains the C related files under the folder *<SDK_Folder>/app/bluetooth/common/*.

The BLE manager is responsible for implementing the BLE central roles of the Initiator device. This includes configuring the scanning parameters, starting/stopping the scanner, and initiating/closing a connection with a Reflector device. It achieves these by calling the Bluetooth stack APIs: `sl_bt_scanner_set_parameters()` , `sl_bt_scanner_start()/sl_bt_scanner_stop()` , `sl_bt_connection_open()/sl_bt_connection_close()` , respectively.

The CS Initiator component is responsible for CS related activities. These include, setting the default parameters for CS transmission power and device role using `sl_bt_cs_set_default_settings()` , and ensuring that the requirements for starting a CS procedure are met.

Before a CS procedure is started:

- The connection between the peer devices must be encrypted. This is done by calling `sl_bt_sm_increase_security()` .
- The CS configuration is created and configured using `sl_bt_cs_create_config()` and `sl_bt_cs_set_procedure_parameters ()` API calls, respectively.
- The CS security information is exchanged by calling `sl_bt_cs_security_enable()` . The Deterministic Random Bit Generator (DRBG) is seeded from security information exchanged at this step.

After these steps, the CS procedure is started using `sl_bt_cs_procedure_enable()` API call.

The CS Initiator is also responsible for handling CS related events triggered by the local controller and providing the CS procedure data to the RTL library, which then calculates the distance. These are done inside *cs_initiator.c* file.

The CS RAS component is responsible for parsing RAS GATT messages and creating RAS GATT responses.

The RTL library is responsible for doing the actual distance estimation.

Usage

Select the **Bluetooth - SoC CS Initiator** application, and build and flash it to the device. This sample application requires the *bootloader-apploader* type of bootloader.

The Initiator will display the measured distance on the LCD mounted on the main board. The LCD display will also show the Bit Error Rate (BER), the likelihood of the measured value, and RSSI based distance estimation. Note that the BER is applicable in RTT mode only.

The device will also send the distances via UART. Open a console application on the PC to see the logs:

```
[I] [Initiator] [1] extract - reflector data
[I] [Initiator] [1] extract @ 1566 ms (51342 ticks)
[I] [Initiator] ----
[I] [Initiator] [1] extract - reflector data
[I] [Initiator] [1] extract @ 1586 ms (51990 ticks)
[I] [Initiator] [1] extract - device ready.
[I] [Initiator] ----
[I] [Initiator] [1] RTL - process CS results
[I] [Initiator] [1] RTL - get distance
[I] [Initiator] [1] RTL - get distance likeliness
[I] [Initiator] [1] RTL - estimation valid.
[I] [Initiator] [1] subevent data reset executed.
[I] [Initiator] [1] Instance new state: START_PROCEDURE
[I] [Initiator] [1] Instance new state: WAIT_PROCEDURE_ENABLE_COMPLETE
[I] +-----------------------------------------------------+
[I] [APP] [1] [Measurement 0001]
[I] [APP] [1] Measurement result: 2863 mm
[I] [APP] [1] Measurement likeliness: 0.70
[I] [APP] [1] RSSI distance: 17700 mm
[I] [APP] [1] CS bit error rate: 0
[I] +-----------------------------------------------------+

[RTL] {"rtl_instance_id": 0, "cs_procedure": {"cs_config": {"num_antenna_paths":
 1, "channel_map_repetition": 1, "main_mode_repetition": 0, "num_calib_steps": 3
```

Exporting Measurements to a File

The Bluetooth—SoC CS Initiator project contains a python script `log_filter.py` designed to log CS measurement, RTL and/or application logs to a file. It reads data from the serial port of the Initiator device and writes the data to a JSONL and txt files.

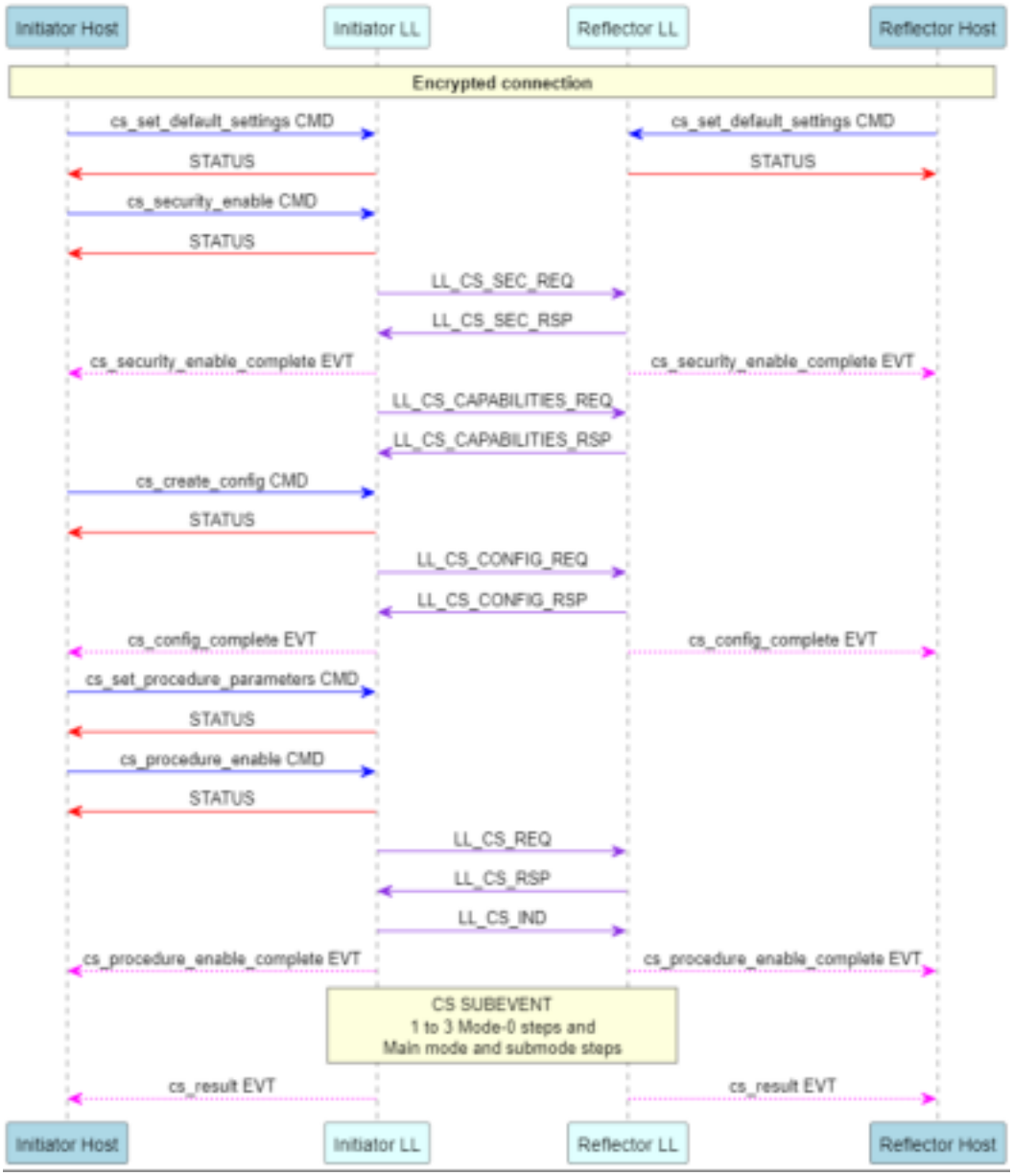
Run the application using the following command:

`python log_filter.py [-h] [-o {all,nortl,rtlonly}] [source]`

Where `[source]` is the serial or TCP address of the Initiator device.

Channel Sounding Sequence Chart

The following sequence diagram illustrates the message exchange between the Initiator and Reflector devices, based on the implementation of our sample applications. The diagram shows the BGAPI commands and events between the controller/LL and host of both devices, and the actual PHY PDUs exchanged between the Initiator and Reflector.



Configuration Options

Several configuration options can be modified to meet the requirements of different use cases.

Changing the CS Mode

By default, the Initiator uses the PBR mode. RTT mode can be selected by pressing Button 0 on the WSTK at startup. Press **BTN0** on the WSTK while resetting the device to select RTT mode.

Alternatively, this setting can be changed in the `config/initiator_app_config.h`. Set **MEASUREMENT_MODE** to `sl_bt_cs_mode_rtt` for RTT, and `sl_bt_cs_mode_pbr` for PBR to explicitly define the mode at compile time.

```
// <i> Specify measurement mode.
// <sl_bt_cs_mode_rtt=> Round Trip Time
// <sl_bt_cs_mode_pbr=> Phase Based Ranging
// <i> Default: sl_bt_cs_mode_pbr
#define MEASUREMENT_MODE    sl_bt_cs_mode_pbr
```

Changing CS Procedure Parameter—CS Procedure Interval

By default, the interval between CS measurement procedures is set to 30 for a stable setup even with multiple connections. However, if only one connection is used, this interval can be reduced to increase the refresh rate. This setting can be changed in the `config/cs_initiator_config.h`.

Care should be taken when adjusting the CS interval as it is defined in number of connection events between consecutive measurement procedures.

Note that the max and min CS intervals are ignored if the maximum procedure count is set to 1.

```
#ifndef CS_INITIATOR_MIN_INTERVAL
#define CS_INITIATOR_MIN_INTERVAL                (30)
#endif

#ifndef CS_INITIATOR_MAX_INTERVAL
#define CS_INITIATOR_MAX_INTERVAL                (30)
#endif
```

Changing CS Procedure Parameter—Maximum Procedure Count

This defines the maximum number of CS procedure to be scheduled. If set to 0, the CS procedures continue until it is stopped. This parameter can be set in the config struct found in: *<SDK_Folder>/app/bluetooth/common/sc_initiator/src/cs_initiator_configurator.c*.

```
// Value: 0. Free running
// Value: 1. Start new procedure after one finished.
config->max_procedure_count =        1;
```

Changing CS Maximum TX Power

The reference board supports a maximum TX power of 10 dBm. This value can be changed in the config struct found in: *<SDK_Folder>/app/bluetooth/common/sc_initiator/src/cs_initiator_configurator.c*.

```
// CS power
config->max_tx_power_dbm =          20;
```

Enabling/Disabling Algorithm Logs

The RTL logs can be enabled or disabled in the SoC Initiator, allowing for the capture of raw IQ samples from both the Initiator and Reflector devices for post-processing. This feature is enabled by default. To disable it, set **CS_INITIATOR_RTL_LOG** to 0 in the `config/cs_initiator_config.h`.

```
#ifndef CS_INITIATOR_RTL_LOG
#define CS_INITIATOR_RTL_LOG                    (1)
#endif
```

Changing the Algorithm Mode

In general, two algorithm modes are supported:

1. SL_RTL_CS_ALGO_MODE_REAL_TIME_BASIC: use this mode to track moving targets.
2. SL_RTL_CS_ALGO_MODE_STATIC_HIGH_ACCURACY: use this mode to estimate the distance of stationary objects.

By default, the Initiator uses the REAL_TIME mode. STATIC mode can be selected by pressing Button 1 on the WSTK at startup. Press **BTN1** on the WSTK while resetting the device to select STATIC mode.

Alternatively, you can change this setting in the config struct found in: *<SDK_Folder>/app/bluetooth/common/sc_initiator/src/cs_initiator_configurator.c*.

```
// RTL configuration
// Moving object tracking is enabled by default
rtl_config->algo_mode = SL_RTL_CS_ALGO_MODE_REAL_TIME_BASIC;
```

Important notes:

- In STATIC mode, a valid distance estimation is produced for every 100 CS procedure. Hence, extra care must be taken to the returned error code by the algorithm.
  - SL_RTL_ERROR_ESTIMATION_IN_PROGRESS is returned by `sl_rtl_cs_process` and `sl_rtl_cs_get_distance_estimate` methods when distance estimation is not ready.
  - SL_RTL_ERROR_SUCCESS is returned by `sl_rtl_cs_process` and `sl_rtl_cs_get_distance_estimate` methods when distance estimation is ready.
- In STATIC mode, the algorithm estimates distances with a resolution of 0.1 meters.
- There is currently no observable difference in the results between STATIC and REAL_TIME modes in RTT.

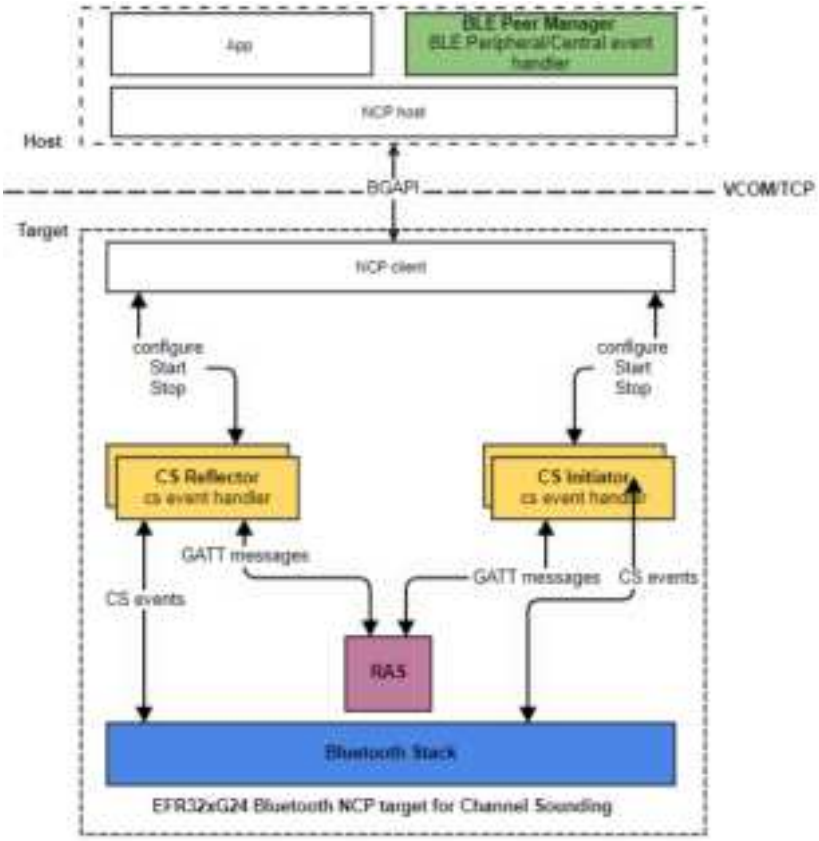
NCP Mode Sample Applications

The Silicon Labs Bluetooth SDK also includes NCP sample applications that support channel Sounding. In NCP mode, the EFR32xG24 device functions as NCP target or coprocessor, working with a host controller that functions as an NCP host. The NCP host and target communicate on a serial interface using the BGAPI protocol.

Starting with version 2024.6.0, a multi-role NCP architecture is introduced for channel sounding. This architecture allows running multiple instances with similar or different roles simultaneously. To realize this architecture, the Bluetooth SDK offers:

- A Studio example project for NCP target (app/Bluetooth/example)
  - bt_cs_ncp: **Bluetooth - NCP for Channel Sounding**
- A host example for the NCP host (app/Bluetooth/example_host)
  - bt_cs_host: supports the multi-role NCP host

In multi-role NCP architecture, the Bluetooth stack and RTL library are run on the SoC, whereas the application is run on the host machine.



Key features of the multi-role NCP architecture:

- As the name suggests, this architecture allows multiple instances* to run with different CS roles; that is, Initiator and Reflector roles.
- The Initiator and Reflector roles are integrated on the NCP target side. As a result, the host application can act as a Reflector, an Initiator, or both.
- The distance estimation is done on the NCP target, and only the measurement results are sent to the host using CS specific BGAPI commands, thereby reducing the load on serial communication.

*Up to 4 simultaneous connections are supported. This means that the number of instances (Initiator and Reflector combined) cannot exceed 4.

NCP Interface

The NCP provides a high-level interface to the host application. The host can configure, start a Reflector, or Initiator instance on the NCP target by calling `sl_bt_user_cs_service_message_to_target()`.

Any asynchronous CS event triggers an `sl_bt_evt_user_cs_service_message_to_host_id` event on the host. The host then parses this event to extract the specific CS event, which can be:

- CS result
- CS Log data from RTL, Initiator, or Reflector
- Error events

This occurs in `app.c` file of `bt_cs_host` project.

Building the Multi-Role NCP Projects

For the target, in Simplicity Studio Launcher perspective, select the **Bluetooth - NCP for Channel Sounding**, and build and flash it to the board. This sample application requires the *bootloader-uart-bgapi* type of bootloader. For the Host

application, go to *app|bluetooth|example_host|bt_cs_host*. For testing purposes, you can build the application by simply calling `$ make` .

When you are developing, however, Silicon Labs recommends that you first generate the project using the export feature of the makefile. This feature allows copying all the SiSDK files that belong to the project into the *export* folder. After the project files are exported, the *export* directory will be your working directory that is completely detached from the SiSDK but has the same folder structure inside. With export, it is clear at glance which files belong to the project.

To generate your multi-role NCP host, open a terminal, go to `app\bluetooth\example_host\bt_cs_host` , and call the following command:

```
$ make export
```

A custom export folder can be also specified using the *EXPORT_DIR* variable.

```
$ make export EXPORT_DIR=/c/users/username/my_bt_cs_host
```

After generating the project, you can build it by:

- Navigating to the exported project directory
  - `$ cd /c/users/username/my_bt_cs_host/app/bluetooth/example_host/bt_cs_host`
- Running the `$ make` command

See the required tools and more information about building an NCP Host application here.

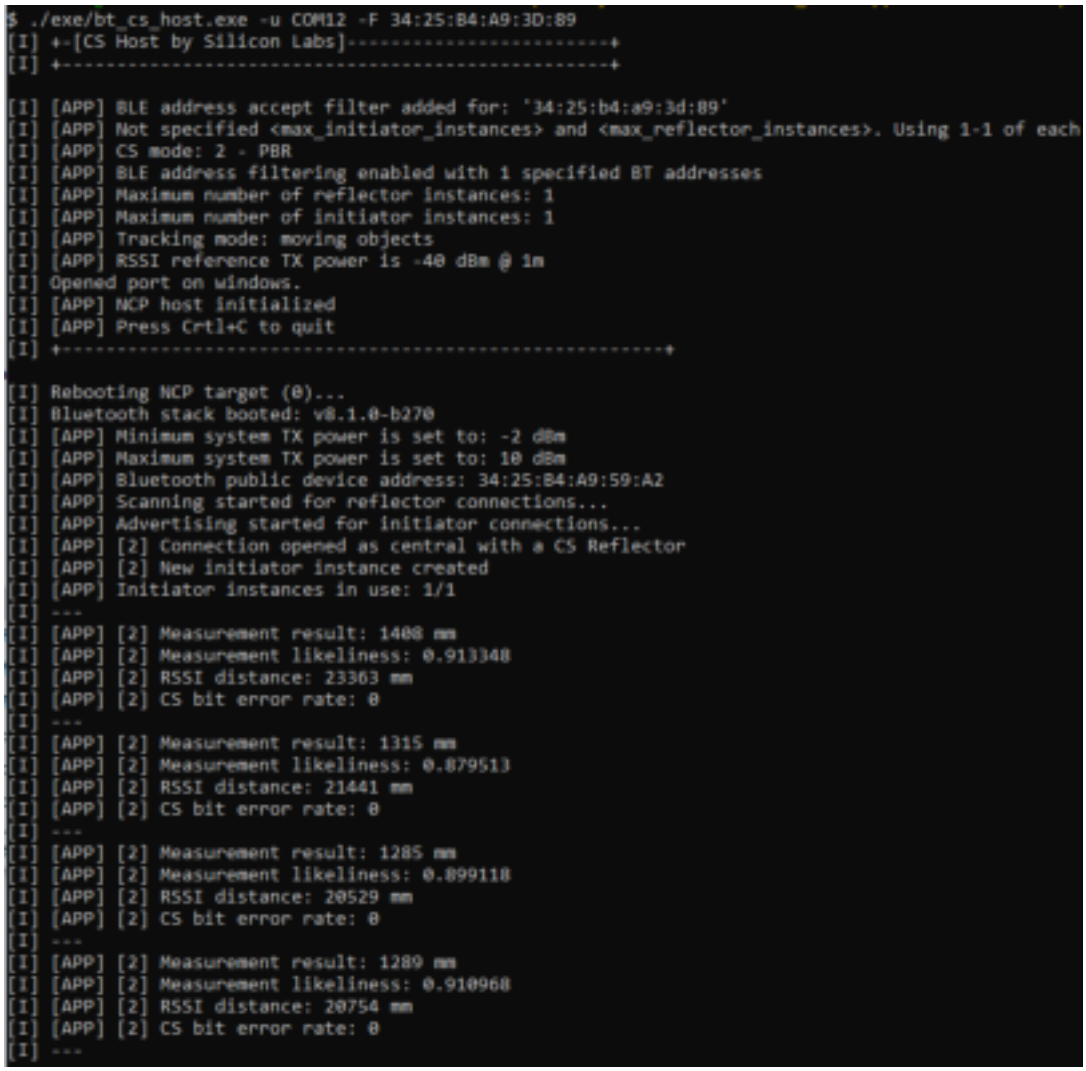
Running the Multi-Role NCP Host Application

After the project is built, an executable file `bt_cs_host.exe` is generated inside the *exe* folder. Run the application using the following command:

```
$ ./exe/bt_cs_host.exe -t <tcp_address> | -u <serial_port> [-b <baud_rate>] [-f] [-l <log_level_filter>] [-m <cs_mode>] [-R <max_reflector_instances>] [-I <max_initiator_instances>] [-F <reflector_ble_address>] [-r] [-w] [-o] [-h]
```

Options	Description
-t	Target TCP/IP connection parameters (if WSTK is connected via Ethernet). <address> IP address of the WSTK board.
-u	Target USB serial connection parameter (if WSTK is connected via USB). <serial_port> COM port (Windows) or device file (POSIX) to be opened.
Optional Parameters	**Description**
-b	Baudrate can be given if the connection is established via serial port. <baud_rate> Baud rate of the serial connection (default: 115200).
-f	Disable flow control (RTS/CTS), default: enabled
-l	Application log level filter. <level>: Integer representing log level. 0: Critical. 1: Critical, error. 2: Critical, error, warning. 3: Critical, error, warning, info (default). 4: Critical, error, warning, info, debug.

Options	Description
-m	CS mode <cs_mode>: number corresponding to CS Mode. 1: RTT. 2: PBR (default).
-I	Maximum number of initiator instances <max_initiator_instances>: 0 to 4 (default: 1).
-R	Maximum number of reflector instances. <max_reflector_instances>: 0 to 4 (default: 1).
-F	Enable reflector BLE address filtering. <reflector_ble_address> e.g -F 68:12:EF:34:18:AB
-w	Use wired antenna offset.
-o	Object tracking mode, default: 0 <o> 0: Moving object tracking (up to 5 km/h). 1: Stationary object tracking.
-h	Displays the help menu.



CPC Support

The CS Initiator sample application also supports the CPC over NCP feature. For details about how to add the CPC interface to the project refer to the NCP Application Note. This covers the basic setup of the host application on a standard Linux system. It is also possible to use OpenWRT based systems as a host. Contact Silicon Labs for a sample setup and instructions.

Simplicity Studio Channel Sounding Analyzer

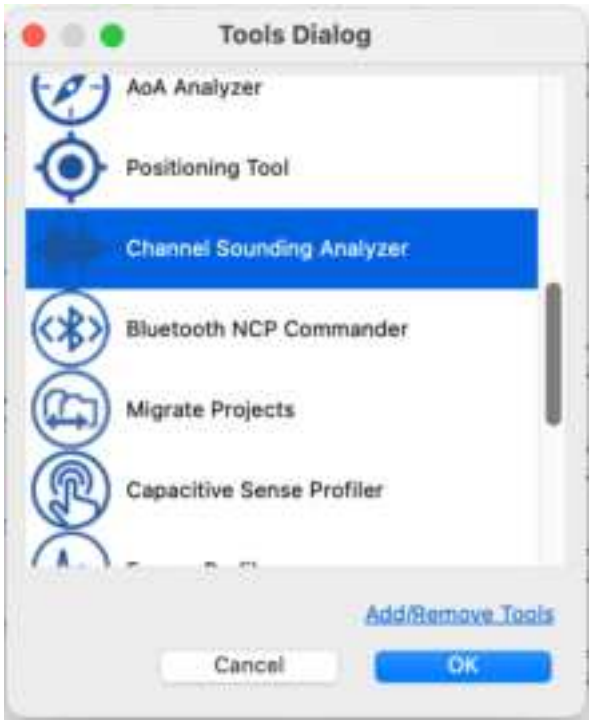
You are viewing documentation for version: 0.1 | Version History

Simplicity Studio Channel Sounding Analyzer

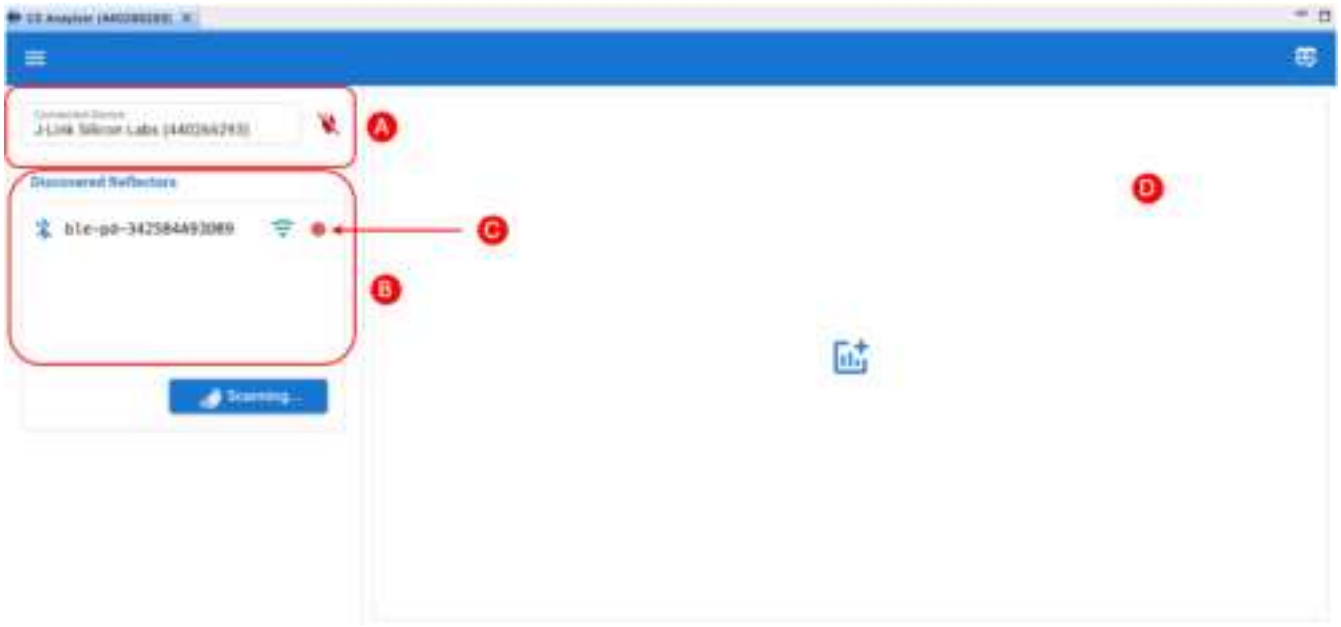
With the release of Simplicity Studio v5.9, Silicon Labs introduces Channel Sounding Analyzer, which is similar to the previous Accurate Bluetooth Ranging Analyzer tool with a few UI enhancements to accommodate the new host-NCP architecture where the RTL runs on the SoC. Channel Sounding Analyzer is designed to visualize the estimated distance, perform data analysis, and help with troubleshooting. The tool can be accessed through Simplicity Studio's Tools menu.

To launch Channel Sounding Analyzer:

1. From Simplicity Studio 5's Launcher perspective, click **Tools**.
2. In the Tools dialog, select **Channel Sounding Analyzer**.



3. Click **OK**. Channel Sounding Analyzer opens in a new tab.



A. Allows connecting to the Initiator device using its J-Link ID.
B. Lists discovered Reflector devices.
C. Starts recording the measurement results.
D. Enables different chart layout.

Note: Channel Sounding Analyzer works with a Channel Sounding enabled radio board (Ex: BRD4198A) that is programmed with **Bluetooth – NCP for Channel Sounding**.

Key Features

Out-of-the-Box Device Connectivity

Through a streamlined interface, you can establish connection to the J-Link of your Initiator device and start scanning for a Reflector device.

To start a connection:

1. Select the J-Link ID of your Initiator device and connect it using the connect plugin. Scanning starts automatically.
2. Connect to your Reflector device from the list.
3. Select the CS mode: **PBR** or **RTT**.
4. Enable/disable **Moving Object Tracking** and click **OK**. The tool starts visualizing the distance estimation automatically.

By default, CS **PBR** mode is selected, and **Moving Object Tracking** is enabled.

When **Moving Object Tracking** is enabled, `SL_RTL_CS_ALGO_MODE_REAL_TIME_BASIC` is selected automatically.

When **Moving Object Tracking** is disabled, `SL_RTL_CS_ALGO_MODE_STATIC_HIGH_ACCURACY` is selected, which is recommended for tracking stationary objects.

The STATIC mode results are visualized using a boxplot as follows:

- Each point corresponds to a distance output from the RTL library using static mode (1).
- A valid distance estimation is generated for every N CS procedure, and its progress is displayed as a percentage in the progress bar (2).
- When hovering over the interquartile range (i.e., the box), you can view the statistical results of the static mode (3).
- Clicking the table icon opens a table wizard that displays timestamps and corresponding distance estimations (4).

Enhanced Visualization

Channel Sounding Analyzer provides a more advanced and adaptable chart layout for representing measurements.

Using the menu bar in the charts you can:

1. Change the chart layout.
2. Select lighting preference.
3. Turn on/off the RSSI based distance measure-ment.
4. Set the sample rate of the visualizer.
5. Adjust the time window.
6. Set the maximum and minimum distance range.
7. Go to full screen.
8. Remove the chart.

Data Analysis and Debugging

Channel Sounding Analyzer can also be used for analyzing different CS data and debugging purpose.

You can use the Channel Sounding Analyzer to:

1. Enable logging RTL logs to a JSON file. This must be enabled before connecting to the J-Link ID of the Initiator.
2. Enable logging distance estimations into a JSON file.
3. Visualize the measured distance values.
4. Monitor the likeliness of the measurement results.
5. Visualize the IQ amplitudes of the Initiator.
6. Visualize the IQ amplitudes of the Reflector.
7. Visualize the IQ phases values calculated by the Initiator.
8. Visualize the tone quality.

Channel Sounding Performance Metrics

You are viewing documentation for version: 0.1 | Version History

Channel Sounding Performance Metrics

Silicon Labs has performed real environment testing in an indoor office setting to best represent the performance of the Channel Sounding distance estimation solution. The images below show the floor map of the office with 6 static devices and one mobile device. The mobile device (H) and one of the static devices (1) are mounted on a ceiling rail robot. All devices used in the tests are the BRD4198A boards.

The table below provides the performance metrics of PBR and RTT distance estimation based on measurements taken at multiple distances between 0.5 m and 34 meters, and in different measurement configurations:

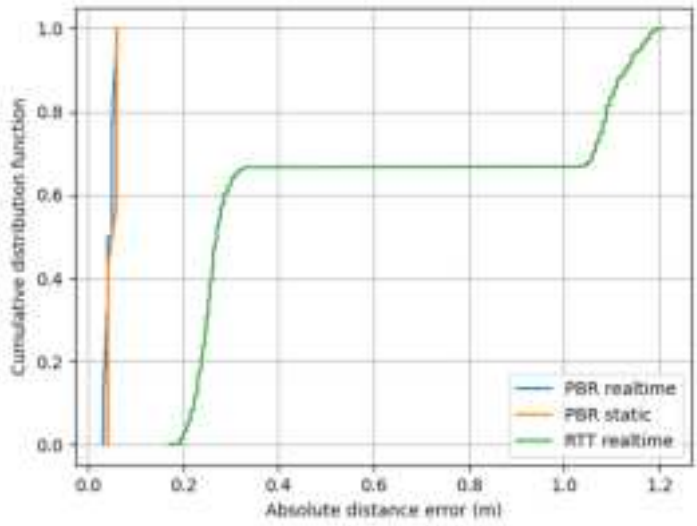
- Conducted, line-of-sight (LOS), and non-line-of-sight (NLOS) environments
- SL_RTL_CS_ALGO_MODE_REAL_TIME_BASIC and SL_RTL_CS_ALGO_MODE_STATIC_HIGH_ACCURACY algorithm modes referenced by Real-time and Static respectively

Measurement configuration			90th percentile of abs error (m)	95th percentile of abs error (m)	Std of the signed error (m)	Mean Signed Error (m)
Conducted	PBR	Static	0.06	0.06	0.01	-0.05
		Real-time	0.06	0.06	0.01	-0.05
	RTT	Real-time	1.13	1.16	0.64	0.2
LOS	PBR	Static	0.22	0.61	0.26	0.01
		Real-time	0.28	0.32	0.13	0.14
	RTT	Real-time	3.95	4.92	0.88	2.81
NLOS	PBR	Static	3.59	5.01	2.2	-0.7
		Real-time	4.25	5.09	2.02	1.81
	RTT	Real-time	14.76	15.5	4.04	6.4

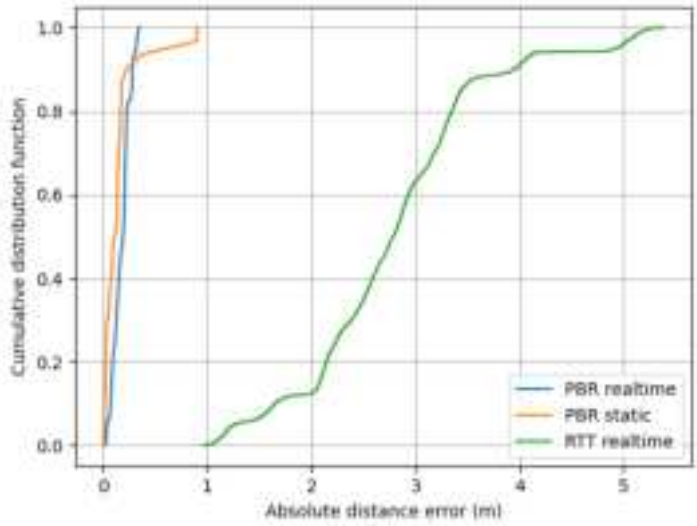
The following Cumulative Distributive function (CDF) plots show the probability of the absolute distance errors in each measurement configuration.

CDF in Conducted Environment

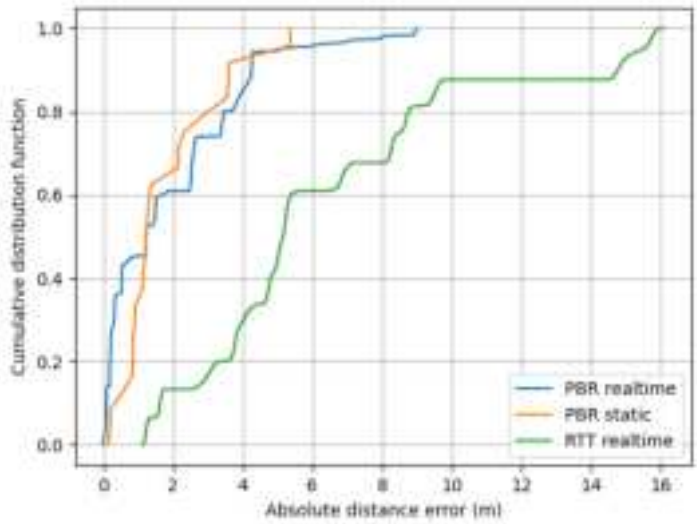
In the conducted test setup, the antenna paths of 2 BRD4198A boards are connected via a RF coaxial cable with attenuation added to the signal path.



CDF in LOS Environment



CDF in NLOS Environment



Note that Channel Sounding is still in the development phase and software improvements are made in each software release to improve the performance of both PBR and RTT modes. See Known Issues and Limitation section for more information.

Optimizing Performance

The antenna of a device plays a significant role in achieving accurate distance estimations based on its characteristics such as group delay, radiation pattern, and others. To learn more about the antenna design guidelines and considerations to improve the accuracy of distance measurement using Channel Sounding, see AN1493.

Known Issues and Limitations

You are viewing documentation for version: 0.1 | Version History

Known Issues and Limitations

- When using the command line to build the project instead of using the Studio interface, the project shall be built without the “-cp” option.
- Environmental conditions e.g. wall, metal acting as RF reflectors can affect accuracy. Improvements to reduce these effects are being worked upon and will be made available in subsequent releases.
- Interoperability with other vendors’ solutions is not yet supported.
- On OpenWRT the host application sometimes fails to open CPC connection over SPI. Restarting the Host application solves the connection issue.
- Limited testing was performed for multiple Initiators <-> one reflector and multiple reflectors <-> one Initiator configurations.
- The accuracy of RTT is sub-optimal compared to PBR and performance improvements are under development.

Calibration of Distance Ranging

You are viewing documentation for version: 0.1 | Version History

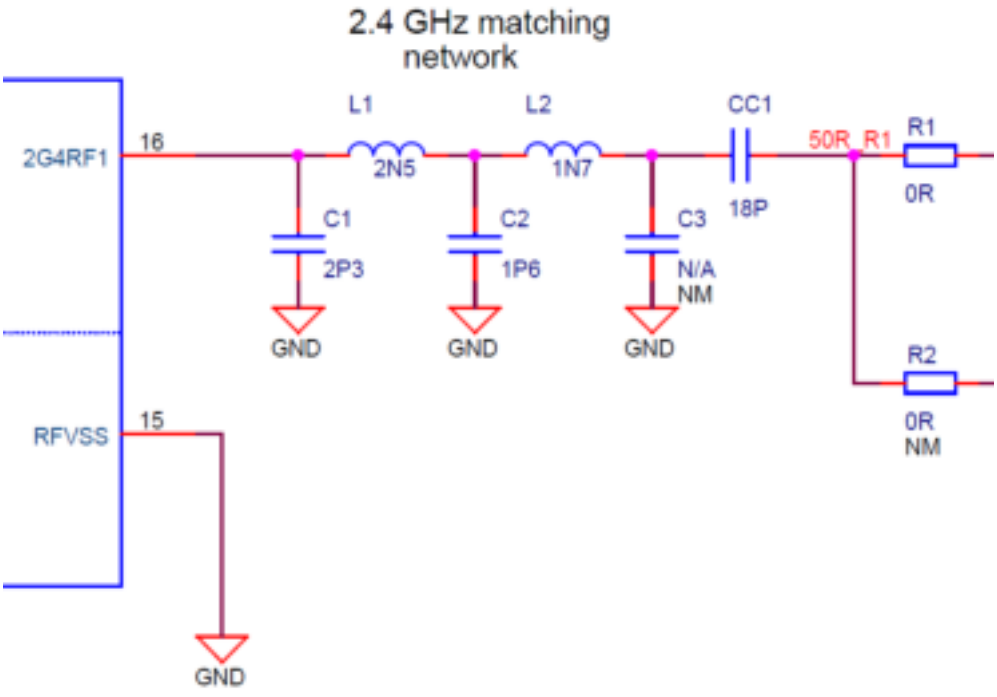
Calibration of Silicon Labs Distance Ranging

This section discusses the theory and procedure of calibrating a Silicon Labs channel sounding SoC for accurate distance ranging. A wireless calibration should be performed per board design. Antenna recommendations should be adhered to.

The Bluetooth Core Spec requires that channel sounding implementations refer their measurements to their antenna port. This allows the system to be interoperable without concern for the circuit delays of implementations. Referring the PCT to the antenna is done by rotating complex numbers to compensate for the group delay. Extra delay is equivalent to extra distance and affects the accuracy of the system. The delays can be separated into the internal radio group delay and the board and antenna group delay.

The internal radio group delay is the delay from the RF and digital filter circuits. Silicon Labs has compensated for the variation of this delay over RF gain. This is automatically applied, and no further compensation is needed.

The board delay can be further divided into the delay from the matching network and delay from traces or cabling. The recommended matching network for the EFR32XG24 was simulated over its component tolerances. The notes below the figure show that, while there is a phase shift, the group delay remains relatively constant. Further analysis shows that the component tolerance should contribute only up to +/- 0.05 m.

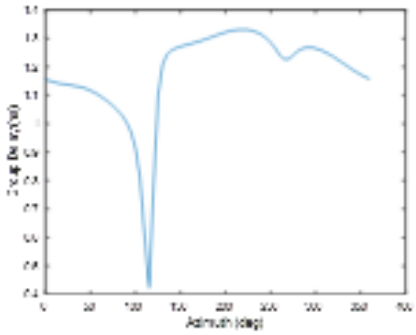


- C1: 2.3pF +/-0.05pF Murata GRM0335C1H2R3WA01
- L1: 2.5nH +/-0.05nH Murata LQP03HQ2N5W02
- C2: 1.6pF +/-0.05pF Murata GRM0335C1H1R6WA01
- L2: 1.7nH +/-0.05nH Murata LQP03HQ1N7W02
- C3: No population

- CC1: 18pF +/-2% Murata GJM0335C1E180GB01

The traces and cabling delay consists of the propagation through the RF traces and coaxial cable if an off-board antenna is used. Direct measurement of these delays is not necessary, as the delays are additive and are captured in the wireless calibration. The speed of propagation through these materials is slower than the speed of light in a vacuum. This ratio is the velocity factor and varies widely from 0.6 – 0.9 depending on the construction and geometry. This explains why when inserting a cable of physical length L, the measured distance will shift by L / VF. For designs with multiple antenna paths, the calibration should be run once per antenna path.

The last component is the antenna contributing delay. The plot below shows the group delay over azimuth for a printed inverted F antenna. With a group delay of 1-1.5 ns, the antennas are contributing approximately 0.75 m of offset. Due to the antenna distance offset, Silicon Labs recommends a wireless distance offset calibration, as a cabled test will not include this significant offset.



Calibration Procedure

This section describes how perform the wireless calibration procedure.

The measurement should be configured for maximum measurement time T_PM=40 us, and at least two mode0 steps. It is preferable to have the entire channel sweep contained in a single subevent. This is the default configuration for the provided sample application. It is acceptable if these parameters deviate from the configuration used in the field. The purpose is to use the highest accuracy configuration for configuration.

The devices should be elevated off the ground by at least 1 m. The environment should be open to avoid multi-path effects, i.e., a large room. Avoid having objects in the line-of-sight path.

1. Prepare two devices for wireless distance measurements.
2. Set the distance offset calibration to 0 m with the API sl_bt_cs_set_antenna_configuration.
3. Reflash the devices.
4. Run the measurements and algorithm for 100 distance measurements. Record the median of the distribution.
5. Repeat for at least 5 distances between 1 and 20 m.
6. Calculate a linear regression of the expected distance vs measured distance. The distance offset is the y-intercept.
7. Set the distance offset calibration to half of the distance offset. Reflash the devices.
8. Repeat step 4 and 5 making measurements.
9. Verify the distance error is centered around zero.

Silicon Labs RTL library API Reference Guide

You are viewing documentation for version: 8.2.0 | Version History

Overview

This is the API for the Silicon Labs Real-Time Locationing library. It provides an interface for estimating arrival and departure angles of signals and the positions of AoA/AoD signal transmitters. AoA stands for Angle-of-Arrival and AoD for Angle-of-Departure. AoX is used here when referring to both techniques.

Estimators can be created individually for each locator node using the API. Additionally, multiple estimators with different parameters can be created for a single node. An instance of the estimator is created as a libitem and the estimator is initialized. Next, the estimation parameters such as antenna array type, number of antennas, estimation mode, and so on can be set using the function calls described below.

One snapshot is a set of IQ-samples with exactly one sample for each antenna.

Software Example

Example: Initialize an estimator, set up parameters, create the estimator, and calculate an angle estimate from previously captured I/Q samples. Finally, deinitialize the estimator after returning the angles.

```
// Estimator instance
sl_rtl_aox_libitem libitem1;

// Initialize the estimator
sl_rtl_aox_init(&libitem1);

// Set number of snapshots to 3
sl_rtl_aox_set_num_snapshots(&libitem1, 3);

// Set array type to SiLabs reference 4×4 Uniform Rectangular Array
sl_rtl_aox_set_array_type(&libitem1, SL_RTL_AOX_ARRAY_TYPE_4×4_URA);

// Set estimator mode to basic mode, which requires 10 estimation rounds
sl_rtl_aox_set_mode(&libitem1, SL_RTL_AOX_MODE_BASIC);

// Enable IQ sample quality analysis processing (optional)
sl_rtl_aox_iq_sample_qa_configure(&libitem1);

// Create the estimator after all of the parameters are set
sl_rtl_aox_create_estimator(&libitem1);

// Set library switch pattern mode and the pattern itself
sl_rtl_aox_set_switch_pattern_mode(&libitem1, SL_RTL_AOX_SWITCH_PATTERN_MODE_EXTERNAL);

const uint32_t number_of_channels = 4*4;
uint32_t switch_pattern[number_of_channels] = {0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15};
sl_rtl_aox_update_switch_pattern(&libitem1, switch_pattern, 0);

// Configure radio with the same switching pattern
radioConfigureSwitchPattern(switch_pattern, number_of_channels); // Dummy function

// Setup variables
float azimuthOut, elevationOut;
float channelFrequency = 2.46e9f;

float *IReference, *QReference;
float **ISamples, **QSamples;
// Allocate buffers for reference period I/Q data, size 8 samples
allocate1DFloatBuffer(&IReference, 8); // Dummy function
allocate1DFloatBuffer(&QReference, 8); // Dummy function

// Allocate buffers for I/Q data from all antennas, size 3 snapshots x 16 antennas
allocate2DFloatBuffer(&ISamples, 3, 16); // Dummy function
allocate2DFloatBuffer(&QSamples, 3, 16); // Dummy function

sl_rtl_error_code result = SL_RTL_ERROR_ESTIMATION_IN_PROGRESS;

while (result != SL_RTL_ERROR_SUCCESS) {
  // Wait for new I/Q samples from radio
  waitForNewIqSampleReport(); // Dummy function

  int index = 0;
  // Get captured I/Q reference period samples from radio (example)
  for (uint32_t sample = 0; sample < 8; ++sample) {
     IReference[sample] = iq_report.samples.data[index]; // Dummy data
     QReference[sample] = iq_report.samples.data[index + 1]; // Dummy data
     index += 2;
   }
  }

  // Calculate and set phase rotation to compensate for frequency offset and switching
  float phaseRotation;
  sl_rtl_aox_calculate_iq_sample_phase_rotation(&libitem1, 2.0f, IReference, QReference, 8, &phaseRotation);
  sl_rtl_aox_set_iq_sample_phase_rotation(&libitem, phaseRotation);

  // Get captured I/Q samples from all antennas from radio (example)
```

```
for(uint32_t snapshot =0; snapshot <3;++snapshot){for(uint32_t antenna =0; antenna <16;++antenna){
    ISamples[snapshot][antenna]= iq_report.samples.data[index];// Dummy data
    QSamples[snapshot][antenna]= iq_report.samples.data[index +1];// Dummy data
    index +=2;}}// Feed I/Q data to estimator and calculate estimates
 result =sl_rtl_aox_process(&libitem1, ISamples, QSamples,
  channelFrequency,&azimuthOut,&elevationOut);*returnAzimuth = azimuthOut;*returnElevation = elevationOut;// get the overall result of the IQ
sample quality results
uint32_t sample_qa_results =sl_rtl_aox_get_iq_sample_qa_results(&libitem1);

sl_rtl_clib_iq_sample_qa_dataset_t quality_results;
sl_rtl_clib_iq_sample_qa_antenna_data_t antenna_data[numAntennas];// get the IQ sample quality results for the latest
packetsl_rtl_aox_iq_sample_qa_get_details(&libitem1,&quality_results, antenna_data);// get the IQ sample quality results for the latest packet using
the certain radio channel// (which is not necessarily the latest packet received)sl_rtl_aox_iq_sample_qa_get_channel_details(&libitem1,
bt_channel,&quality_results, antenna_data);sl_rtl_aox_deinit(&libitem1);
```

License

Angle of Arrival / Departure

Angle of Arrival / Departure

Angle of Arrival / Departure.

These functions are related to the calculation of the Angle of Arrival and Angle of Departure from I/Q samples. The angles can be calculated following these steps:

1. Initialize a sl_rtl_aox_libitem instance.
2. Set up the antenna array and angle calculation parameters.
3. Create the estimator.
4. Set the antenna switching pattern.
5. Input the I/Q data into the libitem.
6. Process the I/Q data into an angle.

Modules

sl_rtl_clib_iq_sample_qa_antenna_data_t

sl_rtl_clib_iq_sample_qa_dataset_t

Enumerations

enum sl_rtl_aox_array_type {

SL_RTL_AOX_ARRAY_TYPE_4×4_URA = 0
SL_RTL_AOX_ARRAY_TYPE_3×3_URA
SL_RTL_AOX_ARRAY_TYPE_1×4_ULA
SL_RTL_AOX_ARRAY_TYPE_4×4_DP_URA
SL_RTL_AOX_ARRAY_TYPE_COREHW_15×15_DP
SL_RTL_AOX_ARRAY_TYPE_COREHW_12×12_DP
SL_RTL_AOX_ARRAY_TYPE_COREHW_4×4_URA
SL_RTL_AOX_ARRAY_TYPE_COREHW_2×2_URA
SL_RTL_AOX_ARRAY_TYPE_LAST

}

AoX antenna array type.

enum sl_rtl_aox_switch_pattern_array {

SL_RTL_AOX_SWITCH_PATTERN_ARRAY_4×4_CP = 0
SL_RTL_AOX_SWITCH_PATTERN_ARRAY_4×4_DP
SL_RTL_AOX_SWITCH_PATTERN_ARRAY_LAST

}

enum sl_rtl_aox_switch_pattern_options {

SL_RTL_AOX_SWITCH_PATTERN_OPTIONS_DUAL_POLARIZED = 0×1
SL_RTL_AOX_SWITCH_PATTERN_OPTIONS_EXTRA_REFERENCE = 0×2

}

enum sl_rtl_aox_mode {

SL_RTL_AOA_MODE_ONE_SHOT_BASIC = 3
SL_RTL_AOA_MODE_ONE_SHOT_BASIC_LIGHTWEIGHT
SL_RTL_AOA_MODE_ONE_SHOT_FAST_RESPONSE
SL_RTL_AOA_MODE_ONE_SHOT_HIGH_ACCURACY
SL_RTL_AOA_MODE_ONE_SHOT_BASIC_AZIMUTH_ONLY
SL_RTL_AOA_MODE_ONE_SHOT_FAST_RESPONSE_AZIMUTH_ONLY
SL_RTL_AOA_MODE_ONE_SHOT_HIGH_ACCURACY_AZIMUTH_ONLY
SL_RTL_AOA_MODE_REAL_TIME_FAST_RESPONSE
SL_RTL_AOA_MODE_REAL_TIME_BASIC
SL_RTL_AOA_MODE_REAL_TIME_HIGH_ACCURACY
SL_RTL_AOA_MODE_LAST
SL_RTL_AOD_MODE_ONE_SHOT_BASIC = 3 | 0×20
SL_RTL_AOD_MODE_ONE_SHOT_BASIC_LIGHTWEIGHT
SL_RTL_AOD_MODE_ONE_SHOT_FAST_RESPONSE
SL_RTL_AOD_MODE_ONE_SHOT_HIGH_ACCURACY
SL_RTL_AOD_MODE_ONE_SHOT_BASIC_AZIMUTH_ONLY
SL_RTL_AOD_MODE_ONE_SHOT_FAST_RESPONSE_AZIMUTH_ONLY
SL_RTL_AOD_MODE_ONE_SHOT_HIGH_ACCURACY_AZIMUTH_ONLY
SL_RTL_AOD_MODE_REAL_TIME_FAST_RESPONSE
SL_RTL_AOD_MODE_REAL_TIME_BASIC
SL_RTL_AOD_MODE_REAL_TIME_HIGH_ACCURACY
SL_RTL_AOD_MODE_LAST
SL_RTL_AOX_MODE_ONE_SHOT_BASIC = 3 | 0×40
SL_RTL_AOX_MODE_ONE_SHOT_BASIC_LIGHTWEIGHT
SL_RTL_AOX_MODE_ONE_SHOT_FAST_RESPONSE
SL_RTL_AOX_MODE_ONE_SHOT_HIGH_ACCURACY
SL_RTL_AOX_MODE_ONE_SHOT_BASIC_AZIMUTH_ONLY
SL_RTL_AOX_MODE_ONE_SHOT_FAST_RESPONSE_AZIMUTH_ONLY
SL_RTL_AOX_MODE_ONE_SHOT_HIGH_ACCURACY_AZIMUTH_ONLY
SL_RTL_AOX_MODE_REAL_TIME_FAST_RESPONSE
SL_RTL_AOX_MODE_REAL_TIME_BASIC
SL_RTL_AOX_MODE_REAL_TIME_HIGH_ACCURACY
SL_RTL_AOX_MODE_LAST

}

AoA, AoD (and AoX for backward compatibility) estimator modes.

enum sl_rtl_aox_constraint_type {

SL_RTL_AOX_CONSTRAINT_TYPE_AZIMUTH = 0
SL_RTL_AOX_CONSTRAINT_TYPE_ELEVATION

}

enum sl_rtl_aox_switch_pattern_mode {

SL_RTL_AOX_SWITCH_PATTERN_MODE_DEFAULT = 0
SL_RTL_AOX_SWITCH_PATTERN_MODE_RANDOM
SL_RTL_AOX_SWITCH_PATTERN_MODE_EXTERNAL
SL_RTL_AOX_SWITCH_PATTERN_MODE_EXTRA_REFERENCE

}

enum sl_rtl_slib_iq_sample_qa_result_t {

SL_RTL_AOX_IQ_SAMPLE_QA_INVAL_REF = 0
SL_RTL_AOX_IQ_SAMPLE_QA_DCOFFSET = 2
SL_RTL_AOX_IQ_SAMPLE_QA_SNDR = 3
SL_RTL_AOX_IQ_SAMPLE_QA_ROTATING_ERROR = 4
SL_RTL_AOX_IQ_SAMPLE_QA_REF_ANT_PHASE_VALUE = 5
SL_RTL_AOX_IQ_SAMPLE_QA_REF_ANT_PHASE_JITTER = 6
SL_RTL_AOX_IQ_SAMPLE_QA_ANT_X_PHASE_JITTER = 7
SL_RTL_AOX_IQ_SAMPLE_QA_ALL_SAME_PHASE = 8
SL_RTL_AOX_IQ_SAMPLE_QA_SWICHING_JITTER = 9

}

Typedefs

typedef void * sl_rtl_aox_libitem
Angle of Arrival / Departure library item.

typedef void * sl_rtl_aox_antenna_pattern
Angle of Arrival / Departure antenna array radiation pattern.

Functions

enum sl_rtl_error_code sl_rtl_aox_init(sl_rtl_aox_libitem *item)

enum sl_rtl_error_code sl_rtl_aox_deinit(sl_rtl_aox_libitem *item)

enum sl_rtl_error_code sl_rtl_aox_set_num_snapshots(sl_rtl_aox_libitem *item, uint32_t num_snapshots)

enum sl_rtl_error_code sl_rtl_aox_set_array_type(sl_rtl_aox_libitem *item, enum sl_rtl_aox_array_type array_type)

enum sl_rtl_error_code sl_rtl_aox_set_mode(sl_rtl_aox_libitem *item, enum sl_rtl_aox_mode mode)

enum sl_rtl_error_code sl_rtl_aox_calculate_iq_sample_phase_rotation(sl_rtl_aox_libitem *item, float iq_data_downsampling_factor, float *i_samples, float *q_samples, uint32_t num_samples, float *phase_rotation_out)

enum sl_rtl_error_code sl_rtl_aox_set_iq_sample_phase_rotation(sl_rtl_aox_libitem *item, float phase_rotation)

enum sl_rtl_error_code sl_rtl_aox_add_constraint(sl_rtl_aox_libitem *item, enum sl_rtl_aox_constraint_type constraint_type, float min_value, float max_value)

enum sl_rtl_error_code sl_rtl_aox_set_sample_rate(sl_rtl_aox_libitem *item, float sampleRate)

enum sl_rtl_error_code sl_rtl_aox_set_num_radio_channels(sl_rtl_aox_libitem *item, uint32_t channels)

enum sl_rtl_error_code sl_rtl_aox_iq_sample_qa_configure(sl_rtl_aox_libitem *item)

uint32_t sl_rtl_aox_iq_sample_qa_get_results(sl_rtl_aox_libitem *item)

enum sl_rtl_error_code sl_rtl_aox_iq_sample_qa_get_details(sl_rtl_aox_libitem *item, sl_rtl_clib_iq_sample_qa_dataset_t *results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)

enum sl_rtl_error_code sl_rtl_aox_iq_sample_qa_get_channel_details(sl_rtl_aox_libitem *item, uint8_t channel, sl_rtl_clib_iq_sample_qa_dataset_t *results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)

enum sl_rtl_error_code sl_rtl_aox_create_estimator(sl_rtl_aox_libitem *item)

enum sl_rtl_error_code sl_rtl_aox_convert_raw_samples(sl_rtl_aox_libitem *item, uint32_t start_offset, float iq_data_downsampling_factor, float *raw_i_samples_in, float *raw_q_samples_in, uint32_t num_raw_samples_in, float **i_samples_out, float **q_samples_out, uint32_t num_snapshots_out)

enum sl_rtl_error_code sl_rtl_aox_calculate_number_of_snapshots(sl_rtl_aox_libitem *item, uint32_t num_raw_samples_in, uint32_t start_offset, float iq_data_downsampling_factor, uint32_t num_channels, uint32_t *num_snapshots_out)

enum sl_rtl_error_code sl_rtl_aox_set_switch_pattern_mode(sl_rtl_aox_libitem *item, enum sl_rtl_aox_switch_pattern_mode mode)

enum sl_rtl_error_code sl_rtl_aox_update_switch_pattern(sl_rtl_aox_libitem *item, uint32_t *switch_pattern_in, uint32_t **switch_pattern_out)

enum sl_rtl_error_code sl_rtl_aox_set_switch_pattern_seed(sl_rtl_aox_libitem *item, int32_t seed_value)

enum sl_rtl_error_code sl_rtl_aox_convert_switch_pattern(sl_rtl_aox_libitem *item, uint32_t array_id, uint32_t options, uint32_t switch_pattern_size_in, uint32_t *switch_pattern_in, uint32_t *switch_pattern_size_out, uint32_t **switch_pattern_out)

enum sl_rtl_error_code sl_rtl_aox_reset_estimator(sl_rtl_aox_libitem *item)

enum sl_rtl_error_code sl_rtl_aox_process(sl_rtl_aox_libitem *item, float **i_samples, float **q_samples, float tone_frequency, float *az_out, float *el_out)

enum sl_rtl_error_code sl_rtl_aox_get_latest_aox_standard_deviation(sl_rtl_aox_libitem *item, float *az_std_dev, float *el_std_dev)

enum sl_rtl_error_code sl_rtl_aox_get_latest_aoa_standard_deviation(sl_rtl_aox_libitem *item, float *az_std_dev, float *el_std_dev)

enum sl_rtl_error_code sl_rtl_aox_get_latest_aod_standard_deviation(sl_rtl_aox_libitem *item, float *az_std_dev, float *el_std_dev)

enum sl_rtl_error_code sl_rtl_aox_set_expected_direction(sl_rtl_aox_libitem *item, float expected_az, float expected_el)

enum sl_rtl_error_code sl_rtl_aox_set_expected_deviation(sl_rtl_aox_libitem *item, float deviation_az, float deviation_el)

enum sl_rtl_error_code sl_rtl_aox_clear_expected_direction(sl_rtl_aox_libitem *item)

enum sl_rtl_error_code sl_rtl_aox_enable_spectrum(sl_rtl_aox_libitem *item, bool enable)

enum sl_rtl_error_code sl_rtl_aox_get_spectrum_size(sl_rtl_aox_libitem *item, uint32_t *rows, uint32_t *cols)

enum sl_rtl_error_code sl_rtl_aox_get_polarization_spectrum_size(sl_rtl_aox_libitem *item, uint32_t *rows, uint32_t *cols)

enum sl_rtl_error_code sl_rtl_aox_get_spectrum(sl_rtl_aox_libitem *item, float **spectrum_out)

enum sl_rtl_error_code sl_rtl_aox_get_polarization_spectrum(sl_rtl_aox_libitem *item, float **spectrum_out)

enum sl_rtl_error_code sl_rtl_aox_antenna_pattern_init(sl_rtl_aox_antenna_pattern *pattern, enum sl_rtl_aox_array_type array_type)

enum sl_rtl_error_code sl_rtl_aox_set_antenna_pattern(sl_rtl_aox_libitem *item, sl_rtl_aox_antenna_pattern *pattern)

enum sl_rtl_error_code sl_rtl_aox_antenna_pattern_deinit(sl_rtl_aox_antenna_pattern *pattern)

Macros

#define SL_RTL_AOX_IQ_SAMPLE_QA_ALL_OK 0

#define SL_RTL_AOX_IQ_SAMPLE_QA_FAILURE 0xffffffff

#define SL_RTL_AOX_IQ_SAMPLE_QA_CLEAR_BIT (code, bit)

#define SL_RTL_AOX_IQ_SAMPLE_QA_SET_BIT (code, bit)

#define SL_RTL_AOX_IQ_SAMPLE_QA_IS_SET (code, bit)

Enumeration Documentation

sl_rtl_aox_array_type

> sl_rtl_aox_array_type

AoX antenna array type.

Enumerator	
SL_RTL_AOX_ARRAY_TYPE_4×4_URA	Silicon Labs Ref. 4x4 Uniform Rectangular Array.
SL_RTL_AOX_ARRAY_TYPE_3×3_URA	Silicon Labs Ref. 3x3 Uniform Rectangular Array.
SL_RTL_AOX_ARRAY_TYPE_1×4_ULA	Silicon Labs Ref. 1x4 Uniform Linear Array.
SL_RTL_AOX_ARRAY_TYPE_4×4_DP_URA	Silicon Labs Ref. 4x4 Uniform Dual Polarized Rectangular Array.
SL_RTL_AOX_ARRAY_TYPE_COREHW_15×15_DP	CoreHw Ref. 150 mm x 150 mm, 8 Element Dual Polarized Array.
SL_RTL_AOX_ARRAY_TYPE_COREHW_12×12_DP	CoreHw Ref. 120 mm x 120 mm, 8 Element Dual Polarized Array.
SL_RTL_AOX_ARRAY_TYPE_COREHW_4×4_URA	CoreHw Ref. 4x4 Uniform Rectangular Array.
SL_RTL_AOX_ARRAY_TYPE_COREHW_2×2_URA	CoreHw Ref. 2x2 Uniform Rectangular Array.
SL_RTL_AOX_ARRAY_TYPE_LAST	Placeholder.

Definition at line 97 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_switch_pattern_array

> sl_rtl_aox_switch_pattern_array

Enumerator	
SL_RTL_AOX_SWITCH_PATTERN_ARRAY_4×4_CP	
SL_RTL_AOX_SWITCH_PATTERN_ARRAY_4×4_DP	
SL_RTL_AOX_SWITCH_PATTERN_ARRAY_LAST	

Definition at line 110 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_switch_pattern_options

> sl_rtl_aox_switch_pattern_options

Enumerator	
SL_RTL_AOX_SWITCH_PATTERN_OPTIONS_DUAL_POLARIZED	
SL_RTL_AOX_SWITCH_PATTERN_OPTIONS_EXTRA_REFERENCE	

Definition at line 117 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_mode

sl_rtl_aox_mode

AoA, AoD (and AoX for backward compatibility) estimator modes.

Enumerator	
SL_RTL_AOA_MODE_ONE_SHOT_BASIC	Medium filtering, medium response. Returns 2D angle, requires 10 rounds. Most suitable for single shot measurement.
SL_RTL_AOA_MODE_ONE_SHOT_BASIC_LIGHTWEIGHT	Medium filtering, medium response, low CPU cost & low elevation resolution. 2D angle, req. 10 rounds. Most suitable for single shot measurement.
SL_RTL_AOA_MODE_ONE_SHOT_FAST_RESPONSE	Low filtering, fast response, low CPU cost & low elevation resolution. 2D angle, requires 2 rounds. Most suitable for single shot measurement.
SL_RTL_AOA_MODE_ONE_SHOT_HIGH_ACCURACY	High filtering, slow response. 2D angle, requires 20 rounds. Most suitable for single shot measurement.
SL_RTL_AOA_MODE_ONE_SHOT_BASIC_AZIMUTH_ONLY	Equivalent to ONE_SHOT_BASIC with low CPU cost and returns 1D angle. Most suitable for single shot measurement.
SL_RTL_AOA_MODE_ONE_SHOT_FAST_RESPONSE_AZIMUTH_ONLY	Equivalent to ONE_SHOT_FAST_RESPONSE with low CPU cost, 1D angle. Most suitable for single shot measurement.
SL_RTL_AOA_MODE_ONE_SHOT_HIGH_ACCURACY_AZIMUTH_ONLY	Equivalent to ONE_SHOT_HIGH_ACCURACY with low CPU cost, 1D angle. Most suitable for single shot measurement.
SL_RTL_AOA_MODE_REAL_TIME_FAST_RESPONSE	Low filtering, fast response, lowest CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOA_MODE_REAL_TIME_BASIC	Medium filtering, medium response, medium CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOA_MODE_REAL_TIME_HIGH_ACCURACY	High filtering, slow response, highest CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOA_MODE_LAST	Placeholder.
SL_RTL_AOD_MODE_ONE_SHOT_BASIC	Medium filtering, medium response. Returns 2D angle, requires 10 rounds. Most suitable for single shot measurement.
SL_RTL_AOD_MODE_ONE_SHOT_BASIC_LIGHTWEIGHT	Medium filtering, medium response, low CPU cost & low elevation resolution. 2D angle, req. 10 rounds. Most suitable for single shot measurement.
SL_RTL_AOD_MODE_ONE_SHOT_FAST_RESPONSE	Low filtering, fast response, low CPU cost & low elevation resolution. 2D angle, requires 2 rounds. Most suitable for single shot measurement.
SL_RTL_AOD_MODE_ONE_SHOT_HIGH_ACCURACY	High filtering, slow response. 2D angle, requires 20 rounds. Most suitable for single shot measurement.
SL_RTL_AOD_MODE_ONE_SHOT_BASIC_AZIMUTH_ONLY	Equivalent to ONE_SHOT_BASIC with low CPU cost and returns 1D angle. Most suitable for single shot measurement.

SL_RTL_AOD_MODE_ONE_SHOT_FAST_RESPONSE_AZIMUTH_ONLY	Equivalent to ONE_SHOT_FAST_RESPONSE with low CPU cost, 1D angle. Most suitable for single shot measurement.
SL_RTL_AOD_MODE_ONE_SHOT_HIGH_ACCURACY_AZIMUTH_ONLY	Equivalent to ONE_SHOT_HIGH_ACCURACY with low CPU cost, 1D angle. Most suitable for single shot measurement.
SL_RTL_AOD_MODE_REAL_TIME_FAST_RESPONSE	Low filtering, fast response, lowest CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOD_MODE_REAL_TIME_BASIC	Medium filtering, medium response, medium CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOD_MODE_REAL_TIME_HIGH_ACCURACY	High filtering, slow response, highest CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOD_MODE_LAST	Placeholder.
SL_RTL_AOX_MODE_ONE_SHOT_BASIC	Medium filtering, medium response. Returns 2D angle, requires 10 rounds. Most suitable for single shot measurement.
SL_RTL_AOX_MODE_ONE_SHOT_BASIC_LIGHTWEIGHT	Medium filtering, medium response, low CPU cost & low elevation resolution. 2D angle, req. 10 rounds. Most suitable for single shot measurement.
SL_RTL_AOX_MODE_ONE_SHOT_FAST_RESPONSE	Low filtering, fast response, low CPU cost & low elevation resolution. 2D angle, requires 2 rounds. Most suitable for single shot measurement.
SL_RTL_AOX_MODE_ONE_SHOT_HIGH_ACCURACY	High filtering, slow response. 2D angle, requires 20 rounds. Most suitable for single shot measurement.
SL_RTL_AOX_MODE_ONE_SHOT_BASIC_AZIMUTH_ONLY	Equivalent to ONE_SHOT_BASIC with low CPU cost and returns 1D angle. Most suitable for single shot measurement.
SL_RTL_AOX_MODE_ONE_SHOT_FAST_RESPONSE_AZIMUTH_ONLY	Equivalent to ONE_SHOT_FAST_RESPONSE with low CPU cost, 1D angle. Most suitable for single shot measurement.
SL_RTL_AOX_MODE_ONE_SHOT_HIGH_ACCURACY_AZIMUTH_ONLY	Equivalent to ONE_SHOT_HIGH_ACCURACY with low CPU cost, 1D angle. Most suitable for single shot measurement.
SL_RTL_AOX_MODE_REAL_TIME_FAST_RESPONSE	Low filtering, fast response, lowest CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOX_MODE_REAL_TIME_BASIC	Medium filtering, medium response, medium CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOX_MODE_REAL_TIME_HIGH_ACCURACY	High filtering, slow response, highest CPU cost, 2D angle, Most suitable for real-time tracking.
SL_RTL_AOX_MODE_LAST	Placeholder.

Definition at line `123` of file `/mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h`

sl_rtl_aox_constraint_type

> sl_rtl_aox_constraint_type

Enumerator

SL_RTL_AOX_CONSTRAINT_TYPE_AZIMUTH	Azimuth constraint in degrees.
SL_RTL_AOX_CONSTRAINT_TYPE_ELEVATION	Elevation constraint in degrees.

Definition at line 175 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_switch_pattern_mode

> sl_rtl_aox_switch_pattern_mode

Enumerator

SL_RTL_AOX_SWITCH_PATTERN_MODE_DEFAULT	Internally defined switch pattern: 0, 1, 2, ..., N-1, where N is the number of antennas.
SL_RTL_AOX_SWITCH_PATTERN_MODE_RANDOM	Internally defined random switch pattern.
SL_RTL_AOX_SWITCH_PATTERN_MODE_EXTERNAL	Switch pattern set externally by the user.
SL_RTL_AOX_SWITCH_PATTERN_MODE_EXTRA_REFERENCE	Switch pattern set externally by the user with extra reference antenna as a first.

Definition at line 181 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_slib_iq_sample_qa_result_t

> sl_rtl_slib_iq_sample_qa_result_t

Enumerator

SL_RTL_AOX_IQ_SAMPLE_QA_INVAL_REF	Invalid reference period data.
SL_RTL_AOX_IQ_SAMPLE_QA_DCOFFSET	DC offset too large.
SL_RTL_AOX_IQ_SAMPLE_QA_SNDR	Reference period SNDR too large.
SL_RTL_AOX_IQ_SAMPLE_QA_ROTATING_ERROR	Rotation error too large.
SL_RTL_AOX_IQ_SAMPLE_QA_REF_ANT_PHASE_VALUE	Reference antenna phase value too big.
SL_RTL_AOX_IQ_SAMPLE_QA_REF_ANT_PHASE_JITTER	Reference antenna phase jitter too large.
SL_RTL_AOX_IQ_SAMPLE_QA_ANT_X_PHASE_JITTER	Antenna X phase jitter too large.
SL_RTL_AOX_IQ_SAMPLE_QA_ALL_SAME_PHASE	All the antennas seen in the same phase.
SL_RTL_AOX_IQ_SAMPLE_QA_SWICHING_JITTER	Switching jitter too large.

Definition at line 213 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Typedef Documentation

sl_rtl_aox_libitem

> typedef void* sl_rtl_aox_libitem

Angle of Arrival / Departure library item.

Definition at line 189 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_antenna_pattern

> typedef void* sl_rtl_aox_antenna_pattern

Angle of Arrival / Departure antenna array radiation pattern.

Definition at line 671 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Function Documentation

sl_rtl_aox_init

```
enum sl_rtl_error_code sl_rtl_aox_init (sl_rtl_aox_libitem *item)
```

Parameters

[in]	item	Pointer to the libitem to be initialized.

Initialize the AoX libitem instance.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 238 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_deinit

```
enum sl_rtl_error_code sl_rtl_aox_deinit (sl_rtl_aox_libitem *item)
```

Parameters

[in]	item	Pointer to the libitem to be deinitialized.

Deinitialize a libitem instance of the AoX estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 246 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_num_snapshots

```
enum sl_rtl_error_code sl_rtl_aox_set_num_snapshots (sl_rtl_aox_libitem *item, uint32_t num_snapshots)
```

Parameters

[in]	item	Pointer to the initialized AoX libitem.
[in]	num_snapshots	Number of snapshots as positive integer value.

Set the number of signal snapshots to be used in the angle estimation.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 257 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_array_type

enum sl_rtl_error_code sl_rtl_aox_set_array_type (sl_rtl_aox_libitem *item, enum sl_rtl_aox_array_type array_type)

Parameters

[in]	item	Pointer to the initialized AoX libitem.
[in]	array_type	Array type as sl_rtl_aox_array_type.

Set the array type used with the estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Set the array type used with the estimator. For example, the array type should be set to SL_RTL_AOX_ARRAY_TYPE_4x4_URA when using the reference 4x4 URA board.

Definition at line 270 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_mode

enum sl_rtl_error_code sl_rtl_aox_set_mode (sl_rtl_aox_libitem *item, enum sl_rtl_aox_mode mode)

Parameters

[in]	item	Pointer to the initialized AoX libitem.
[in]	mode	Estimator mode as sl_rtl_aox_mode.

Set the estimation mode.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Set the estimation mode. For example, ::SL_RTL_AOX_MODE_BASIC sets medium filtering and estimates both azimuth and elevation. For further description of the modes, see the documentation of sl_rtl_aox_mode.

Definition at line 283 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_calculate_iq_sample_phase_rotation

enum sl_rtl_error_code sl_rtl_aox_calculate_iq_sample_phase_rotation (sl_rtl_aox_libitem *item, float iq_data_downsampling_factor, float *i_samples, float *q_samples, uint32_t num_samples, float *phase_rotation_out)

Parameters

[in]	item	Pointer to the initialized AoX libitem.
[in]	iq_data_downsampling_factor	Ratio between reference period IQ-data sampling rate. and actual IQ-data (i.e., antenna array data) sampling rate. For example, 1e6 / 500e3 = 2.0
[in]	i_samples	Float-array of the reference period I samples.
[in]	q_samples	Float-array of the reference period Q samples.
[in]	num_samples	Number of samples, or size of the I or Q sample array.
[out]	phase_rotation_out	Returned phase rotation value as float in degrees.

Estimate the I/Q-sample phase rotation error in degrees.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Estimate the I/Q-sample phase rotation error caused by switching and CTE frequency error based on I/Q samples from the reference period.

Definition at line 301 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_iq_sample_phase_rotation

> enum sl_rtl_error_code sl_rtl_aox_set_iq_sample_phase_rotation (sl_rtl_aox_libitem *item, float phase_rotation)

Parameters

[in]	item	Pointer to the initialized AoX libitem.
[in]	phase_rotation	Float value of I/Q sample phase rotation in degrees.

Set a constant value of I/Q sample phase rotation in degrees.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Set a constant value of I/Q sample phase rotation in degrees. The sample rotation can be used to correct the switching and CTE frequency error. **Note**

- : This function should be called only after the estimator is created.

Definition at line 315 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_add_constraint

> enum sl_rtl_error_code sl_rtl_aox_add_constraint (sl_rtl_aox_libitem *item, enum sl_rtl_aox_constraint_type constraint_type, float min_value, float max_value)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	constraint_type	Select which constraints should be added.
[in]	min_value	Starting (minimum) value of the range including the min_value.
[in]	max_value	Ending (maximum) value of the range including the max_value.

Add constraints for the estimator. Call before sl_rtl_aox_create_estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

For example, constraint for SL_RTL_AOX_CONSTRAINT_TYPE_AZIMUTH with min_value = 0 and max_value = 90 means that the angular range 0 to 90 degrees is excluded from the estimators internal processing range and angle estimated between that range are not considered as valid results. Setting constraints will help ruling out false multipath-detections when the locator is installed nearby a wall or a RF-reflective surface. **Note**

- : This function must called before the estimator is created.

Definition at line 334 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_sample_rate

enum sl_rtl_error_code sl_rtl_aox_set_sample_rate (sl_rtl_aox_libitem *item, float sampleRate)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	sampleRate	The new sample rate.

Configure the IQ sampling sample rate for the library.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 343 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_num_radio_channels

enum sl_rtl_error_code sl_rtl_aox_set_num_radio_channels (sl_rtl_aox_libitem *item, uint32_t channels)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	channels	The new number of radio channels.

Configure the number of radio channels.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 352 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_iq_sample_qa_configure

enum sl_rtl_error_code sl_rtl_aox_iq_sample_qa_configure (sl_rtl_aox_libitem *item)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.

Configure the IQ sample quality analysis.

Returns

- SL_RTL_ERROR_SUCCESS if successful

This function turns the IQ sample quality analysis, which is not done by default.

Definition at line 363 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_iq_sample_qa_get_results

uint32_t sl_rtl_aox_iq_sample_qa_get_results (sl_rtl_aox_libitem *item)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.

Get the IQ sample quality analysis overall results.

Returns

- bitmask of found problems, zero indicates that everything is OK

Definition at line 371 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_iq_sample_qa_get_details

enum sl_rtl_error_code sl_rtl_aox_iq_sample_qa_get_details (sl_rtl_aox_libitem *item, sl_rtl_clib_iq_sample_qa_dataset_t *results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[out]	results	The data structure with data related to the latest data packet.
[out]	antenna_data	The array of antenna-specific results.

Get the IQ sample quality analysis detailed results for the latest packet.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Fetch the results for the latest packet.

Definition at line 383 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_iq_sample_qa_get_channel_details

enum sl_rtl_error_code sl_rtl_aox_iq_sample_qa_get_channel_details (sl_rtl_aox_libitem *item, uint8_t channel, sl_rtl_clib_iq_sample_qa_dataset_t *results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	channel	Radio channel to show results for.
[out]	results	The data structure with data related to the last data packet using the requested channel.
[out]	antenna_data	The array of antenna-specific results

Get the IQ sample quality analysis detailed results for the requested radio channel.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Fetch the latest results for the given radio channel, which may be other than the latest packet received.

Definition at line 397 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_create_estimator

```
enum sl_rtl_error_code sl_rtl_aox_create_estimator (sl_rtl_aox_libitem *item)
```

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.

Create the estimator after initializing the libitem and setting parameters.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 405 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_convert_raw_samples

```
enum sl_rtl_error_code sl_rtl_aox_convert_raw_samples (sl_rtl_aox_libitem *item, uint32_t start_offset, float
iq_data_downsampling_factor, float *raw_i_samples_in, float *raw_q_samples_in, uint32_t num_raw_samples_in, float
**i_samples_out, float **q_samples_out, uint32_t num_snapshots_out)
```

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	start_offset	The start offset in the buffer (for example, for skipping a reference period).
[in]	iq_data_downsampling_factor	Ratio between chip IQ-data sampling rate and downsampled rate. For example, 4.8e6 / 500e3 = 9.6
[in]	raw_i_samples_in	I-part sample buffer input
[in]	raw_q_samples_in	Q-part sample buffer input
[in]	num_raw_samples_in	Number of IQ-sample pairs in the input buffers.
[out]	i_samples_out	Buffer for the processed I-samples. Must be allocated by the user. Indexing: i_samples_out[snapshot][antenna]
[out]	q_samples_out	Buffer for the processed Q-samples. Must be allocated by the user. Indexing: q_samples_out[snapshot][antenna]
[in]	num_snapshots_out	Number of snapshots allocated in the output buffers. This function checks if the given number of snapshots and calculated snapshots based on the length of RAW-data match

Convert data in RAW IQ-data buffers to downsampled IQ-data buffers.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 425 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_calculate_number_of_snapshots

```
enum sl_rtl_error_code sl_rtl_aox_calculate_number_of_snapshots (sl_rtl_aox_libitem *item, uint32_t num_raw_samples_in,
uint32_t start_offset, float iq_data_downsampling_factor, uint32_t num_channels, uint32_t *num_snapshots_out)
```

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	num_raw_samples_in	Total number of RAW IQ-sample pairs.
[in]	start_offset	The start offset in the buffer, for example, for skipping a reference period.
[in]	iq_data_downsampling_factor	Ratio between chip IQ-data sampling rate and downsampled rate. For example, 4.8e6 / 500e3 = 9.6.
[in]	num_channels	Number of channels in the RAW data
[out]	num_snapshots_out	Calculated number of snapshots based on the input.

Calculate the number of downsampled snapshots in a RAW IQ-data buffer. Use this function to get the number of snapshots to allocate the i_samples and q_samples buffers for the process-function.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 440 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_switch_pattern_mode

> enum sl_rtl_error_code sl_rtl_aox_set_switch_pattern_mode (sl_rtl_aox_libitem *item, enum sl_rtl_aox_switch_pattern_mode mode)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	mode	Required mode.

Set switch pattern mode. Sets internal mode used by the library. See enum sl_rtl_aox_switch_pattern_mode for detailed description of the modes. If this function isn't called, a default switch pattern of: 0, 1, 2, 3, ..., N-1 will be used, where N is the number of antennas. Call this function at run-time after initializing and creating the estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 452 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_update_switch_pattern

> enum sl_rtl_error_code sl_rtl_aox_update_switch_pattern (sl_rtl_aox_libitem *item, uint32_t *switch_pattern_in, uint32_t **switch_pattern_out)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	switch_pattern_in	Pointer to the switch pattern array. The pointer must point to a valid switch pattern when SL_RTL_AOX_SWITCH_PATTERN_MODE_EXTERNAL is used.
[out]	switch_pattern_out	Pointer to the user defined uint32_t* variable. This pointer must be non-zero when using internally defined switch pattern modes, otherwise can be zero.

Update switch pattern, which is used by the estimator algorithm. This function must be called before calling the function sl_rtl_aox_process. Call this function at run-time after initializing and creating the estimator.

When using a custom pattern in SL_RTL_AOX_SWITCH_PATTERN_MODE_EXTERNAL mode, the pattern in switch_pattern_in must not contain duplicate elements and the antenna indices must be within the range 0 to (number_of_antennas - 1). Number_of_antennas is the amount of antennas in the set array type, for example for the 4x4 URA this is 16 and for the 1x4 ULA this is 4. The length of the switching pattern must be equal to number_of_antennas.

Note: The RF switch pattern indices may be different to the indices required by the RTL algorithm. For example, a 1x4 ULA might use RF controls [12, 13, 14, 15] for switching the antennas 1-4, but the RTL algorithm requires the switch pattern [0, 1, 2, 3]. If the RF controls are switched as [14, 15, 13, 12], the RTL algorithm requires the pattern [2, 3, 1, 0].

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 479 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_switch_pattern_seed

> enum sl_rtl_error_code sl_rtl_aox_set_switch_pattern_seed (sl_rtl_aox_libitem *item, int32_t seed_value)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	seed_value	The seed value to be used.

Set the random seed for the switch pattern, which is used for the random pattern mode. Call this function at run-time after initializing and creating the estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 489 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_convert_switch_pattern

> enum sl_rtl_error_code sl_rtl_aox_convert_switch_pattern (sl_rtl_aox_libitem *item, uint32_t array_id, uint32_t options, uint32_t switch_pattern_size_in, uint32_t *switch_pattern_in, uint32_t *switch_pattern_size_out, uint32_t **switch_pattern_out)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	array_id	Array ID, see sl_rtl_aox_switch_pattern_array, for example SL_RTL_AOX_SWITCH_PATTERN_ARRAY_4x4_DP
[in]	options	Switch pattern options, see sl_rtl_aox_switch_pattern_options. Can be used to set for example dual polarization (default is single polarization).
[in]	switch_pattern_size_in	The length of the input switching pattern.
[in]	switch_pattern_in	The input switching pattern.
[out]	switch_pattern_size_out	The outputted length of the converted switching pattern.
[out]	switch_pattern_out	The outputted switching pattern.

Convert switching pattern from antenna indices to antenna switching pin control logic. Call this function at run-time after initializing and creating the estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful
- SL_RTL_ERROR_SUCCESS if successful

Definition at line 507 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_reset_estimator

> enum sl_rtl_error_code sl_rtl_aox_reset_estimator (sl_rtl_aox_libitem *item)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.

Reset estimator state. Calling this function causes the selected aox-mode to start from its initial state. Call this function at run-time after initializing and creating the estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 516 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_process

> enum sl_rtl_error_code sl_rtl_aox_process (sl_rtl_aox_libitem *item, float **i_samples, float **q_samples, float tone_frequency, float *az_out, float *el_out)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	i_samples	Two-dimensional float-array of captured I samples as i_samples[snapshot][antenna].
[in]	q_samples	Two-dimensional float-array of captured Q samples as q_samples[snapshot][antenna], corresponding to the I samples array.
[in]	tone_frequency	The frequency of the signal from which the I/Q data was captured from as float (e.g., 2.46e9f).
[out]	az_out	Output azimuth angle.
[out]	el_out	Output elevation angle.

Calculate the angle estimate.

Returns

- SL_RTL_ERROR_SUCCESS if successful and SL_RTL_ERROR_ESTIMATION_IN_PROGRESS if estimate is not yet final and more I/Q data needs to be processed.

Calculate the angle estimate from the given I/Q samples captured at the given frequency. Call this function with new I/Q data as many times as indicated by the sl_rtl_aox_mode used by the estimator before the final estimate is output.

Definition at line 539 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_get_latest_aox_standard_deviation

> enum sl_rtl_error_code sl_rtl_aox_get_latest_aox_standard_deviation (sl_rtl_aox_libitem *item, float *az_std_dev, float *el_std_dev)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[out]	az_std_dev	Pointer for getting standard deviation of the latest azimuth estimate.
[out]	el_std_dev	Pointer for getting standard deviation of the latest elevation estimate.

Get the standard deviation for the latest AoA/AoD-estimate.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Call this function after sl_rtl_aox_process to fetch standard deviation for the latest AoA/AoD-estimate. Positive standard deviations indicate line-of-sight detection and negative values indicate likely non-line-of-sight detection.

Definition at line 556 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_get_latest_aoa_standard_deviation

> static enum sl_rtl_error_code sl_rtl_aox_get_latest_aoa_standard_deviation (sl_rtl_aox_libitem *item, float *az_std_dev, float *el_std_dev)

Parameters

N/A	item	
N/A	az_std_dev	
N/A	el_std_dev	

Kept for backward compatibility

Definition at line 561 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_get_latest_aod_standard_deviation

> static enum sl_rtl_error_code sl_rtl_aox_get_latest_aod_standard_deviation (sl_rtl_aox_libitem *item, float *az_std_dev, float *el_std_dev)

Parameters

N/A	item	
N/A	az_std_dev	
N/A	el_std_dev	

Kept for backward compatibility

Definition at line 569 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_expected_direction

> enum sl_rtl_error_code sl_rtl_aox_set_expected_direction (sl_rtl_aox_libitem *item, float expected_az, float expected_el)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.

[in]	expected_az	Expected azimuth angle calculated by position algorithm.
[in]	expected_el	Expected elevation angle calculated by position algorithm.

Feed the expected angles back to the locator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

The position algorithm has a more complete view of the asset's location. The direction it should be found can be calculated back and fed to the locator, so that it can recover faster when, for example, it has locked to a reflection rather than to the line of sight signal. See also sl_rtl_loc_get_expected_direction().

Definition at line 588 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_expected_deviation

> enum sl_rtl_error_code sl_rtl_aox_set_expected_deviation (sl_rtl_aox_libitem *item, float deviation_az, float deviation_el)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.
[in]	deviation_az	Deviation of the expected azimuth angle calculated by position algorithm.
[in]	deviation_el	Deviation of the expected elevation angle calculated by position algorithm.

Feed the expected angle deviations back to the locator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Report expected deviations calculated by the position algorithm back to the locator so that angles are calculated more accurately taking in account the correctness of the expected directions. If sl_rtl_aox_set_expected_direction() is called but this function is not called, the algorithm will use default values for the expected deviations. See also sl_rtl_loc_get_expected_deviation() and sl_rtl_aox_set_expected_direction().

Definition at line 605 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_clear_expected_direction

> enum sl_rtl_error_code sl_rtl_aox_clear_expected_direction (sl_rtl_aox_libitem *item)

Parameters

[in]	item	Pointer to the initialized and configured AoX libitem.

Clear the expected directions and deviations from the locator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

See also sl_rtl_aox_set_expected_direction() and sl_rtl_aox_set_expected_deviation().

Definition at line 615 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_enable_spectrum

> enum sl_rtl_error_code sl_rtl_aox_enable_spectrum (sl_rtl_aox_libitem *item, bool enable)

Parameters

[in]	item	Pointer to the initialized libitem.
[in]	enable	Set to true to enable outputting the pseudospectrum.

Enable or disable outputting the pseudospectrum. This function should be called before creating the estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 625 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_get_spectrum_size

> enum sl_rtl_error_code sl_rtl_aox_get_spectrum_size (sl_rtl_aox_libitem *item, uint32_t *rows, uint32_t *cols)

Parameters

[in]	item	Pointer to the initialized libitem.
[out]	rows	Number of rows in the pseudospectrum data array.
[out]	cols	Number of columns in the pseudospecutrm data array.

Get the size of the pseudospectrum. This function should be called after calling the AoX process function.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 636 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_get_polarization_spectrum_size

> enum sl_rtl_error_code sl_rtl_aox_get_polarization_spectrum_size (sl_rtl_aox_libitem *item, uint32_t *rows, uint32_t *cols)

Parameters

[in]	item	Pointer to the initialized libitem.
[out]	rows	Number of rows in the polarization pseudospectrum data array.
[out]	cols	Number of columns in the pseudospecutrm data array.

Get the size of the polarization pseudospectrum. This function should be called after calling the AoX process function.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 648 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_get_spectrum

enum sl_rtl_error_code sl_rtl_aox_get_spectrum (sl_rtl_aox_libitem *item, float **spectrum_out)

Parameters

[in]	item	Pointer to the initialized libitem.
[out]	spectrum_out	Outputted pseudospectrum.

Get the pseudospectrum. Memory for the output spectrum must be allocated by the user, the size of the rows and columns are given by sl_rtl_aox_get_spectrum_size().

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 658 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_get_polarization_spectrum

enum sl_rtl_error_code sl_rtl_aox_get_polarization_spectrum (sl_rtl_aox_libitem *item, float **spectrum_out)

Parameters

[in]	item	Pointer to the initialized libitem.
[out]	spectrum_out	Outputted polarization pseudospectrum.

Get the polarization spectrum. Memory for the output spectrum must be allocated by the user, the size of the rows and columns are given by sl_rtl_aox_get_polarization_spectrum_size().

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 668 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_antenna_pattern_init

enum sl_rtl_error_code sl_rtl_aox_antenna_pattern_init (sl_rtl_aox_antenna_pattern *pattern, enum sl_rtl_aox_array_type array_type)

Parameters

[in]	pattern	Pointer to the pattern item to initialize.
[in]	array_type	Array type of the board for which the pattern is loaded.

Initialize the antenna radiation pattern item for the given array board. The antenna pattern item can be shared by multiple different estimators.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 681 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_set_antenna_pattern

```
enum sl_rtl_error_code sl_rtl_aox_set_antenna_pattern (sl_rtl_aox_libitem *item, sl_rtl_aox_antenna_pattern *pattern)
```

Parameters

[in]	item	Pointer to the initialized libitem.
[in]	pattern	Pointer to the initialized antenna pattern item.

Set the given estimator to use the given antenna radiation pattern in the model. The same antenna pattern can be shared between multiple estimators. The antenna pattern can be cleared for an estimator by setting it to NULL. The estimator will then use the default model. **Note**

- : This function should be called only after the estimator is created.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 695 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_aox_antenna_pattern_deinit

```
enum sl_rtl_error_code sl_rtl_aox_antenna_pattern_deinit (sl_rtl_aox_antenna_pattern *pattern)
```

Parameters

[in]	pattern	Pointer to the initialized pattern item.

Deinitialize the antenna radiation pattern item.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 703 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Macro Definition Documentation

SL_RTL_AOX_IQ_SAMPLE_QA_ALL_OK

```
#define SL_RTL_AOX_IQ_SAMPLE_QA_ALL_OK
```

Value:

```
0
```

Definition at line 211 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_AOX_IQ_SAMPLE_QA_FAILURE

```
#define SL_RTL_AOX_IQ_SAMPLE_QA_FAILURE
```

Value:

```
0xffffffff
```

Definition at line 212 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_AOX_IQ_SAMPLE_QA_CLEAR_BIT

```
#define SL_RTL_AOX_IQ_SAMPLE_QA_CLEAR_BIT
```

Value:

```
(code, bit)
```

Definition at line 225 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_AOX_IQ_SAMPLE_QA_SET_BIT

```
#define SL_RTL_AOX_IQ_SAMPLE_QA_SET_BIT
```

Value:

```
(code, bit)
```

Definition at line 227 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_AOX_IQ_SAMPLE_QA_IS_SET

```
#define SL_RTL_AOX_IQ_SAMPLE_QA_IS_SET
```

Value:

```
(code, bit)
```

Definition at line 229 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_clib_iq_sample_qa_antenna_data_t

You are viewing documentation for version: 8.2.0 | Version History

Public Attributes

float	level Antenna signal level, in decibels.
float	snr Antenna level signal to noise ratio, in decibels.
float	phase_value Antenna's average unrotated phase value in the packet, in radians.
float	phase_jitter Phase variation of snapshots' data of an antenna in packet, in radians.

Public Attribute Documentation

level

> float sl_rtl_clib_iq_sample_qa_antenna_data_t::level

Antenna signal level, in decibels.

Definition at line 193 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

snr

> float sl_rtl_clib_iq_sample_qa_antenna_data_t::snr

Antenna level signal to noise ratio, in decibels.

Definition at line 194 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

phase_value

> float sl_rtl_clib_iq_sample_qa_antenna_data_t::phase_value

Antenna's average unrotated phase value in the packet, in radians.

Definition at line 195 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

phase_jitter

> float sl_rtl_clib_iq_sample_qa_antenna_data_t::phase_jitter

Phase variation of snapshots' data of an antenna in packet, in radians.

Definition at line 196 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_clib_iq_sample_qa_dataset_t

You are viewing documentation for version: 8.2.0 | Version History

Public Attributes

bool	data_available If false, all the antenna values are undefined.
uint32_t	curr_channel Radio channel for the last packet.
float	ref_freq Apparent supplemental tone frequency.
float	ref_sndr Reference period signal to noise and distortion ratio.
float	switching_jitter Estimated antenna switching clock jitter.

Public Attribute Documentation

data_available

> bool sl_rtl_clib_iq_sample_qa_dataset_t::data_available

If false, all the antenna values are undefined.

Definition at line 201 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

curr_channel

> uint32_t sl_rtl_clib_iq_sample_qa_dataset_t::curr_channel

Radio channel for the last packet.

Definition at line 202 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

ref_freq

> float sl_rtl_clib_iq_sample_qa_dataset_t::ref_freq

Apparent supplemental tone frequency.

Definition at line 203 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

ref_sndr

> float sl_rtl_clib_iq_sample_qa_dataset_t::ref_sndr

Reference period signal to noise and distortion ratio.

Definition at line 204 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

switching_jitter

> float sl_rtl_clib_iq_sample_qa_dataset_t::switching_jitter

Estimated antenna switching clock jitter.

Definition at line 207 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Channel Sounding (CS)

You are viewing documentation for version: 8.2.0 | Version History

Channel Sounding (CS)

Channel Sounding.

Distances are calculated by following the steps below:

1. Initialize a sl_rtl_cs_libitem instance.
2. Set up the algorithm mode and CS parameters.
3. Create the estimator.
4. Process the CS procedure data into distance.
5. Get distance and/or likeliness estimates.

Enumerations

enum sl_rtl_cs_role {

SL_RTL_CS_ROLE_INITIATOR = 0
SL_RTL_CS_ROLE_REFLECTOR

}

enum sl_rtl_cs_algo_mode {

SL_RTL_CS_ALGO_MODE_REAL_TIME_BASIC = 0
SL_RTL_CS_ALGO_MODE_STATIC_HIGH_ACCURACY = 1

}

enum sl_rtl_cs_mode {

SL_RTL_CS_MODE_CALIBRATION = 0
SL_RTL_CS_MODE_RTT = 1
SL_RTL_CS_MODE_PBR = 2

}

enum sl_rtl_cs_rtt_type {

SL_RTL_CS_RTT_TYPE_AAONLY = 0
SL_RTL_CS_RTT_TYPE_SS_32BIT = 1
SL_RTL_CS_RTT_TYPE_SS_96BIT = 2

}

enum sl_rtl_cs_estimator_param_type {

SL_RTL_LAST_KNOWN_DISTANCE = 0
SL_RTL_RANGE_MIN = 1
SL_RTL_RANGE_MAX = 2
SL_RTL_REF_TX_POWER = 3

}

enum sl_rtl_cs_distance_estimate_type {

SL_RTL_CS_DISTANCE_ESTIMATE_TYPE_FILTERED = 0
SL_RTL_CS_DISTANCE_ESTIMATE_TYPE_RSSI

}

enum sl_rtl_cs_distance_estimate_confidence_type {

SL_RTL_CS_DISTANCE_ESTIMATE_CONFIDENCE_TYPE_LIKELINESS = 0
SL_RTL_CS_DISTANCE_ESTIMATE_CONFIDENCE_TYPE_BIT_ERROR_RATE

}

enum sl_rtl_cs_distance_estimate_extended_info_type {

SL_RTL_CS_DISTANCE_ESTIMATE_EXTENDED_INFO_TYPE_PROGRESS_PERCENTAGE = 0

}

Typedefs

typedef void * sl_rtl_cs_libitem

CS library item.

Functions

typedef(struct { uint32_t max_number_of_frequencies;uint32_t delta_f;}) sl_rtl_cs_params

typedef(struct { uint8_t num_antenna_paths;uint8_t channel_map_repetition;uint32_t main_mode_repetition;uint8_t num_calib_steps;uint8_t T_PM_time;uint8_t T_IP1_time;uint8_t T_IP2_time;uint8_t T_FCS_time;uint8_t channel_selection_type;uint8_t ch3c_shape;uint8_t ch3c_jump;uint32_t subevent_len;uint8_t subevents_per_event;uint16_t subevent_interval;uint16_t event_interval;uint16_t procedure_interval;uint16_t procedure_count;uint8_t cs_sync_phy;sl_rtl_cs_rtt_type rtt_type :8;uint16_t acl_connection_interval;}) sl_rtl_cs_config

typedef(union { float last_known_distance;float range_min;float range_max;float ref_tx_power;}) sl_rtl_cs_estimator_param_value

typedef(struct { sl_rtl_cs_estimator_param_type type :8;sl_rtl_cs_estimator_param_value value;}) sl_rtl_cs_estimator_param

typedef(struct { uint8_t packet_quality;uint8_t packet_rssi;uint8_t packet_antenna;int16_t measured_freq_offset;}) sl_rtl_cs_mode_calib_data

typedef(struct { uint8_t packet_quality;uint8_t packet_nadm;int8_t packet_rssi;int16_t packet_rtt;uint8_t packet_antenna;}) sl_rtl_cs_mode_rtt_data

typedef(struct { uint8_t antenna_permutation_index;1(struct tone { int32_t pct_i:SL_RTL_CS_PCT_IQ_BIT_LEN;int32_t pct_q:SL_RTL_CS_PCT_IQ_BIT_LEN;uint8_t quality_indicator;}) tones[SL_RTL_CS_TONE_ARRAY_LEN];}) sl_rtl_cs_mode_pbr_data

typedef(struct { sl_rtl_cs_mode step_mode :SL_RTL_CS_STEP_MODE_BIT_LEN;uint8_t step_channel;uint8_t step_data_length;1(union { sl_rtl_cs_mode_calib_data mode_calib_step;sl_rtl_cs_mode_rtt_data mode_rtt_step;sl_rtl_cs_mode_pbr_data mode_pbr_step;});}) sl_rtl_cs_step_data

typedef(struct { uint64_t subevent_timestamp_ms;int16_t frequency_compensation;int8_t reference_power_level;uint8_t num_steps;uint16_t step_data_count;uint8_t *step_data;}) sl_rtl_cs_subevent_data

typedef(struct { sl_rtl_cs_config cs_config;uint8_t initiator_subevent_data_count;sl_rtl_cs_subevent_data *initiator_subevent_data;uint8_t reflector_subevent_data_count;sl_rtl_cs_subevent_data *reflector_subevent_data;}) sl_rtl_cs_procedure

typedef(struct { struct timespec timestamp;uint64_t timestamp_ms;}) sl_rtl_cs_log_params

enum sl_rtl_error_code sl_rtl_cs_init(sl_rtl_cs_libitem *item)

enum sl_rtl_error_code sl_rtl_cs_deinit(sl_rtl_cs_libitem *item)

enum sl_rtl_error_code — sl_rtl_cs_set_algo_mode(sl_rtl_cs_libitem *item, const sl_rtl_cs_algo_mode mode)

enum sl_rtl_error_code — sl_rtl_cs_set_cs_mode(sl_rtl_cs_libitem *item, const sl_rtl_cs_mode cs_mode)

enum sl_rtl_error_code — sl_rtl_cs_set_cs_params(sl_rtl_cs_libitem *item, const sl_rtl_cs_params *parameters)

enum sl_rtl_error_code — sl_rtl_cs_create_estimator(sl_rtl_cs_libitem *item)

enum sl_rtl_error_code — sl_rtl_cs_set_estimator_param(sl_rtl_cs_libitem *item, const sl_rtl_cs_estimator_param *param)

enum sl_rtl_error_code — sl_rtl_cs_process(sl_rtl_cs_libitem *item, const uint8_t procedure_count, const sl_rtl_cs_procedure *procedure_data)

enum sl_rtl_error_code — sl_rtl_cs_get_distance_estimate(sl_rtl_cs_libitem *item, const sl_rtl_cs_distance_estimate_type estimate_type, float *distance)

enum sl_rtl_error_code — sl_rtl_cs_get_distance_estimate_confidence(sl_rtl_cs_libitem *item, const sl_rtl_cs_distance_estimate_confidence_type confidence_type, float *confidence)

enum sl_rtl_error_code — sl_rtl_cs_get_distance_estimate_extended_info(sl_rtl_cs_libitem *item, const sl_rtl_cs_distance_estimate_extended_info_type extended_info_type, float *extended_info)

enum sl_rtl_error_code — sl_rtl_cs_log_enable(sl_rtl_cs_libitem *item, const sl_rtl_cs_log_params *log_params)

enum sl_rtl_error_code — sl_rtl_cs_log_disable(sl_rtl_cs_libitem *item)

uint8_t — sl_rtl_cs_log_get_instance_id(sl_rtl_cs_libitem *item)

Macros

#define SL_RTL_CS_PCT_IQ_BIT_LEN 12

#define SL_RTL_CS_STEP_MODE_BIT_LEN 8

#define SL_RTL_CS_TONE_ARRAY_LEN 2

Enumeration Documentation

sl_rtl_cs_role

> sl_rtl_cs_role

Enumerator	
SL_RTL_CS_ROLE_INITIATOR	
SL_RTL_CS_ROLE_REFLECTOR	

Definition at line 752 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_algo_mode

> sl_rtl_cs_algo_mode

Enumerator	
SL_RTL_CS_ALGO_MODE_REAL_TIME_BASIC	
SL_RTL_CS_ALGO_MODE_STATIC_HIGH_ACCURACY	

Definition at line 757 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_mode

> sl_rtl_cs_mode

Enumerator	
SL_RTL_CS_MODE_CALIBRATION	
SL_RTL_CS_MODE_RTT	
SL_RTL_CS_MODE_PBR	

Definition at line 766 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_rtt_type

> sl_rtl_cs_rtt_type

Enumerator	
SL_RTL_CS_RTT_TYPE_AAONLY	
SL_RTL_CS_RTT_TYPE_SS_32BIT	
SL_RTL_CS_RTT_TYPE_SS_96BIT	

Definition at line 775 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_estimator_param_type

> sl_rtl_cs_estimator_param_type

Enumerator	
SL_RTL_LAST_KNOWN_DISTANCE	
SL_RTL_RANGE_MIN	
SL_RTL_RANGE_MAX	
SL_RTL_REF_TX_POWER	

Definition at line 822 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_distance_estimate_type

> sl_rtl_cs_distance_estimate_type

Enumerator	
SL_RTL_CS_DISTANCE_ESTIMATE_TYPE_FILTERED	
SL_RTL_CS_DISTANCE_ESTIMATE_TYPE_RSSI	

Definition at line 918 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_distance_estimate_confidence_type

> sl_rtl_cs_distance_estimate_confidence_type

Enumerator

SL_RTL_CS_DISTANCE_ESTIMATE_CONFIDENCE_TYPE_LIKELINESS	
SL_RTL_CS_DISTANCE_ESTIMATE_CONFIDENCE_TYPE_BIT_ERROR_RATE	

Definition at line 923 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_distance_estimate_extended_info_type

> sl_rtl_cs_distance_estimate_extended_info_type

Enumerator

SL_RTL_CS_DISTANCE_ESTIMATE_EXTENDED_INFO_TYPE_PROGRESS_PERCENTAGE	

Definition at line 935 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Typedef Documentation

sl_rtl_cs_libitem

> typedef void* sl_rtl_cs_libitem

CS library item.

Definition at line 942 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Function Documentation

typedef

> typedef (struct { uint32_t max_number_of_frequencies;uint32_t delta_f;}) sl_rtl_cs_params

Parameters

N/A		

Definition at line 781 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

> typedef (struct { uint8_t num_antenna_paths;uint8_t channel_map_repetition;uint32_t main_mode_repetition;uint8_t num_calib_steps;uint8_t T_PM_time;uint8_t T_IP1_time;uint8_t T_IP2_time;uint8_t T_FCS_time;uint8_t channel_selection_type;uint8_t ch3c_shape;uint8_t ch3c_jump;uint32_t subevent_len;uint8_t subevents_per_event;uint16_t subevent_interval;uint16_t event_interval;uint16_t procedure_interval;uint16_t procedure_count;uint8_t cs_sync_phy;sl_rtl_cs_rtt_type rtt_type :8;uint16_t acl_connection_interval;}) sl_rtl_cs_config

Parameters

N/A		

Definition at line 788 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

```
typedef (union { float last_known_distance;float range_min;float range_max;float ref_tx_power;})
sl_rtl_cs_estimator_param_value
```

Parameters

N/A		

Definition at line 835 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

```
typedef (struct { sl_rtl_cs_estimator_param_type type :8;sl_rtl_cs_estimator_param_value value;}) sl_rtl_cs_estimator_param
```

Parameters

N/A		

Definition at line 842 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

```
typedef (struct { uint8_t packet_quality;uint8_t packet_rssi;uint8_t packet_antenna;int16_t measured_freq_offset;})
sl_rtl_cs_mode_calib_data
```

Parameters

N/A		

Definition at line 847 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

```
typedef (struct { uint8_t packet_quality;uint8_t packet_nadm;int8_t packet_rssi;int16_t packet_rtt;uint8_t packet_antenna;})
sl_rtl_cs_mode_rtt_data
```

Parameters

N/A		

Definition at line 854 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

typedef (struct { uint8_t antenna_permutation_index;1(struct tone { int32_t pct_i:SL_RTL_CS_PCT_IQ_BIT_LEN;int32_t pct_q:SL_RTL_CS_PCT_IQ_BIT_LEN;uint8_t quality_indicator;}) tones[SL_RTL_CS_TONE_ARRAY_LEN];}) sl_rtl_cs_mode_pbr_data

Parameters

N/A		

Definition at line 862 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

typedef (struct { sl_rtl_cs_mode step_mode :SL_RTL_CS_STEP_MODE_BIT_LEN;uint8_t step_channel;uint8_t step_data_length;1(union { sl_rtl_cs_mode_calib_data mode_calib_step;sl_rtl_cs_mode_rtt_data mode_rtt_step;sl_rtl_cs_mode_pbr_data mode_pbr_step;});}) sl_rtl_cs_step_data

Parameters

N/A		

Definition at line 871 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

typedef (struct { uint64_t subevent_timestamp_ms;int16_t frequency_compensation;int8_t reference_power_level;uint8_t num_steps;uint16_t step_data_count;uint8_t *step_data;}) sl_rtl_cs_subevent_data

Parameters

N/A		

Definition at line 883 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

typedef (struct { sl_rtl_cs_config cs_config;uint8_t initiator_subevent_data_count;sl_rtl_cs_subevent_data *initiator_subevent_data;uint8_t reflector_subevent_data_count;sl_rtl_cs_subevent_data *reflector_subevent_data;}) sl_rtl_cs_procedure

Parameters

N/A		

Definition at line 910 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

typedef

typedef (struct { struct timespec timestamp;uint64_t timestamp_ms;}) sl_rtl_cs_log_params

Parameters

N/A		

Definition at line 944 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_init

```
enum sl_rtl_error_code sl_rtl_cs_init (sl_rtl_cs_libitem *item)
```

Parameters

[in]	item	Pointer to the libitem to be initialized

Initialize the CS libitem instance.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 972 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_deinit

```
enum sl_rtl_error_code sl_rtl_cs_deinit (sl_rtl_cs_libitem *item)
```

Parameters

[in]	item	Pointer to the libitem to be deinitialized

Deinitialize a libitem instance of the CS estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 980 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_set_algo_mode

```
enum sl_rtl_error_code sl_rtl_cs_set_algo_mode (sl_rtl_cs_libitem *item, const sl_rtl_cs_algo_mode mode)
```

Parameters

[in]	item	Pointer to the initialized CS libitem
[in]	mode	Estimator mode as sl_rtl_cs_algo_mode

Set the estimation algorithm mode.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Set the estimation algorithm mode. For further description of the modes, see the documentation of::sl_rtl_cs_algo_mode. CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called before estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_ALREADY_CREATED.

Definition at line 997 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_set_cs_mode

> enum sl_rtl_error_code sl_rtl_cs_set_cs_mode (sl_rtl_cs_libitem *item, const sl_rtl_cs_mode cs_mode)

Parameters

[in]	item	Pointer to the initialized CS libitem
[in]	cs_mode	cs_mode as sl_rtl_cs_mode

Set the estimation CS mode.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Set the desired cs_mode. For example, SL_RTL_CS_MODE_PBR uses round-trip phase based cs_mode. CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called before estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_ALREADY_CREATED.

Definition at line 1014 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_set_cs_params

> enum sl_rtl_error_code sl_rtl_cs_set_cs_params (sl_rtl_cs_libitem *item, const sl_rtl_cs_params *parameters)

Parameters

[in]	item	Pointer to the initialized CS libitem
[in]	parameters	cs_params as ::sl_rtl_cs_params

Set the estimation CS parameters.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Set the parameters for the estimator. CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called before estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_ALREADY_CREATED.

Definition at line 1031 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_create_estimator

> enum sl_rtl_error_code sl_rtl_cs_create_estimator (sl_rtl_cs_libitem *item)

Parameters

[in]	item	Pointer to the initialized and configured CS libitem

Create an estimator with the given parameters after initializing the libitem.

Returns

- SL_RTL_ERROR_SUCCESS if successful

CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called before estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_ALREADY_CREATED.

Definition at line 1046 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_set_estimator_param

> enum sl_rtl_error_code sl_rtl_cs_set_estimator_param (sl_rtl_cs_libitem *item, const sl_rtl_cs_estimator_param *param)

Parameters

[in]	item	Pointer to the initialized and configured CS libitem
[in]	param	Data structure with the CS algorithm parameter to be set

Set CS algorithm parameters. The function can be called any time before or during the processing calls.

Returns

- SL_RTL_ERROR_SUCCESS if successful

CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called after estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_NOT_CREATED.

Definition at line 1063 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_process

> enum sl_rtl_error_code sl_rtl_cs_process (sl_rtl_cs_libitem *item, const uint8_t procedure_count, const sl_rtl_cs_procedure *procedure_data)

Parameters

[in]	item	Pointer to the initialized and configured CS libitem
[in]	procedure_count	Number of procedures in the procedure_data array
[in]	procedure_data	Data structure with the completed CS procedures

Calculate distance estimate based on CS procedures.

Returns

- SL_RTL_ERROR_SUCCESS if successful and SL_RTL_ERROR_ESTIMATION_IN_PROGRESS if estimate is not yet final and more input data needs to be processed.

Calculate the distance estimate from the given procedure data. The function can be called with multiple procedures. Use sl_rtl_cs_get_distance to fetch the latest distance estimate. CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called after estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_NOT_CREATED.

Definition at line 1085 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_get_distance_estimate

enum sl_rtl_error_code sl_rtl_cs_get_distance_estimate (sl_rtl_cs_libitem *item, const sl_rtl_cs_distance_estimate_type estimate_type, float *distance)

Parameters

[in]	item	Pointer to the initialized CS libitem
[in]	estimate_type	Type of distance estimate
[out]	distance	Distance out in meters.

Get distance estimate

Returns

- SL_RTL_ERROR_SUCCESS if successful

CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called after estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_NOT_CREATED.

Definition at line 1104 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_get_distance_estimate_confidence

enum sl_rtl_error_code sl_rtl_cs_get_distance_estimate_confidence (sl_rtl_cs_libitem *item, const sl_rtl_cs_distance_estimate_confidence_type confidence_type, float *confidence)

Parameters

[in]	item	Pointer to the initialized CS libitem
[in]	confidence_type	Type of confidence estimate
[out]	confidence	Confidence between 0.0 - 1.0 (unlikely - likely) of the estimated distance

Get distance estimate confidence.

Returns

- SL_RTL_ERROR_SUCCESS if successful

CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called after estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_NOT_CREATED.

Definition at line 1124 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_get_distance_estimate_extended_info

enum sl_rtl_error_code sl_rtl_cs_get_distance_estimate_extended_info (sl_rtl_cs_libitem *item, const sl_rtl_cs_distance_estimate_extended_info_type extended_info_type, float *extended_info)

Parameters

[in]	item	Pointer to the initialized CS libitem
[in]	extended_info_type	Type of distance estimate extended information
[out]	extended_info	Extended info value depending on the extedended_info_type selection

Get distance estimate extended information.

Returns

- SL_RTL_ERROR_SUCCESS if successful

CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED. This method should be called after estimator is created for CS libitem, or it will fail with error code SL_RTL_ERROR_ESTIMATOR_NOT_CREATED.

Definition at line 1145 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_log_enable

```
enum sl_rtl_error_code sl_rtl_cs_log_enable (sl_rtl_cs_libitem *item, const sl_rtl_cs_log_params *log_params)
```

Parameters

[in]	item	Pointer to the CS libitem
[in]	log_params	Pointer to the log initialisation parameters

Enable logging for the given CS libitem.

Returns

- SL_RTL_ERROR_SUCCESS if successful

CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED.

Definition at line 1161 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_log_disable

```
enum sl_rtl_error_code sl_rtl_cs_log_disable (sl_rtl_cs_libitem *item)
```

Parameters

[in]	item	Pointer to the CS libitem

Disable logging for the given CS libitem.

Returns

- SL_RTL_ERROR_SUCCESS if successful

CS libitem must be initialized before calling this method, or it will fail with return code SL_RTL_ERROR_NOT_INITIALIZED.

Definition at line 1173 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_cs_log_get_instance_id

```
uint8_t sl_rtl_cs_log_get_instance_id (sl_rtl_cs_libitem *item)
```

Parameters

[in]	item	Pointer to the CS libitem

Get the instance ID of the given CS libitem.

Returns

- Instance ID of the given CS libitem. The value 0 is returned, if the libitem is not initialized.

Definition at line 1182 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Macro Definition Documentation

SL_RTL_CS_PCT_IQ_BIT_LEN

```
#define SL_RTL_CS_PCT_IQ_BIT_LEN
```

Value:

```
12
```

Number of bits used for indicating I or Q sample in the phase correction term (PCT) in CS step data.

Definition at line 742 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_CS_STEP_MODE_BIT_LEN

```
#define SL_RTL_CS_STEP_MODE_BIT_LEN
```

Value:

```
8
```

Number of bits used for indicating step mode in CS step data.

Definition at line 745 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_CS_TONE_ARRAY_LEN

```
#define SL_RTL_CS_TONE_ARRAY_LEN
```

Value:

```
2
```

Array length of tones in CS step data. This is calculated as number of antenna paths + 1. Note: Currently, RTL library supports only single antenna path.

Definition at line 747 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Error Codes

You are viewing documentation for version: 8.2.0 | Version History

Error Codes

RTL library Error Codes.

Enumerations

enum sl_rtl_error_code {

SL_RTL_ERROR_SUCCESS = 0
SL_RTL_ERROR_ARGUMENT
SL_RTL_ERROR_OUT_OF_MEMORY
SL_RTL_ERROR_ESTIMATION_IN_PROGRESS
SL_RTL_ERROR_NUMBER_OF_SNAPHOTS_DO_NOT_MATCH
SL_RTL_ERROR_ESTIMATOR_NOT_CREATED
SL_RTL_ERROR_ESTIMATOR_ALREADY_CREATED
SL_RTL_ERROR_NOT_INITIALIZED
SL_RTL_ERROR_INTERNAL
SL_RTL_ERROR_IQ_SAMPLE_QA
SL_RTL_ERROR_FEATURE_NOT_SUPPORTED
SL_RTL_ERROR_INCORRECT_MEASUREMENT
SL_RTL_ERROR_LAST

}

RTL error code.

Enumeration Documentation

sl_rtl_error_code

> sl_rtl_error_code

RTL error code.

Enumerator	
SL_RTL_ERROR_SUCCESS	Successful execution / estimation complete.
SL_RTL_ERROR_ARGUMENT	Invalid argument.
SL_RTL_ERROR_OUT_OF_MEMORY	Memory / allocation failure.
SL_RTL_ERROR_ESTIMATION_IN_PROGRESS	Estimation not yet finished.
SL_RTL_ERROR_NUMBER_OF_SNAPHOTS_DO_NOT_MATCH	Initialized and calculated number of snapshots do not match.
SL_RTL_ERROR_ESTIMATOR_NOT_CREATED	Estimator not yet created.
SL_RTL_ERROR_ESTIMATOR_ALREADY_CREATED	Estimator already created, operation is not supported.
SL_RTL_ERROR_NOT_INITIALIZED	Library item not yet initialized.
SL_RTL_ERROR_INTERNAL	An internal error occurred.
SL_RTL_ERROR_IQ_SAMPLE_QA	IQ sample quality analysis failed.
SL_RTL_ERROR_FEATURE_NOT_SUPPORTED	The requested feature is not supported by the library.
SL_RTL_ERROR_INCORRECT_MEASUREMENT	The error of the last measurement for this locator was too large.
SL_RTL_ERROR_LAST	Number of error codes.

Definition at line 60 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Location Finding

You are viewing documentation for version: 8.2.0 | Version History

Location Finding

Location Finding.

These functions are related to the calculation of positions from angles and distances.

Modules

sl_rtl_loc_locator_item

Enumerations

enum sl_rtl_loc_locator_measurement_field {

SL_RTL_LOC_LOCATOR_MEASUREMENT_AZIMUTH
SL_RTL_LOC_LOCATOR_MEASUREMENT_ELEVATION
SL_RTL_LOC_LOCATOR_MEASUREMENT_DISTANCE
SL_RTL_LOC_LOCATOR_MEASUREMENT_AZIMUTH_DEVIATION
SL_RTL_LOC_LOCATOR_MEASUREMENT_ELEVATION_DEVIATION
SL_RTL_LOC_LOCATOR_MEASUREMENT_DISTANCE_DEVIATION
SL_RTL_LOC_LOCATOR_LAST

}

Locator-specific measurements.

enum sl_rtl_loc_target_measurement_field {

SL_RTL_LOC_TARGET_MEASUREMENT_LAST

}

Target-specific measurements - Not yet supported.

enum sl_rtl_loc_estimation_mode {

SL_RTL_LOC_ESTIMATION_MODE_TWO_DIM_FAST_RESPONSE
SL_RTL_LOC_ESTIMATION_MODE_THREE_DIM_FAST_RESPONSE
SL_RTL_LOC_ESTIMATION_MODE_TWO_DIM_HIGH_ACCURACY
SL_RTL_LOC_ESTIMATION_MODE_THREE_DIM_HIGH_ACCURACY
SL_RTL_LOC_ESTIMATION_MODE_LAST

}

Estimation mode.

enum sl_rtl_loc_measurement_validation_method {

SL_RTL_LOC_MEASUREMENT_VALIDATION_MINIMUM
SL_RTL_LOC_MEASUREMENT_VALIDATION_MEDIUM
SL_RTL_LOC_MEASUREMENT_VALIDATION_FULL

}

Measurement validation method.

enum sl_rtl_loc_locator_parameter {

SL_RTL_LOC_LOCATOR_PARAMETER_X_COORDINATE
SL_RTL_LOC_LOCATOR_PARAMETER_Y_COORDINATE
SL_RTL_LOC_LOCATOR_PARAMETER_Z_COORDINATE
SL_RTL_LOC_LOCATOR_PARAMETER_X_ORIENTATION
SL_RTL_LOC_LOCATOR_PARAMETER_Y_ORIENTATION
SL_RTL_LOC_LOCATOR_PARAMETER_Z_ORIENTATION
SL_RTL_LOC_LOCATOR_PARAMETER_LAST

}

Locator-specific parameters related to locationing.

enum sl_rtl_loc_default_accuracy_parameter {

SL_RTL_LOC_DEFAULT_AZIMUTH_STDEV
SL_RTL_LOC_DEFAULT_ELEVATION_STDEV
SL_RTL_LOC_DEFAULT_MINIMUM_DISTANCE_STDEV
SL_RTL_LOC_DEFAULT_DISTANCE_STDEV_COEFF

}

enum sl_rtl_loc_target_parameter {

SL_RTL_LOC_TARGET_PARAMETER_TARGET_HEIGHT
SL_RTL_LOC_TARGET_CLEAR_TARGET_HEIGHT
SL_RTL_LOC_TARGET_PARAMETER_LAST

}

Target-specific parameters related to locationing.

enum sl_rtl_loc_result {

SL_RTL_LOC_RESULT_POSITION_X = 0
SL_RTL_LOC_RESULT_POSITION_Y
SL_RTL_LOC_RESULT_POSITION_Z
SL_RTL_LOC_RESULT_POSITION_RADIUS
SL_RTL_LOC_RESULT_POSITION_STDEV_X
SL_RTL_LOC_RESULT_POSITION_STDEV_Y
SL_RTL_LOC_RESULT_POSITION_STDEV_Z
SL_RTL_LOC_RESULT_POSITION_VELOCITY_X
SL_RTL_LOC_RESULT_POSITION_VELOCITY_Y
SL_RTL_LOC_RESULT_POSITION_VELOCITY_Z
SL_RTL_LOC_RESULT_POSITION_ACCELERATION_X
SL_RTL_LOC_RESULT_POSITION_ACCELERATION_Y
SL_RTL_LOC_RESULT_POSITION_ACCELERATION_Z
SL_RTL_LOC_RESULT_LAST

}

Locationing state results.

Typedefs

typedef void * sl_rtl_loc_libitem

Locationing library item.

Functions

enum sl_rtl_error_code sl_rtl_loc_init(sl_rtl_loc_libitem *item)

enum sl_rtl_error_code sl_rtl_loc_deinit(sl_rtl_loc_libitem *item)

enum sl_rtl_error_code sl_rtl_loc_reinit(sl_rtl_loc_libitem *item)

enum sl_rtl_error_code sl_rtl_loc_set_mode(sl_rtl_loc_libitem *item, enum sl_rtl_loc_estimation_mode mode)

enum sl_rtl_error_code sl_rtl_loc_set_measurement_validation(sl_rtl_loc_libitem *item, enum sl_rtl_loc_measurement_validation_method method)

enum sl_rtl_error_code sl_rtl_loc_add_locator(sl_rtl_loc_libitem *item, struct sl_rtl_loc_locator_item *locator_item, uint32_t *locator_id_out)

enum sl_rtl_error_code sl_rtl_loc_add_tag(sl_rtl_loc_libitem *item, int32_t *tag_id_out)

enum sl_rtl_error_code sl_rtl_loc_remove_tag(sl_rtl_loc_libitem *item, int32_t tag_id)

enum sl_rtl_error_code sl_rtl_loc_create_position_estimator(sl_rtl_loc_libitem *item)

enum sl_rtl_error_code sl_rtl_loc_set_locator_parameter(sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_parameter parameter, float value)

enum sl_rtl_error_code sl_rtl_loc_set_target_parameter(sl_rtl_loc_libitem *item, enum sl_rtl_loc_target_parameter parameter, float value)

enum sl_rtl_error_code sl_rtl_loc_set_target_parameter_tag(sl_rtl_loc_libitem *item, enum sl_rtl_loc_target_parameter parameter, float value, int32_t tag_id)

enum sl_rtl_error_code sl_rtl_loc_clear_measurements(sl_rtl_loc_libitem *item)

enum sl_rtl_error_code sl_rtl_loc_clear_measurements_tag(sl_rtl_loc_libitem *item, int32_t tag_id)

enum sl_rtl_error_code sl_rtl_loc_set_locator_measurement(sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float value)

enum sl_rtl_error_code sl_rtl_loc_set_locator_measurement_tag(sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float value, int32_t tag_id)

enum sl_rtl_error_code sl_rtl_loc_set_target_measurement(sl_rtl_loc_libitem *item, enum sl_rtl_loc_target_measurement_field field, float value)

enum sl_rtl_error_code sl_rtl_loc_set_default_accuracy(sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_default_accuracy_parameter param, float value)

enum sl_rtl_error_code sl_rtl_loc_process(sl_rtl_loc_libitem *item, float time_step)

enum sl_rtl_error_code sl_rtl_loc_process_tag(sl_rtl_loc_libitem *item, float time_step, int32_t tag_id)

enum sl_rtl_error_code sl_rtl_loc_get_result(sl_rtl_loc_libitem *item, enum sl_rtl_loc_result result, float *value)

enum sl_rtl_error_code sl_rtl_loc_get_result_tag(sl_rtl_loc_libitem *item, enum sl_rtl_loc_result result, float *value, int32_t tag_id)

enum sl_rtl_error_code sl_rtl_loc_get_measurement_in_system_coordinates(sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float *value_out)

enum sl_rtl_error_code sl_rtl_loc_get_measurement_in_system_coordinates_tag(sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float *value_out, int32_t tag_id)

enum sl_rtl_error_code	sl_rtl_loc_get_expected_direction(sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance)
enum sl_rtl_error_code	sl_rtl_loc_get_expected_direction_tag(sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance, int32_t tag_id)
enum sl_rtl_error_code	sl_rtl_loc_get_expected_deviation(sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance)
enum sl_rtl_error_code	sl_rtl_loc_get_expected_deviation_tag(sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance, int32_t tag_id)
int	sl_rtl_loc_get_number_disabled(sl_rtl_loc_libitem *item)
int	sl_rtl_loc_get_number_disabled_tag(sl_rtl_loc_libitem *item, int32_t tag_id)
bool	sl_rtl_loc_filter_in_reach(sl_rtl_loc_libitem *item, uint32_t locator_id)
enum sl_rtl_error_code	sl_rtl_loc_filter_clear(sl_rtl_loc_libitem *item, uint32_t locator_id)
enum sl_rtl_error_code	sl_rtl_loc_filter_sphere(sl_rtl_loc_libitem *item, uint32_t locator_id, float radius, bool exclude_region)
enum sl_rtl_error_code	sl_rtl_loc_filter_circle(sl_rtl_loc_libitem *item, uint32_t locator_id, float radius, bool exclude_region)
enum sl_rtl_error_code	sl_rtl_loc_filter_box(sl_rtl_loc_libitem *item, uint32_t locator_id, float xDelta, float yDelta, float zDelta, bool exclude_region)
enum sl_rtl_error_code	sl_rtl_loc_filter_rect(sl_rtl_loc_libitem *item, uint32_t locator_id, float xDelta, float yDelta, bool exclude_region)
enum sl_rtl_error_code	sl_rtl_loc_filter_room(sl_rtl_loc_libitem *item, uint32_t locator_id, float minX, float maxX, float minY, float maxY, float minZ, float maxZ, bool exclude_region)
enum sl_rtl_error_code	sl_rtl_loc_filter_floor(sl_rtl_loc_libitem *item, uint32_t locator_id, float minX, float maxX, float minY, float maxY, bool exclude_region)
enum sl_rtl_error_code	sl_rtl_loc_enable_trilateration(sl_rtl_loc_libitem *item, bool value)
enum sl_rtl_error_code	sl_rtl_loc_trilateration_prefer_below_plane(sl_rtl_loc_libitem *item, bool value)

Macros

#define	SL_RTL_LOC_ALL_LOCATORS (uint32_t)(-1)
#define	SL_RTL_LOC_ALL_TAGS INT32_MAX

Enumeration Documentation

sl_rtl_loc_locator_measurement_field

```
sl_rtl_loc_locator_measurement_field
```

Locator-specific measurements.

Enumerator

SL_RTL_LOC_LOCATOR_MEASUREMENT_AZIMUTH	Measured azimuth from locator to tag.
SL_RTL_LOC_LOCATOR_MEASUREMENT_ELEVATION	Measured elevation from locator to tag.
SL_RTL_LOC_LOCATOR_MEASUREMENT_DISTANCE	Measured distance from locator to tag.
SL_RTL_LOC_LOCATOR_MEASUREMENT_AZIMUTH_DEVIATION	Azimuth stdev.
SL_RTL_LOC_LOCATOR_MEASUREMENT_ELEVATION_DEVIATION	Elevation stdev.
SL_RTL_LOC_LOCATOR_MEASUREMENT_DISTANCE_DEVIATION	Distance stdev.
SL_RTL_LOC_LOCATOR_LAST	

Definition at line 1199 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_target_measurement_field

> sl_rtl_loc_target_measurement_field

Target-specific measurements - Not yet supported.

Enumerator	
SL_RTL_LOC_TARGET_MEASUREMENT_LAST	

Definition at line 1212 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_estimation_mode

> sl_rtl_loc_estimation_mode

Estimation mode.

Enumerator	
SL_RTL_LOC_ESTIMATION_MODE_TWO_DIM_FAST_RESPONSE	Two-dimensional mode - Only X and Y plane considered, less filtered mode for fast moving assests.
SL_RTL_LOC_ESTIMATION_MODE_THREE_DIM_FAST_RESPONSE	Three-dimensional mode - Covers X, Y and Z planes, less filtered mode for fast moving assests.
SL_RTL_LOC_ESTIMATION_MODE_TWO_DIM_HIGH_ACCURACY	Two-dimensional mode - Only X and Y plane considered, more filtered mode for relatively static assets.
SL_RTL_LOC_ESTIMATION_MODE_THREE_DIM_HIGH_ACCURACY	Three-dimensional mode - Covers X, Y and Z planes, more filtered mode for relatively static assets.
SL_RTL_LOC_ESTIMATION_MODE_LAST	

Definition at line 1217 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_measurement_validation_method

> sl_rtl_loc_measurement_validation_method

Measurement validation method.

Enumerator	
SL_RTL_LOC_MEASUREMENT_VALIDATION_MINIMUM	Only the basic validation integrated in the locationing algorithm (default)

SL_RTL_LOC_MEASUREMENT_VALIDATION_MEDIUM	May discard the most inaccurate measurements with an additional calculation round.
SL_RTL_LOC_MEASUREMENT_VALIDATION_FULL	May discard the most inaccurate measurements with several calculation rounds.

Definition at line 1228 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_locator_parameter

> sl_rtl_loc_locator_parameter

Locator-specific parameters related to locationing.

Enumerator	
SL_RTL_LOC_LOCATOR_PARAMETER_X_COORDINATE	Set X-axis coordinate of the locator.
SL_RTL_LOC_LOCATOR_PARAMETER_Y_COORDINATE	Set Y-axis coordinate of the locator.
SL_RTL_LOC_LOCATOR_PARAMETER_Z_COORDINATE	Set Z-axis coordinate of the locator.
SL_RTL_LOC_LOCATOR_PARAMETER_X_ORIENTATION	Set X-axis rotation of the locator (Euler angles) in degrees.
SL_RTL_LOC_LOCATOR_PARAMETER_Y_ORIENTATION	Set Y-axis rotation of the locator (Euler angles) in degrees.
SL_RTL_LOC_LOCATOR_PARAMETER_Z_ORIENTATION	Set Z-axis rotation of the locator (Euler angles) in degrees.
SL_RTL_LOC_LOCATOR_PARAMETER_LAST	

Definition at line 1235 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_default_accuracy_parameter

> sl_rtl_loc_default_accuracy_parameter

Enumerator	
SL_RTL_LOC_DEFAULT_AZIMUTH_STDEV	Set the default azimuth stdev.
SL_RTL_LOC_DEFAULT_ELEVATION_STDEV	Set the default elevation stdev.
SL_RTL_LOC_DEFAULT_MINIMUM_DISTANCE_STDEV	Set the default minimum distance stdev, will be added to the effective stdev value.
SL_RTL_LOC_DEFAULT_DISTANCE_STDEV_COEFF	Set the default distance stdev coeff, will be added after multiplied with the distance measurement to the effective stdev value.

Definition at line 1246 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_target_parameter

> sl_rtl_loc_target_parameter

Target-specific parameters related to locationing.

Enumerator	
SL_RTL_LOC_TARGET_PARAMETER_TARGET_HEIGHT	
SL_RTL_LOC_TARGET_CLEAR_TARGET_HEIGHT	
SL_RTL_LOC_TARGET_PARAMETER_LAST	

Definition at line 1254 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_result

```
sl_rtl_loc_result
```

Locationing state results.

Enumerator	
SL_RTL_LOC_RESULT_POSITION_X	Estimated X-axis position of the target.
SL_RTL_LOC_RESULT_POSITION_Y	Estimated Y-axis position of the target.
SL_RTL_LOC_RESULT_POSITION_Z	Estimated Z-axis position of the target.
SL_RTL_LOC_RESULT_POSITION_RADIUS	The combined radius (stdev) of the location estimate.
SL_RTL_LOC_RESULT_POSITION_STDEV_X	The accuracy (stdev) of the location x-coordinate estimate.
SL_RTL_LOC_RESULT_POSITION_STDEV_Y	The accuracy (stdev) of the location y-coordinate estimate.
SL_RTL_LOC_RESULT_POSITION_STDEV_Z	The accuracy (stdev) of the location z-coordinate estimate.
SL_RTL_LOC_RESULT_POSITION_VELOCITY_X	Estimated X-axis velocity of the target.
SL_RTL_LOC_RESULT_POSITION_VELOCITY_Y	Estimated Y-axis velocity of the target.
SL_RTL_LOC_RESULT_POSITION_VELOCITY_Z	Estimated Z-axis velocity of the target.
SL_RTL_LOC_RESULT_POSITION_ACCELERATION_X	Estimated X-axis acceleration of the target.
SL_RTL_LOC_RESULT_POSITION_ACCELERATION_Y	Estimated Y-axis acceleration of the target.
SL_RTL_LOC_RESULT_POSITION_ACCELERATION_Z	Estimated Z-axis acceleration of the target.
SL_RTL_LOC_RESULT_LAST	Number of results.

Definition at line 1262 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Typedef Documentation

sl_rtl_loc_libitem

```
typedef void* sl_rtl_loc_libitem
```

Locationing library item.

Definition at line 1293 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Function Documentation

sl_rtl_loc_init

```
enum sl_rtl_error_code sl_rtl_loc_init (sl_rtl_loc_libitem *item)
```

Parameters

[in]	item	Pointer to the libitem to be initialized

Initialize the locationing libitem instance.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1303 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_deinit

> enum sl_rtl_error_code sl_rtl_loc_deinit (sl_rtl_loc_libitem *item)

Parameters

[in]	item	Pointer to the libitem to be initialized

Deinitialize the locationing libitem instance.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1311 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_reinit

> enum sl_rtl_error_code sl_rtl_loc_reinit (sl_rtl_loc_libitem *item)

Parameters

[in]	item	Pointer to the libitem to be initialized

Reinitialize the locationing libitem instance.

Reset the libitem's internal values to the starting point so that it can start all over from the beginning. This can be used, for example, in testing instead of deleting and re-creating the libitem object.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1323 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_mode

> enum sl_rtl_error_code sl_rtl_loc_set_mode (sl_rtl_loc_libitem *item, enum sl_rtl_loc_estimation_mode mode)

Parameters

[in]	item	Pointer to the libitem
[in]	mode	Estimation mode as enum

Set the locationing dimensionality mode. Possible choices are 2D or 3D modes. Two-dimensional mode does not vary the z-position of the target and assumes it is 0 at all times. When updating, for example, the distance measure- ment from the locator to the target in 2D mode, it should be noted that even if the locator and tag are on different z-planes, the distance should be given as the distance as if they were on the same z-plane.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1337 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_measurement_validation

> enum sl_rtl_error_code sl_rtl_loc_set_measurement_validation (sl_rtl_loc_libitem *item, enum sl_rtl_loc_measurement_validation_method method)

Parameters

[in]	item	Pointer to the libitem
[in]	method	Validation method as enum

Set the measurement validation method. If not given, the minimum method is used. Possible choices are minimum, medium, and full. The minimum does not increase the calculation CPU load because it is integrated inside the algorithm's normal operation. The medium and full methods use additionally one or multiple calculation rounds respectively and may discard the most inaccurate measurements from the calculation. However, they will always leave enough of them to calculate the estimate.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1352 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_add_locator

> enum sl_rtl_error_code sl_rtl_loc_add_locator (sl_rtl_loc_libitem *item, struct sl_rtl_loc_locator_item *locator_item, uint32_t *locator_id_out)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_item	Locator item instance to be added
[out]	locator_id_out	ID of the locator assigned by the library

Add a locator item into the locationing estimator after setting its position and orientation parameters. Add all locators using this function before calling sl_rtl_loc_create_position_estimator to create the estimator instance.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1365 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_add_tag

> enum sl_rtl_error_code sl_rtl_loc_add_tag (sl_rtl_loc_libitem *item, int32_t *tag_id_out)

Parameters

[in]	item	Pointer to the libitem
[out]	tag_id_out	ID of the tag assigned by the library

Add an asset tag item into the locationing estimator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1374 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_remove_tag

> enum sl_rtl_error_code sl_rtl_loc_remove_tag (sl_rtl_loc_libitem *item, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[out]	tag_id	of the tag to be removed

Remove an asset tag item previously added with sl_rtl_loc_add_tag.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1383 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_create_position_estimator

> enum sl_rtl_error_code sl_rtl_loc_create_position_estimator (sl_rtl_loc_libitem *item)

Parameters

[in]	item	Pointer to the libitem

Create the position estimator instance after all locators have been added.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1391 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_locator_parameter

> enum sl_rtl_error_code sl_rtl_loc_set_locator_parameter (sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_parameter parameter, float value)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator for which the parameter will be set
[in]	parameter	Parameter to be set (as enum)
[in]	value	New value for the parameter

Set locator-specific parameters, such as the azimuth and elevation measurement covariances.

Returns

SL_RTL_ERROR_SUCCESS if successful

Definition at line 1405 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_target_parameter

> enum sl_rtl_error_code sl_rtl_loc_set_target_parameter (sl_rtl_loc_libitem *item, enum sl_rtl_loc_target_parameter parameter, float value)

Parameters

[in]	item	Pointer to the libitem
[in]	parameter	Parameter to be set (as enum)
[in]	value	New value for the parameter

Set target-specific parameters, such as the known target height.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1416 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_target_parameter_tag

> enum sl_rtl_error_code sl_rtl_loc_set_target_parameter_tag (sl_rtl_loc_libitem *item, enum sl_rtl_loc_target_parameter parameter, float value, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	parameter	Parameter to be set (as enum)
[in]	value	New value for the parameter
[in]	tag_id	TagID of the asset to be changed or SL_RTL_LOC_ALL_TAGS

Set target-specific parameters, such as the known target height for a specified tag.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1427 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_clear_measurements

> enum sl_rtl_error_code sl_rtl_loc_clear_measurements (sl_rtl_loc_libitem *item)

Parameters

[in]	item	Pointer to the libitem

Clear previous measurements. Call this function before setting new measurements if the previous measurements are not valid for the next iteration step.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1437 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_clear_measurements_tag

enum sl_rtl_error_code sl_rtl_loc_clear_measurements_tag (sl_rtl_loc_libitem *item, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	tag_id	TagID of the asset to be changed or SL_RTL_LOC_ALL_TAGS

Clear previous measurements for a specified tag. Call this function before setting new measurements if the previous measurements are not valid for the next iteration step.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1448 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_locator_measurement

enum sl_rtl_error_code sl_rtl_loc_set_locator_measurement (sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float value)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator for which the measurement will be set
[in]	field	Measurement to be updated (as enum)
[in]	value	New value for the measurement

Set a new measurement for a locator specified by the locator ID.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1459 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_locator_measurement_tag

enum sl_rtl_error_code sl_rtl_loc_set_locator_measurement_tag (sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float value, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator for which the measurement will be set

[in]	field	Measurement to be updated (as enum)
[in]	value	New value for the measurement
[in]	tag_id	TagID of the asset to be changed

Set a new measurement for a locator specified by the locator ID for a specified tag.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1471 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_target_measurement

> enum sl_rtl_error_code sl_rtl_loc_set_target_measurement (sl_rtl_loc_libitem *item, enum sl_rtl_loc_target_measurement_field field, float value)

Parameters

[in]	item	Pointer to the libitem
[in]	field	Measurement to be updated (as enum)
[in]	value	New value for the measurement

Set a new measurement for the target. Because the current version does not support this function, calling it has no effect.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1482 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_set_default_accuracy

> enum sl_rtl_error_code sl_rtl_loc_set_default_accuracy (sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_default_accuracy_parameter param, float value)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator whose parameter will be set
[in]	param	Parameter to be set (as enum)
[in]	value	New value for the default accuracy

Set the default accuracy value for direction / distance. This value will be used if not given explicitly by measurement.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1494 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_process

> enum sl_rtl_error_code sl_rtl_loc_process (sl_rtl_loc_libitem *item, float time_step)

Parameters

[in]	item	Pointer to the libitem
[in]	time_step	Process time interval in seconds. This is the time separation between the previous step and this current step.

Process the current time step of the locationing filter. Call this function after old measurements are cleared and new measurements are set for the current time step. The process will step the filter forward and provide new state results and state covariance values, which can be retrieved with separate function calls.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1508 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_process_tag

> enum sl_rtl_error_code sl_rtl_loc_process_tag (sl_rtl_loc_libitem *item, float time_step, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	time_step	Process time interval in seconds. This is the time separation between the previous step and this current step.
[in]	tag_id	TagID of the asset to be processed or SL_RTL_LOC_ALL_TAGS

Process the current time step of the locationing filter for a specified tag. Call this function after old measurements are cleared and new measurements are set for the current time step. The process will step the filter forward and provide new state results and state covariance values, which can be retrieved with separate function calls.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1523 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_result

> enum sl_rtl_error_code sl_rtl_loc_get_result (sl_rtl_loc_libitem *item, enum sl_rtl_loc_result result, float *value)

Parameters

[in]	item	Pointer to the libitem
[in]	result	The desired state variable result as enum
[out]	value	The value output of the state variable

Get result of the current state variables or state covariances.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1533 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_result_tag

enum sl_rtl_error_code sl_rtl_loc_get_result_tag (sl_rtl_loc_libitem *item, enum sl_rtl_loc_result result, float *value, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	result	The desired state variable result as enum
[out]	value	The value output of the state variable
[in]	tag_id	TagID of the asset to fetch the results for

Get result of the current state variables or state covariances for a specified tag.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1544 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_measurement_in_system_coordinates

enum sl_rtl_error_code sl_rtl_loc_get_measurement_in_system_coordinates (sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float *value_out)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator whose measurement will be retrieved
[in]	field	Measurement to be retrieved (as enum)
[out]	value_out	The value output of the state variable

Get the latest measurement converted into the coordinate system of the multi-locator system.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1556 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_measurement_in_system_coordinates_tag

enum sl_rtl_error_code sl_rtl_loc_get_measurement_in_system_coordinates_tag (sl_rtl_loc_libitem *item, uint32_t locator_id, enum sl_rtl_loc_locator_measurement_field field, float *value_out, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator whose measurement will be retrieved
[in]	field	Measurement to be retrieved (as enum)
[out]	value_out	The value output of the state variable
[in]	tag_id	TagID of the asset

Get the latest measurement converted into the coordinate system of the multi-locator system for a specified tag.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1569 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_expected_direction

enum sl_rtl_error_code sl_rtl_loc_get_expected_direction (sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator whose measurement will be retrieved
[out]	azimuth	The expected azimuth value
[out]	elevation	The expected elevation value
[out]	distance	The expected distance value

Get the expected direction of the asset for the locator.

Returns

- SL_RTL_ERROR_SUCCESS if the locator does not require correction
- SL_RTL_ERROR_INCORRECT_MEASUREMENT if the locator's previous measurement was ignored because the error is too large
- Another error code in case of error

Position calculation has a better overall view to the asset's location than any individual locator. If the direction angles differ too much from the expected direction, the locator can be instructed to correct its internal state so that in can recover from this incorrectness faster. See also sl_rtl_aox_set_expected_direction()

Definition at line 1589 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_expected_direction_tag

enum sl_rtl_error_code sl_rtl_loc_get_expected_direction_tag (sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator whose measurement will be retrieved
[out]	azimuth	The expected azimuth value
[out]	elevation	The expected elevation value
[out]	distance	The expected distance value
[in]	tag_id	TagID of the asset

Get the expected direction of the asset for the locator for a specified tag.

Returns

- SL_RTL_ERROR_SUCCESS if the locator does not require correction
- SL_RTL_ERROR_INCORRECT_MEASUREMENT if the locator's previous measurement was ignored because of too large error

Another error code in case of error

Position calculation has a better overall view to the asset's location than any individual locator. If the direction angles differ too much from the expected direction, the locator can be instructed to correct its internal state so that in can recover from this incorrectness faster. See also sl_rtl_aox_set_expected_direction().

Definition at line 1610 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_expected_deviation

enum sl_rtl_error_code sl_rtl_loc_get_expected_deviation (sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance)

Parameters

[in]	item	ID of the locator whose measurement will be retrieved
[out]	locator_id	The expected azimuth value's standard deviation
[out]	azimuth	The expected elevation value's standard deviation
[in]	elevation	Pointer to the libitem
[out]	distance	The expected distance value's standard deviation

Get the deviation values for expected direction of the asset for the locator.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1622 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_expected_deviation_tag

enum sl_rtl_error_code sl_rtl_loc_get_expected_deviation_tag (sl_rtl_loc_libitem *item, uint32_t locator_id, float *azimuth, float *elevation, float *distance, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator whose measurement will be retrieved
[out]	azimuth	The expected azimuth value's standard deviation
[out]	elevation	The expected elevation value's standard deviation
[out]	distance	The expected distance value's standard deviation
[in]	tag_id	TagID of the asset

Get the deviation values for expected direction of the asset for the locator for a specified tag.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1636 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_number_disabled

> int sl_rtl_loc_get_number_disabled (sl_rtl_loc_libitem *item)

Parameters

[in]	item	Pointer to the libitem

Get the number of locators, which are disabled from position calculation because the direction error is too large.

Returns

- Number of locators needing correction, or -1 in case of any other error

Definition at line 1645 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_get_number_disabled_tag

> int sl_rtl_loc_get_number_disabled_tag (sl_rtl_loc_libitem *item, int32_t tag_id)

Parameters

[in]	item	Pointer to the libitem
[in]	tag_id	TagID of the asset

Get the number of locators, which are disabled from position calculation because the direction error is too large, for a specified tag.

Returns

- Number of locators needing correction, or -1 in case of any other error

Definition at line 1655 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_in_reach

> bool sl_rtl_loc_filter_in_reach (sl_rtl_loc_libitem *item, uint32_t locator_id)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of the locator to run the test

Check if the asset is in reach of a given locator according to the position-based filtering.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1665 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_clear

> enum sl_rtl_error_code sl_rtl_loc_filter_clear (sl_rtl_loc_libitem *item, uint32_t locator_id)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of a single locator or SL_RTL_LOC_ALL_LOCATORS

Clear all the position-based filters from a locator or from all locators.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1675 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_sphere

> enum sl_rtl_error_code sl_rtl_loc_filter_sphere (sl_rtl_loc_libitem *item, uint32_t locator_id, float radius, bool exclude_region)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of a single locator or SL_RTL_LOC_ALL_LOCATORS
[in]	radius	Distance from the locator
[in]	exclude_region	If true, define an excluded region instead

Add a position-based filter. This filter is based on the distance from a locator. The filter can be added to one locator or all locators. A locator may have several filters which are combined.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1688 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_circle

> enum sl_rtl_error_code sl_rtl_loc_filter_circle (sl_rtl_loc_libitem *item, uint32_t locator_id, float radius, bool exclude_region)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of a single locator or SL_RTL_LOC_ALL_LOCATORS
[in]	radius	Distance from the locator
[in]	exclude_region	If true, define an excluded region instead

Add a position-based filter. The two dimensional filter is based on the distance from a locator. The filter can be added to one locator or all locators. A locator may have several filters which are combined.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1702 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_box

enum sl_rtl_error_code sl_rtl_loc_filter_box (sl_rtl_loc_libitem *item, uint32_t locator_id, float xDelta, float yDelta, float zDelta, bool exclude_region)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of a single locator or SL_RTL_LOC_ALL_LOCATORS
[in]	xDelta	Maximum distance in X-coordinate direction
[in]	yDelta	Maximum distance in Y-coordinate direction
[in]	zDelta	Maximum distance in Z-coordinate direction
[in]	exclude_region	If true, define an excluded region instead

Add a position-based filter. This filter defines relative distances in coordinate axes' directions. The filter can be added to one locator or all locators. A locator may have several filters which are combined.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1718 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_rect

enum sl_rtl_error_code sl_rtl_loc_filter_rect (sl_rtl_loc_libitem *item, uint32_t locator_id, float xDelta, float yDelta, bool exclude_region)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of a single locator or SL_RTL_LOC_ALL_LOCATORS
[in]	xDelta	Maximum distance in X-coordinate direction
[in]	yDelta	Maximum distance in Y-coordinate direction
[in]	exclude_region	If true, define an excluded region instead

Add a position-based filter. This two dimensional filter defines relative distances in coordinate axes' directions. The filter can be added to one locator or all locators. A locator may have several filters which are combined.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1733 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_room

enum sl_rtl_error_code sl_rtl_loc_filter_room (sl_rtl_loc_libitem *item, uint32_t locator_id, float minX, float maxX, float minY, float maxY, float minZ, float maxZ, bool exclude_region)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of a single locator or SL_RTL_LOC_ALL_LOCATORS

[in]	minX	Minimum X-coordinate value
[in]	maxX	Maximum X-coordinate value
[in]	minY	Minimum Y-coordinate value
[in]	maxY	Maximum Y-coordinate value
[in]	minZ	Minimum Z-coordinate value
[in]	maxZ	Maximum Z-coordinate value
[in]	exclude_region	If true, define an excluded region instead

Add a position-based filter. This filter defines an area in global coordinates. The filter can be added to one locator or all locators. A locator may have several filters which are combined.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1752 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_filter_floor

> enum sl_rtl_error_code sl_rtl_loc_filter_floor (sl_rtl_loc_libitem *item, uint32_t locator_id, float minX, float maxX, float minY, float maxY, bool exclude_region)

Parameters

[in]	item	Pointer to the libitem
[in]	locator_id	ID of a single locator or SL_RTL_LOC_ALL_LOCATORS
[in]	minX	Minimum X-coordinate value
[in]	maxX	Maximum X-coordinate value
[in]	minY	Minimum Y-coordinate value
[in]	maxY	Maximum Y-coordinate value
[in]	exclude_region	If true, define an excluded region instead

Add a position-based filter. The two dimensional filter defines an area in global coordinates. The filter can be added to one locator or all locators. A locator may have several filters which are combined.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1769 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_enable_trilateration

> enum sl_rtl_error_code sl_rtl_loc_enable_trilateration (sl_rtl_loc_libitem *item, bool value)

Parameters

[in]	item	Pointer to the libitem
[in]	value	Turn the trilateration on or off

Enable trilateration method in location calculation.

The trilateration method calculates the position based on the distances from the locators rather than the direction angles. This function enables the method, but the location calculation may still utilize also the directon if available to further increase the accuracy.

Trilateration method is not turned on by default.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1785 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_trilateration_prefer_below_plane

```
enum sl_rtl_error_code sl_rtl_loc_trilateration_prefer_below_plane (sl_rtl_loc_libitem *item, bool value)
```

Parameters

[in]	item	Pointer to the libitem
[in]	value	Turn the trilateration on or off

Prefer the asset location below or above the locator plane.

If the locators are all in the same plane (or close to it), the height of the asset may become ambiguous. Set the preference to be below or above plane so that is more likely to end up in the correct position.

The default options is below the plane.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1800 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Macro Definition Documentation

SL_RTL_LOC_ALL_LOCATORS

```
#define SL_RTL_LOC_ALL_LOCATORS
```

Value:

```
(uint32_t)(-1)
```

Definition at line 1195 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_LOC_ALL_TAGS

```
#define SL_RTL_LOC_ALL_TAGS
```

Value:

```
INT32_MAX
```

Definition at line 1196 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_loc_locator_item

You are viewing documentation for version: 8.2.0 | Version History

Locator item which contains the position and orientation of the locator array.

Public Attributes

float	coordinate_x X-axis coordinate of the locator.
float	coordinate_y Y-axis coordinate of the locator.
float	coordinate_z Z-axis coordinate of the locator.
float	orientation_x_axis_degrees X-axis rotation of the locator (Euler angles) in degrees.
float	orientation_y_axis_degrees Y-axis rotation of the locator (Euler angles) in degrees.
float	orientation_z_axis_degrees Z-axis rotation of the locator (Euler angles) in degrees.

Public Attribute Documentation

coordinate_x

```
float sl_rtl_loc_locator_item::coordinate_x
```

X-axis coordinate of the locator.

Definition at line 1284 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

coordinate_y

```
float sl_rtl_loc_locator_item::coordinate_y
```

Y-axis coordinate of the locator.

Definition at line 1285 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

coordinate_z

```
float sl_rtl_loc_locator_item::coordinate_z
```

Z-axis coordinate of the locator.

Definition at line 1286 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

orientation_x_axis_degrees

> float sl_rtl_loc_locator_item::orientation_x_axis_degrees

X-axis rotation of the locator (Euler angles) in degrees.

Definition at line 1287 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

orientation_y_axis_degrees

> float sl_rtl_loc_locator_item::orientation_y_axis_degrees

Y-axis rotation of the locator (Euler angles) in degrees.

Definition at line 1288 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

orientation_z_axis_degrees

> float sl_rtl_loc_locator_item::orientation_z_axis_degrees

Z-axis rotation of the locator (Euler angles) in degrees.

Definition at line 1289 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Logging

Logging

Logging.

RTL events are logged as follows:

1. Initialize the logging.
2. Configure the logging with the configuration parameters.
   - Configuration is only allowed once in a logging session.
3. Enable logging for the RTL libitem(s).
   - The libitem(s) can be added before or after the estimator(s) are created.
   - Disable logging for the RTL libitem(s) to stop the logging for the given RTL libitem.
4. Deinitialize logging. This will disable logging for all the remaining RTL libitems.

Typedefs

typedef void(*	sl_rtl_log_callback_function)(uint8_t *log_data, size_t log_data_len)

Functions

	typedef(struct { sl_rtl_log_callback_function log_callback_function;char sdk_version[SL_RTL_LOG_SDK_VERSION_CHAR_ARRAY_MAX_SIZE];char command_line_options[SL_RTL_LOG_COMMAND_LINE_OPTIONS_CHAR_ARRAY_MAX_SIZE];}) sl_rtl_log_params
enum sl_rtl_error_code	sl_rtl_log_init(void)
enum sl_rtl_error_code	sl_rtl_log_configure(const sl_rtl_log_params *log_params)
enum sl_rtl_error_code	sl_rtl_log_deinit(void)

Macros

#define	SL_RTL_LOG_SDK_VERSION_CHAR_ARRAY_MAX_SIZE 64
#define	SL_RTL_LOG_COMMAND_LINE_OPTIONS_CHAR_ARRAY_MAX_SIZE 64

Typedef Documentation

sl_rtl_log_callback_function

```
typedef void(* sl_rtl_log_callback_function) (uint8_t *log_data, size_t log_data_len) )(uint8_t *log_data, size_t log_data_len)
```

Definition at line 2033 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Function Documentation

typedef

```
typedef (struct { sl_rtl_log_callback_function log_callback_function;char
sdk_version[SL_RTL_LOG_SDK_VERSION_CHAR_ARRAY_MAX_SIZE];char
command_line_options[SL_RTL_LOG_COMMAND_LINE_OPTIONS_CHAR_ARRAY_MAX_SIZE];}) sl_rtl_log_params
```

Parameters

N/A		

Definition at line `2036` of file `/mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h`

sl_rtl_log_init

```
enum sl_rtl_error_code sl_rtl_log_init (void)
```

Parameters

N/A		

Initiate the RTL logging.

Definition at line `2054` of file `/mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h`

sl_rtl_log_configure

```
enum sl_rtl_error_code sl_rtl_log_configure (const sl_rtl_log_params *log_params)
```

Parameters

[in]	log_params	Pointer to the sl_rtl_log_params

Register the configuration parameters for the logging.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line `2063` of file `/mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h`

sl_rtl_log_deinit

```
enum sl_rtl_error_code sl_rtl_log_deinit (void)
```

Parameters

N/A		

Deinitiate the RTL logging.

Definition at line `2068` of file `/mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h`

Macro Definition Documentation

SL_RTL_LOG_SDK_VERSION_CHAR_ARRAY_MAX_SIZE

```
#define SL_RTL_LOG_SDK_VERSION_CHAR_ARRAY_MAX_SIZE
```

Value:

```
64
```

Definition at line 2030 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

SL_RTL_LOG_COMMAND_LINE_OPTIONS_CHAR_ARRAY_MAX_SIZE

```
#define SL_RTL_LOG_COMMAND_LINE_OPTIONS_CHAR_ARRAY_MAX_SIZE
```

Value:

```
64
```

Definition at line 2031 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Utility Functionality

You are viewing documentation for version: 8.2.0 | Version History

Utility Functionality

Utility Functionality.

These functions provide supporting functionality such as conversions, additional filtering, diagnostics, and setup functions.

Enumerations

enum sl_rtl_util_parameter {
  SL_RTL_UTIL_PARAMETER_AMOUNT_OF_FILTERING = 0
}
Util parameter.

Typedefs

typedef void * sl_rtl_util_libitem
Utility library item.

Functions

enum sl_rtl_error_code sl_rtl_util_init(sl_rtl_util_libitem *item)

enum sl_rtl_error_code sl_rtl_util_deinit(sl_rtl_util_libitem *item)

enum sl_rtl_error_code sl_rtl_util_set_parameter(sl_rtl_util_libitem *item, enum sl_rtl_util_parameter parameter, float value)

enum sl_rtl_error_code sl_rtl_util_filter(sl_rtl_util_libitem *item, float value_in, float *value_out)

enum sl_rtl_error_code sl_rtl_util_rssi2distance(float tx_power, float rssi, float *distance_out)

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_setup(sl_rtl_util_libitem *item, uint32_t level, uint32_t num_antennas, bool raw_samples)

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_parameters(sl_rtl_util_libitem *item, uint32_t sample_rate, uint8_t sample_offset, uint8_t samples_in_slot, uint8_t number_of_channels)

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_downsampling_factor(sl_rtl_util_libitem *item, float dsf)

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_data_offset(sl_rtl_util_libitem *item, uint32_t offset)

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_switch_pattern(sl_rtl_util_libitem *item, uint32_t size, uint32_t *pattern)

enum sl_rtl_error_code	sl_rtl_util_iq_sample_qa_add_reference(sl_rtl_util_libitem *item, uint32_t num_samples, float *i_samples, float *q_samples)
enum sl_rtl_error_code	sl_rtl_util_iq_sample_qa_add_data(sl_rtl_util_libitem *item, uint32_t num_snapshots, float **i_samples, float **q_samples, uint8_t channel)
uint32_t	sl_rtl_util_iq_sample_qa_get_results(sl_rtl_util_libitem *item)
enum sl_rtl_error_code	sl_rtl_util_iq_sample_qa_get_details(sl_rtl_util_libitem *item, sl_rtl_clib_iq_sample_qa_dataset_t *iq_sample_qa_results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)
enum sl_rtl_error_code	sl_rtl_util_iq_sample_qa_get_channel_details(sl_rtl_util_libitem *item, uint8_t channel, sl_rtl_clib_iq_sample_qa_dataset_t *iq_sample_qa_results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)
char *	sl_rtl_util_iq_sample_qa_code2string(char *buf, int size, uint32_t code)

Enumeration Documentation

sl_rtl_util_parameter

> sl_rtl_util_parameter

Util parameter.

Enumerator	
SL_RTL_UTIL_PARAMETER_AMOUNT_OF_FILTERING	Amount of filtering in range [0.0f, 1.0f].

Definition at line 1818 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Typedef Documentation

sl_rtl_util_libitem

> typedef void* sl_rtl_util_libitem

Utility library item.

Definition at line 1815 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

Function Documentation

sl_rtl_util_init

> enum sl_rtl_error_code sl_rtl_util_init (sl_rtl_util_libitem *item)

Parameters

[in]	item	Pointer to the libitem to be initialized

Initialize the Util libitem instance.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1828 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_deinit

```
enum sl_rtl_error_code sl_rtl_util_deinit (sl_rtl_util_libitem *item)
```

Parameters

[in]	item	Pointer to the libitem to be deinitialized

Deinitialize a Util libitem instance.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1836 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_set_parameter

```
enum sl_rtl_error_code sl_rtl_util_set_parameter (sl_rtl_util_libitem *item, enum sl_rtl_util_parameter parameter, float value)
```

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	parameter	Parameter to change
[in]	value	Value of the parameter

Set util parameters.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1846 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_filter

```
enum sl_rtl_error_code sl_rtl_util_filter (sl_rtl_util_libitem *item, float value_in, float *value_out)
```

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	value_in	Value to be fed to the filter
[out]	value_out	Pointer to the filtered value

Filter a value using the moving average.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1856 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_rssi2distance

enum sl_rtl_error_code sl_rtl_util_rssi2distance (float tx_power, float rssi, float *distance_out)

Parameters

[in]	tx_power	Reference RSSI value of the TX-device at 1.0 m distance in dBm, for example -45.0f
[in]	rssi	Measured RSSI from the receiver
[out]	distance_out	Distance in meters

Convert an RSSI-value to distance in meters.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1866 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_setup

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_setup (sl_rtl_util_libitem *item, uint32_t level, uint32_t num_antennas, bool raw_samples)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	level	Analysis level
[in]	num_antennas	The number of antennas in the array
[in]	raw_samples	Data contains raw samples instead of picked ones

Set up the IQ sample quality analysis.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1877 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_set_parameters

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_parameters (sl_rtl_util_libitem *item, uint32_t sample_rate, uint8_t sample_offset, uint8_t samples_in_slot, uint8_t number_of_channels)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	sample_rate	The sampling rate
[in]	sample_offset	The offset for the picked sample in the sampling slot
[in]	samples_in_slot	Number of samples in the sampling slot
[in]	number_of_channels	Number of radio channels

Set up parameters for the IQ sample quality analysis.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1889 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_set_downsampling_factor

> enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_downsampling_factor (sl_rtl_util_libitem *item, float dsf)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	dsf	The downsampling factor

Set up IQ sample QA downsampling ratio.

Set the downsampling factor used during the antenna samples (related to the sampling rate of the reference period)

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1901 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_set_data_offset

> enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_data_offset (sl_rtl_util_libitem *item, uint32_t offset)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	offset	The data samples' offset

Set data samples offset.

Set the offset for the data samples from the start of the reference period. The use of this function is optional and, if not given, the default values are used.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1914 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_set_switch_pattern

> enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_set_switch_pattern (sl_rtl_util_libitem *item, uint32_t size, uint32_t *pattern)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	size	The size of the switch pattern
[in]	pattern	Array of integers representing the switch pattern

Set the switch pattern for the IQ sample quality.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1924 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_add_reference

> enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_add_reference (sl_rtl_util_libitem *item, uint32_t num_samples, float *i_samples, float *q_samples)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	num_samples	The size of data
[in]	i_samples	I-part of the sample data
[in]	q_samples	Q-part of the sample data

Feed the IQ sample quality analysis the reference period data.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1935 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_add_data

> enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_add_data (sl_rtl_util_libitem *item, uint32_t num_snapshots, float **i_samples, float **q_samples, uint8_t channel)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	num_snapshots	The size of the data
[in]	i_samples	I-part of the sample data
[in]	q_samples	Q-part of the sample data
[in]	channel	Radio channel the packet is using.

Feed the IQ sample quality analysis the antenna data.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1946 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_get_results

> uint32_t sl_rtl_util_iq_sample_qa_get_results (sl_rtl_util_libitem *item)

Parameters

[in]	item	Pointer to the initialized Util libitem

Get the overall results of the IQ sample quality analysis.

Returns

- Bitmask of the failing tests, zero if everything passed

Definition at line 1955 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_get_details

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_get_details (sl_rtl_util_libitem *item, sl_rtl_clib_iq_sample_qa_dataset_t *iq_sample_qa_results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)

Parameters

[in]	item	Pointer to the initialized Util libitem
[out]	iq_sample_qa_results	The data structure with data related to the previous data packet
[out]	antenna_data	The array of antenna-specific results

Get the detailed results of the IQ sample quality analysis.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1965 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_get_channel_details

enum sl_rtl_error_code sl_rtl_util_iq_sample_qa_get_channel_details (sl_rtl_util_libitem *item, uint8_t channel, sl_rtl_clib_iq_sample_qa_dataset_t *iq_sample_qa_results, sl_rtl_clib_iq_sample_qa_antenna_data_t *antenna_data)

Parameters

[in]	item	Pointer to the initialized Util libitem
[in]	channel	Radio channel to show results for.
[out]	iq_sample_qa_results	The data structure with data related to the last data packet using the requested channel
[out]	antenna_data	The array of antenna specific results

Get the detailed results of the IQ sample quality analysis for the specified radio channel.

Returns

- SL_RTL_ERROR_SUCCESS if successful

Definition at line 1976 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h

sl_rtl_util_iq_sample_qa_code2string

char * sl_rtl_util_iq_sample_qa_code2string (char *buf, int size, uint32_t code)

Parameters

[in]	buf	Buffer for the results to be written
[in]	size	Size of the buffer
[in]	code	The results previously received from the sl_rtl_util_iq_sample_qa_get_results()

Write the IQ sample quality analysis code as human readable strings in the provided buffer.

Returns

- pointer to the buffer containing data, or to a constant string

Definition at line 1986 of file /mnt/raid/workspaces/ws.Dg1ch1o1Y/overlay/gsdk/util/silicon_labs/rtl/inc/sl_rtl_clib_api.h