AN1296: Application Development with Silicon Labs' RTL Library

This application note provides guidance on how to start developing Bluetooth 5.1 Direction Finding (DF) applications using the Silicon Labs Bluetooth LE stack and the Real Time Locating Library (RTL lib).

Silicon Labs provides sample projects including examples for asset tags and locators to demonstrate Bluetooth LE 5.1-based DF applications. This document provides an overview of the sample projects, software architecture, and a step-by-step guide on how to create your own applications using Simplicity Studio 5 and Bluetooth SDK v3.x.

UG514: Using the Bluetooth® Direction Finding Tool Suite complements this document with the description of the Simplicity Studio tools that helps development with the RTL library.

Readers of this document should be familiar with the basics and terms of Direction Finding. To learn more about the theory of Direction Finding, see *UG103.18: Bluetooth® Direction Finding Fundamentals.* To get started with Silicon Labs Direction Finding Solution, read *QSG175: Silicon Labs Direction Finding Solution Quick-Start Guide*.

KEY POINTS

- Overview of Silicon Labs' sample applications
- Connection-based asset tag
- Connectionless asset tag
- Silicon Labs Enhanced asset tag
- Single locator sample app
- Positioning sample app

1 Introduction

Silicon Labs provides modularized software sample projects for Angle of Arrival (AoA) asset tags and locators that can be easily extended to address different use case scenarios.

In general, the sample applications can be grouped in two main categories:

- AoA asset tag sample app—demonstrates a Constant Tone Extension (CTE) transmitter.
- AoA locator sample app—demonstrates a CTE receiver.

1.1 AoA Asset Tags—CTE Transmitters

Beginning with version 3.1. the Bluetooth SDK provides an AoA asset tag sample project (***Bluetooth AoA - SoC Asset Tag***) that can easily be extended to address the following three scenarios by simply installing software components using Simplicity Studio 5's Project Configurator.

- Bluetooth 5.1 Connection-based AoA asset-tag
- Bluetooth 5.1 Connectionless AoA asset-tag
- Silicon Labs enhanced AoA asset-tag

1.2 AoA Locators—CTE Receivers

Beginning with version 3.1, the Bluetooth SDK also provides sample projects for AoA locators. Due to the resource-constrained nature of the EFR32 device, all the locator sample applications supported in Bluetooth SDK v3.x work in NCP (Network Co-Processor) mode. Thus, two sample applications are provided in the SDK to support both the NCP target and the NCP host of the locator:

- AoA NCP locator sample app
- AoA locator host sample app

The Bluetooth stack runs on the EFR32 (AoA locator target) and the application runs on a host (MCU or PC). While the NCP AoA locator is generic for all variants, the AoA locator host sample app must be configured for the desired type of CTE receiver mode during runtime (that is, connection-based, connectionless, or Silicon Labs enhanced).

AoA can be measured accurately using a single locator. However, a single locator can only provide a rough estimation of the asset tag's position. To determine the precise position of the asset tag, using multiple locators is recommended. By using multiple antenna arrays, the position of an asset tag can be determined using triangulation.

To demonstrate position estimation, the Bluetooth SDK also provides a sample host project supporting multiple locators—the positioning host application. In addition, the SDK also provides a Python based direction-finding visualization app for a multi-locator setup.

In summary, Bluetooth SDK v3.x offers the following examples that can be adapted for different use case scenarios:

- Studio Examples
  - bt_aoa_soc_asset_tag— ***Bluetooth AoA – SoC Asset Tag***
  - bt_aoa_ncp_locator—***Bluetooth AoA - NCP locator***
- Host Examples (app/Bluetooth/example_host)
  - bt_aoa_host_locator—*single AoA locator host for angle estimation*
  - bt_host_positioning—*host app for position estimation using multiple locators*
  - bt_host_positioning_gui—*Python host app for the visualization of a multi-locator setup*

1.3 Software Architecture

The following diagram provides an overview of the software architecture for the asset-tag, NCP AoA locator, and host sample applications provided by Silicon Labs.

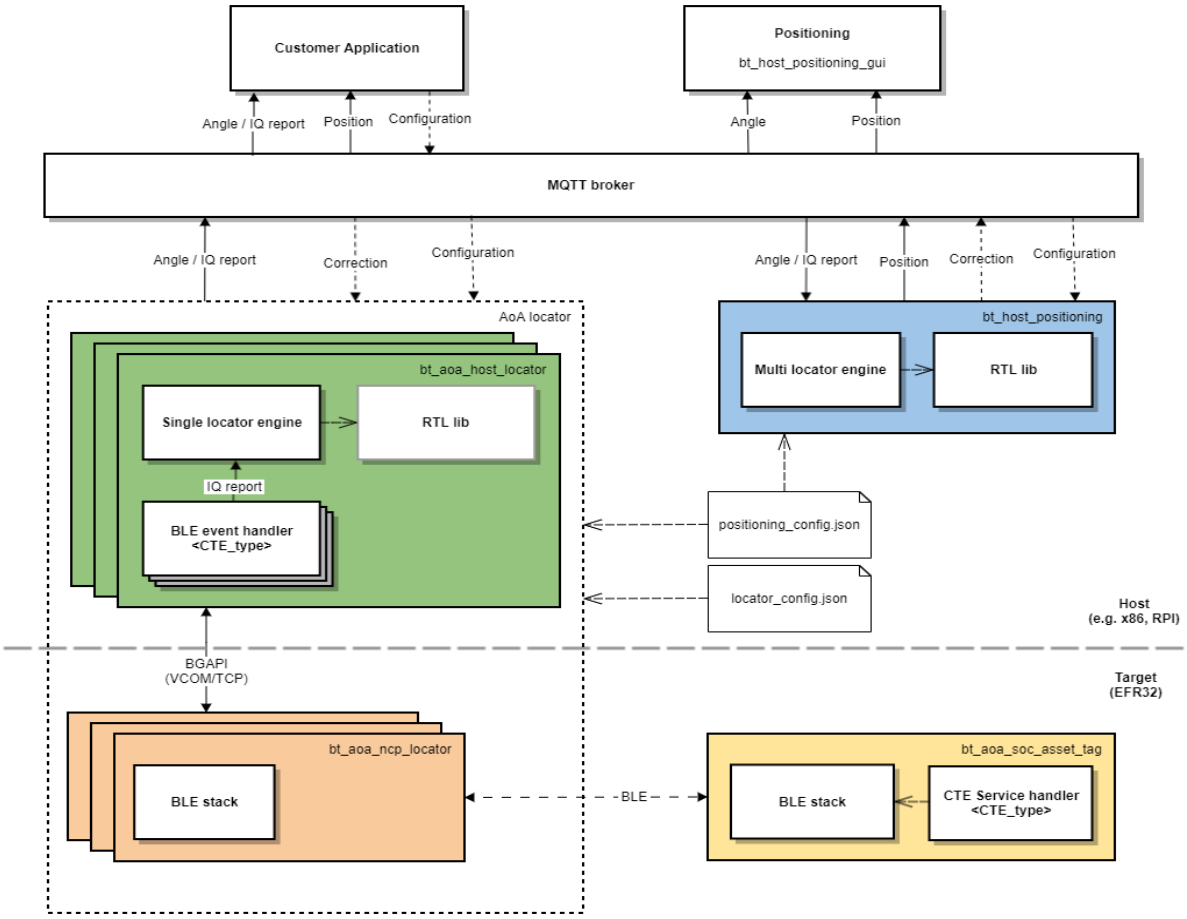


Figure 1-1. Software Architecture for the Asset-Tag, NCP AoA Locator, and Host Sample Applications

The yellow and orange boxes represent the AoA asset tag and NCP AoA locator sample projects, respectively. These example projects are meant to run on an EFR32xG22 or on an EFR32xG24 device.

The green box represents the AoA locator host sample app. This sample app is meant to run on a host machine (for example, x86, Raspberry Pi). The orange and green boxes logically form a single locator (a CTE receiver).

The AoA locator host (green box) connects to the NCP AoA locator (orange box) via a serial port (VCOM) or TCP/IP. In the latter case, the AoA locator host and NCP AoA locator can be in different locations.

Silicon Labs AoA sample projects utilize the MQTT messaging protocol for sending and receiving the asset tag's angle and position information, and making changes to the runtime configuration parameters. MQTT is a publish and subscribe messaging exchange protocol where a publisher sends, and the subscriber receives topics of interest via a message broker. The subscribers and publishers in the MQTT protocol do not interact with each other. The connection between them is handled by the broker. The broker filters all incoming messages and distributes them to the subscribers.

The AoA locator host controls the Bluetooth stack running on the NCP AoA locator and receives the CTE IQ samples (In-Phase and Quadrature-Phase pair of readings) using BGAPI protocol. Using the RTL library, the AoA locator host calculates the Angle of Arrival of an asset tag and publishes the result to the MQTT broker (white box in the figure above).

The positioning host (that is, the box in blue) represents the positioning host sample project. This host application subscribes to MQTT topics (related to the angle data) published by single locators to calculate and publish the exact position of an asset tag in an X, Y, Z coordinate.

1.4 Prerequisites

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

- EFR32xG22 or EFR32xG24 based device serving as a tag (for example, Thunderboard BG22).
- 4x4 antenna array board: one for the single locator sample app, more than one for the positioning sample app.
- A Wireless Starter Kit (WSTK) for each antenna array board.
- Simplicity Studio v5 installed on your PC.
- Bluetooth 3.1.0 or higher SDK installed from Gecko SDK Suite v3.1.0 or later.
- MinGW64 for building the AoA locator host applications, if you are using a Windows PC as a host.
- Mosquitto MQTT Broker— https://mosquitto.org/download/.
- MQTT Explorer (optional)— http://mqtt-explorer.com/.
- Python 3.7 for visualization purposes.

1.5 Tools

As well as the sample applications, Silicon Labs also provides tools inside Simplicity Studio to help development with the RTL library. This set of tools is called the Silicon Labs Direction Finding Tool Suite, and it is documented in *UG514: Using the Bluetooth® Direction Finding Tool Suite*.

The tools can be used for creating configurations files, which are needed by the host application, as discussed later. Furthermore, they are also able to consume the configuration files, and run the RTL library to calculate and display angles and positions without the need of any host application. If you want to test the RTL library without building the host application first, continue with *UG514: Using the Bluetooth® Direction Finding Tool Suite* while referring to this document for further details.

Note: The tools were created for demonstration purposes, and not to be used in production. For development purposes, please start with the sample application described in this document.

2 Bluetooth AoA- SoC Asset Tag

Asset tags are relatively simple as their only goal is to send CTEs on a single antenna. However, CTEs can be sent in several different ways depending on the use case. Bluetooth SDK v3.1 provides an asset tag sample project, ***Bluetooth AoA- SoC Asset Tag***, that can easily be extended to address the following three scenarios by installing software components using Simplicity Studio 5's Project Configurator.

- Bluetooth 5.1 Connection-based AoA asset-tag—sends CTE responses on a connection when a CTE request is received.
- Bluetooth 5.1 Connectionless AoA asset-tag— sends CTE in periodic advertisements.
- Silicon Labs Enhanced (Silicon Labs proprietary) AoA asset-tag—sends CTE in extended advertisements.

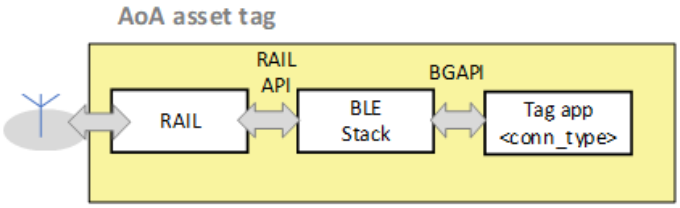


Figure 2-1. Bluetooth – SoC Asset Tag

At its core, the ***Bluetooth AoA - SoC Asset Tag*** is simply a ***Bluetooth – SoC Empty*** sample project extended with a CTE transmitter and an Asset Tracking Profile (ATP) in its GATT database. These properties can be added by installing one or two of the following software components, depending on the use case.

- *Constant Tone Extension GATT Service (Connection)*
- *Constant Tone Extension GATT Service (Connectionless)*
- *Constant Tone Extension GATT Service (Silabs proprietary)*

By default, the ***Bluetooth AoA - SoC Asset Tag*** has the *Constant Tone Extension GATT Service (Connection)* and *Constant Tone Extension GATT Service (Silabs proprietary)* component preinstalled in the project. The *Constant Tone Extension GATT Service (Connection)* component has dependency on *AoA Transmitter*. Therefore, it is installed in the background. The *AoA Transmitter* component enables initializing the CTE transmitter.

In addition to enabling the AoA transmitter, the *Constant Tone Extension GATT Service (Connection)* component also contributes to the Bluetooth GATT configuration, which is the *Constant Tone Extension Service* with a *Constant Tone Extension Enable* characteristic, as shown by *(a)* in the following figure. The *Constant Tone Extension Enable* characteristics is mandatory, and thus must be included in Connection, Connectionless, and Silicon Labs proprietary tag implementations.

On the other hand, the *Constant Tone Extension GATT Service (Silabs Proprietary)* component allows broadcasting CTEs in extended advertisements. This component contributes the following characteristics to the *Constant Tone Extension GATT Service* which can be used to alter different CTE parameters:

- Advertising Constant Tone Extension Minimum Length
- Advertising Constant Tone Extension Minimum Transmit Count
- Advertising Constant Tone Extension Transmit Duration
- Advertising Constant Tone Extension Interval
- Advertising Constant Tone Extension PHY

These characteristics are mandatory when CTE transmission is supported on advertising channels, which is the case for the Connectionless and Silicon Labs proprietary approaches.

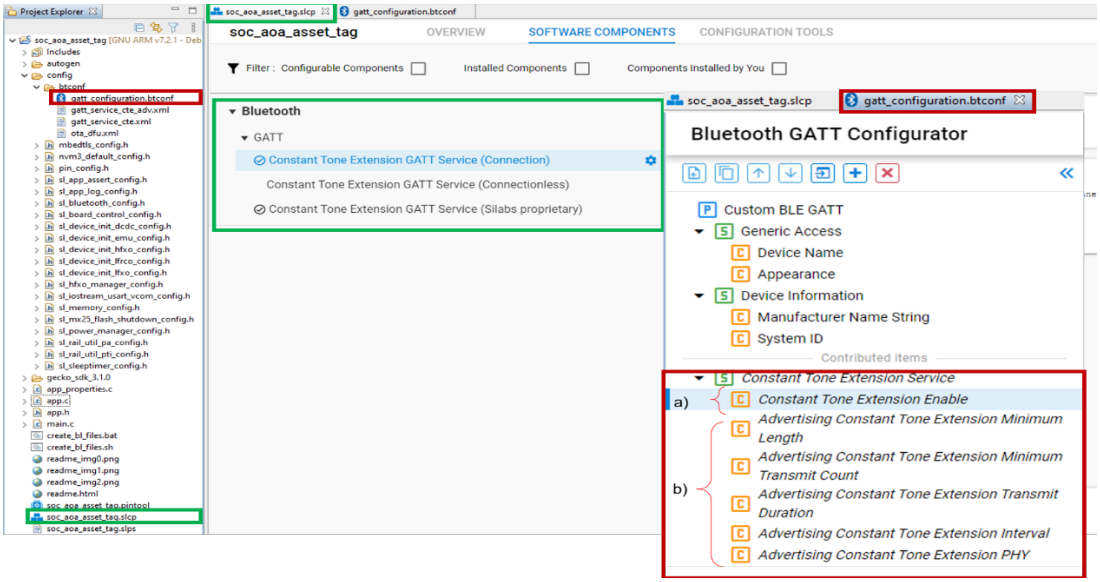


Figure 2-2. Bluetooth AoA – SoC Asset Tag Sample Project

The ***Bluetooth AoA - SoC Asset Tag*** sample project enables CTE transmission, and automatically adds the CTE Service specified by the Bluetooth SIG (https://www.bluetooth.com/specifications/specs/) to the GATT database. Since a locator device finds the asset tag by looking for this service in the advertising packets, advertising CTE service is also enabled in the sample project by default (see "**advertise service**" checked in the Bluetooth GATT configurator in the figure below).

Note: In Bluetooth SDK v3.1, a temporary UUID was used for the CTE service. In Bluetooth SDK v3.2, the official UUID of the CTE service is supported. This means that tags programmed with v3.1 are not compatible with locators programmed with v3.2 and vica versa.

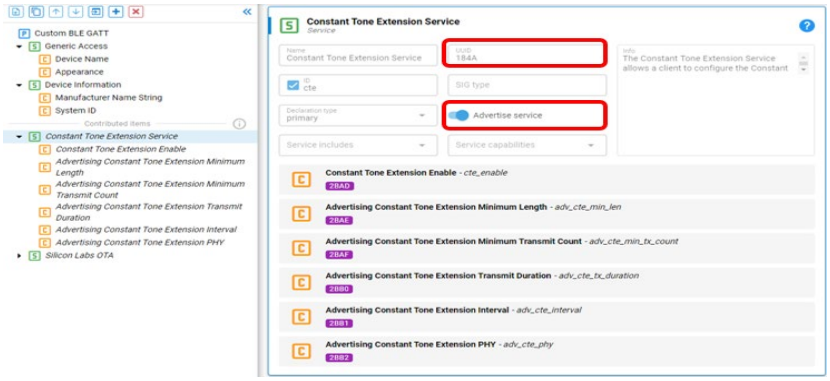


Figure 2-3. Bluetooth AoA – SoC Asset Tag CTE Service

2.1 Connection-Based Asset Tag Sample Application

A connection-based asset tag needs to implement:

- A connectable Bluetooth peripheral that starts advertising itself
- CTE transmitter to be able to send CTE responses, and
- ATP in its GATT database.

As mentioned above, the *Constant Tone Extension GATT Service (Connection)* component implements both the CTE transmitter and ATP in the GATT database of your project. As such, for the connection-based tag, the ***Bluetooth AoA - SoC Asset Tag*** sample app works out of the box without having to install any additional software components. You can uninstall the *Constant Tone Extension GATT Service (Silabs Proprietary)* component if your custom project does not need to send CTEs on extended advertising.

The CTE transmitter is initialized by calling the `sl_bt_init_classes()`. This API is automatically added in the Bluetooth initialization by the *AoA Transmitter* component. The *AoA Transmitter* component is added to the project (in the background) when the *Constant Tone Extension GATT Service (Connection)* is installed, which is the default in the ***Bluetooth AoA - SoC Asset Tag*** sample app.

```
main() -> sl_system_init() -> sl_stack_init() -> sl_bt_init() -> sl_bt_init_classes()
```

CTE transmission is enabled by `sl_bt_cte_transmitter_enable_connection_cte()` after a locator device connects to the tag and writes 0x01 into the *Constant Tone Extension Enable* characteristics. This process is handled by `sl_gatt_service_cte_on_event()` when the `sl_bt_evt_gatt_server_user_write_request` event is triggered. `sl_gatt_service_cte_on_event()` is defined in `sl_gatt_service_cte.c`, which is generated inside the *gecko_sdk_3.1.x/app/bluetooth/common/gatt_service_cte* directory by the *Constant Tone Extension GATT Service (Connection)*.

It is important that the "Write" property of the *Constant Tone Extension Enable* characteristics is enabled in the Bluetooth GATT configurator service (see "Write" checked under Properties).

To test a connection-based asset tag application:

1. Create a Bluetooth AoA - SoC Asset Tag project in Simplicity Studio 5.
2. Build the project.
3. Flash it to an EFR32xG22 or EFR32xG24 device. Note: If you use a new Thunderboard, push its reset button before programming. On some boards, the factory default firmware puts the device into EM4 after 30 seconds, and in this case the device must be restarted to be accessible by the programmer.
4. The sample does not contain a bootloader. By default, a new device is factory-programmed with a bootloader. If you have a new device, haven't cleared the bootloader region for your part or have a supported bootloader image already flashed on your device, you do not need to flash a bootloader. Once you have installed a bootloader image, it remains installed until you erase the device. If you need to load a bootloader, select an example, such as **SPI Flash Storage Bootloader (single image)**, and build it and flash it as described above. For more information about the Gecko Bootloader, see *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

2.2 Connectionless Asset Tag Sample Application

A connectionless asset tag needs to implement:

- A periodic advertiser.
- CTE transmitter – to be able to send CTEs in periodic advertisements.
- ATP (Asset Tracking Profile) in its GATT database.

The CTE transmitter is initialized by calling the `sl_bt_init_classes()`. This API is automatically added in the Bluetooth initialization by the *AoA Transmitter* component. This component is added to the project (in the background) when the *Constant Tone Extension GATT Service (Connection)* is installed, which is the default in the ***Bluetooth AoA- SoC Asset Tag*** sample app.

```
main() -> sl_system_init() -> sl_stack_init() -> sl_bt_init() -> sl_bt_init_classes()
```

The connectionless version of the asset tag is rather simple. It starts periodic advertising with CTE enabled, and that is all. To achieve this, remove the *Constant Tone Extension GATT Service (Silabs Proprietary)* and install the *Constant Tone Extension GATT Service (connectionless)* component.

The *Constant Tone Extension GATT Service (Connectionless)* component also contributes to the Bluetooth GATT configurator. It adds the following characteristics to the *Constant Tone Extension GATT Service*, which can be used to alter different CTE parameters:

- Advertising Constant Tone Extension Minimum Length
- Advertising Constant Tone Extension Minimum Transmit Count
- Advertising Constant Tone Extension Transmit Duration
- Advertising Constant Tone Extension Interval
- Advertising Constant Tone Extension PHY

It is important that the "**Write**" property of each of these characteristics is enabled in the Bluetooth GATT configurator service (see "Write" checked under Properties).

In a nutshell, when the tag is booted, it initializes the CTE transmitter and periodic advertising feature, and enters an infinite loop of processing Bluetooth stack events.

```
void sl_bt_process_event(sl_bt_msg_t *evt)
{
  sl_bt_ota_dfu_on_event(evt);
  sl_gatt_service_cte_on_event(evt);
  sl_gatt_service_cte_adv_on_event(evt);
```

```
  sl_bt_on_event(evt);
}
```

The `sl_gatt_service_cte_adv_on_event()` handles events related to system boot and user write requests. When a system boot event is triggered, CTE advertising is initialized and started automatically by this handler using `adv_cte_init()` and `adv_cte_start()`, respectively. These functions are defined in `sl_gatt_service_cte_adv.c` and `sl_gat_service_cte_connectionless.c`, generated inside *gecko_sdk_3.1.x/app/Bluetooth/common/gatt_service_cte_adv* directory when the ***Constant Tone Extension GATT Service (Connectionless)*** component is installed.

`adv_cte_init()` is called only once during the init phase. It sets the default parameter values for the CTE advertising. In contrast, `adv_cte_start()` is triggered during the initialization and when the user write request for updating the value of one of the connectionless CTE characteristics (listed above) is completed. The `adv_cte_start()` normally sets the advertising phy, starts a connectionless advertising, and adds CTEs to the periodic advertisements using `sl_bt_cte_transmitter_enable_connectionless_cte()`.

To test the connectionless asset tag application:

1. Create Bluetooth AoA - SoC Asset Tag project in Simplicity Studio 5.
2. Uninstall the Constant Tone Extension GATT Service (Silabs Proprietary) component.
3. Install the Constant Tone Extension GATT Service (Connectionless) component using the Project Configurator.
4. Build the project.
5. Flash it to an EFR32xG22 or EFR32xG24 device. Note: If you use a new Thunderboard, push its reset button before programming. On some boards, the factory default firmware puts the device into EM4 after 30 seconds, and in this case the device must be restarted to be accessible by the programmer.
6. The sample does not contain a bootloader. By default, a new device is factory-programmed with a bootloader. If you have a new device, haven't cleared the bootloader region for your part or have a supported bootloader image already flashed on your device, you do not need to flash a bootloader. Once you have installed a bootloader image, it remains installed until you erase the device. If you need to load a bootloader, select an example, such as **SPI Flash Storage Bootloader (single image)**, and build it and flash it as described above. For more information about the Gecko Bootloader, see *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Note that you cannot uninstall the *Constant Tone Extension GATT Service (Connection)* in a connectionless asset tag as it is the only component that installs the *AoA Transmitter* and contributes the *Constant Tone Extension Enable* characteristic, which is mandatory for the *Constant Tone Extension Service*.

2.3 Silicon Labs Enhanced (Silabs proprietary) Asset Tag Sample Application

The Silicon Labs asset tag sample app sends CTEs in extended advertisements, which is not a standard solution, and therefore can only be used with Silicon Labs locators. The advantage of this solution is that it uses extended advertisements, in which case no synchronization information needs to be stored on the locator for each tag, in contrast to connections and periodic advertisements. This solution scales much better than the other two and can be used with hundreds of tags.

This sample app implements:

- An advertiser broadcasting extended advertisements.
- CTE transmitter – to be able to send CTEs in extended advertisements.
- ATP (Asset Tracking Profile) in its GATT database.

The difference between the connectionless version and the Silicon Labs proprietary version of the asset tag sample app is that the Silicon Labs solution starts extended advertising with CTE instead of periodic advertising. This offers better scalability since it puts no RAM constraints on the receiver side, like connections and periodic advertisement synchronizations.

The CTE transmitter is initialized by calling the `sl_bt_init_classes()`. This API is automatically added in the Bluetooth initialization by the *AoA Transmitter* component. This component is added to the project (in the background) when the *Constant Tone Extension GATT Service (Connection)* is installed, which is the default in the ***Bluetooth AoA- SoC Asset Tag*** sample app.

```
main() -> sl_system_init() -> sl_stack_init() -> sl_bt_init() -> sl_bt_init_classess_cte()
```

To broadcast CTEs in extended advertisement, the *Constant Tone Extension GATT Service (Silabs Proprietary)* software component must be installed, which is the default in the *Bluetooth AoA - SoC Asset Tag* sample app. As such, for the Silicon Labs proprietary tag, the ***Bluetooth AoA - SoC Asset Tag*** sample app works out of the box without having to install any additional software components.

Similar to the connectionless version, the ***Constant Tone Extension GATT Service (Silabs Proprietary)*** component adds the following characteristics to the *Constant Tone Extension GATT Service*, which can be used to alter different CTE parameters:

- Advertising Constant Tone Extension Minimum Length
- Advertising Constant Tone Extension Minimum Transmit Count
- Advertising Constant Tone Extension Transmit Duration
- Advertising Constant Tone Extension Interval
- Advertising Constant Tone Extension PHY

It is important that the "**Write**" property of each of these characteristics is enabled in the Bluetooth GATT configurator service (see "Write" checked under Properties).

In a nutshell, when the tag is booted, it initializes the CTE transmitter, and enters an infinite loop of processing Bluetooth stack events.

```
void sl_bt_process_event(sl_bt_msg_t *evt)
{
  sl_bt_ota_dfu_on_event(evt);
  sl_gatt_service_cte_on_event(evt);
  sl_gatt_service_cte_adv_on_event(evt);
  sl_bt_on_event(evt);
}
```

The `sl_gatt_service_cte_adv_on_event()` handles events related to system boot and user write requests. When a system boot event is triggered, CTE advertising is initialized and started automatically by this handler using `adv_cte_init()` and `adv_cte_start()`, respectively. These functions are defined in `sl_gatt_service_cte_adv.c` and `sl_gat_service_cte_silabs.c`, generated inside *gecko_sdk_3.1.x/app/Bluetooth/common/gatt_service_cte_adv* directory when the *Constant Tone Extension GATT Service (Silabs Proprietary)* component is installed.

`adv_cte_init()` is called only once during the init phase. It sets the default parameter values for the CTE advertising. Whereas, `adv_cte_start()` is triggered during the initialization and when a user write request for updating the value of one of the connectionless CTE characteristics (listed above) is completed. The `adv_cte_start()` normally sets the advertising phy, starts a connectionless advertising, and adds CTEs to the extended advertisements using `sl_bt_cte_transmitter_enable_silabs_cte()`.

To test the Silicon Labs proprietary asset tag application:

1. Create a ***Bluetooth AoA - SoC Asset Tag*** project in Simplicity Studio 5.
2. Build the project.
3. Flash it to an EFR32xG22 or EFR32xG24 device. Note: if you use a new Thunderboard, push its reset button before programming. On some boards, the factory default firmware puts the device into EM4 after 30 seconds, and in this case the device must be restarted to be accessible by the programmer.
4. The sample does not contain a bootloader. By default, a new device is factory-programmed with a bootloader. If you have a new device, haven't cleared the bootloader region for your part or have a supported bootloader image already flashed on your device, you do not need to flash a bootloader. Once you have installed a bootloader image, it remains installed until you erase the device. If you need to load a bootloader, select an example, such as **SPI Flash Storage Bootloader (single image)**, and build it and flash it as described above. For more information about the Gecko Bootloader, see *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

Note that you cannot uninstall the *Constant Tone Extension GATT Service (Connection)* in a Silicon Labs proprietary asset tag as it is the only component that installs the *AoA Transmitter* and contributes the *Constant Tone Extension Enable* characteristic, which is mandatory for the *Constant Tone Extension Service*.

The Bluetooth specification allows CTEs to be added to periodic advertisements only. Therefore, this is a proprietary, non-standard solution. It can, however, help you scale your system. In the case of periodic advertisement, the locator must keep track of each periodic advertiser one-by-one. For each, it needs to know when and on which channel to expect the next packet. With hundreds of tags, this can result in huge RAM consumption, making this solution less scalable. In contrast to this, to receive an extended advertisement the locator must scan on the primary advertising channels only, listen for legacy advertisements that point to extended advertisements, and jump to the reported channel at the reported time. No time/channel tracking is needed; therefore, hundreds of tags can be followed.

3 Single Locator Sample Application

Locators are much more complicated than asset tags. They control an array of antennae (not a single antenna), they must precisely sample the incoming signal, and optionally must also calculate the angle values from the received signal. Because of the limited capabilities of the EFR32, all the locator sample applications supported in Bluetooth SDK v3.x work in NCP (Network Co-Processor) mode, meaning that the Bluetooth stack runs on the EFR32 (NCP target) and the application runs on a host (MCU or PC).

The Bluetooth SDK v3.x provides one sample project for the EFR32 NCP AoA locator target (***Bluetooth AoA - NCP locator***) and one sample app for a locator host (***bt_aoa_host_locator***). The NCP target sample project can be found in Simplicity Studio, and the host sample app can be found in the SDK folder inside <SDK Installation location>*app/bluetooth/example_host*. The default Windows SDK installation locations are:

GSDK 3.x: *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>*

GSDK 4.0 and higher: *C:\Users\<NAME>\SimplicityStudio\SDKs\gecko_sdk*

While the NCP AoA locator application is unified for all variants (connection-based, connectionless and Silicon Labs proprietary), the locator host should be configured with the right CTE mode during runtime.

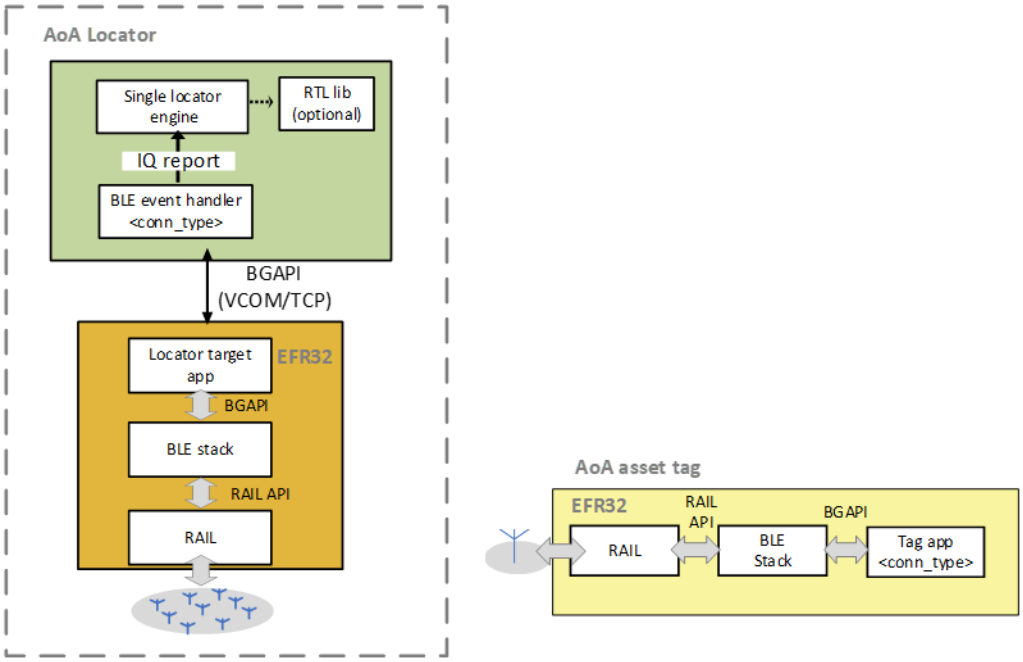


Figure 3-1. AoA Locator

The physical interface between the AoA locator host and NCP AoA locator can be either virtual COM port over USB (VCOM) or TCP/IP. In the latter case, the WSTK with the antenna board can be decoupled in space from the host if the host can reach the NCP AoA locator using its IP address.

3.1 NCP AoA Locator Sample Application

The Bluetooth SDK v3.x provides the ***Bluetooth AoA - NCP locator*** example project to support an NCP AoA locator that can receive and sample CTEs transmitted by asset tags.

The NCP AoA locator (EFR32) is responsible for:

- Running the Bluetooth stack
- Enabling the CTE receiver feature
- IQ sampling
- Antenna switching
- Mirroring the BGAPI interface to UART

The CTE receiver feature is enabled automatically by installing the *AoA Receiver* components, which is default in the ***Bluetooth AoA - NCP locator*** project.

Also, the ***Bluetooth AoA - NCP locator*** project has two main additional components:

- ***Periodic Advertising Synchronization***—enables the periodic advertising synchronization feature.
- ***RAIL Utility, AoX***—supports antenna pin configuration.

To test the NCP AoA locator application:

1. Create Bluetooth AoA - NCP locator project in Simplicity Studio 5.
2. Build the project.
3. Flash it to an EFR32xG22 device with an antenna array (that is, to a Silicon Labs Direction Finding board).
4. The sample does not contain a bootloader. By default, a new device is factory-programmed with a bootloader. If you have a new device, haven't cleared the bootloader region for your part or have a supported bootloader image already flashed on your device, you do not need to flash a bootloader. Once you have installed a bootloader image, it remains installed until you erase the device. If you need to load a bootloader, select an example, such as **SPI Flash Storage Bootloader (single image)**, and build it and flash it as described above. For more information about the Gecko Bootloader, see *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*.

3.2 Single AoA Locator Host Application

The AoA locator host application is responsible for:

- Controlling the stack, for example to find tags, connect to them, sync on periodic advertisings, and so on.
- Initializing a buffer for IQ samples, and receiving the IQ samples from the stack.
- Using the RTL library to calculate angles from the IQ samples (optional).
- Publishing of angle/IQ reports, and subscription of config and correction topics to the MQTT broker.

The AoA locator relies on several software components to accomplish the above-mentioned tasks. The main software components that characterize the operation of the AoA locator are ***aoa_angle*** and ***aoa_cte***. The aoa_angle software component is responsible for feeding the raw IQ samples to the RTL library and calculate the angle values. On the other hand, the aoa_cte component handles the implementation of different CTE receiving modes—connection, connectionless, and Silabs modes. The type of CTE mode can be set at runtime via MQTT, without recompiling or restarting the application manually.

3.2.1 Connection Based CTE Receiving Locator Host

In a connection based CTE mode, the locator must:

- Find a connectable advertising tag with a CTE service.
- Connect to the asset tag.
- Discover the GATT database.
- Enable CTE using the Constant Tone Extension Enable characteristic.
- Send a CTE request and receive a CTE response.
- Collect and convert IQ samples into common IQ report format.

Once the IQ report is available, the angle is calculated and published to the MQTT broker using the `aoa_cte_on_iq_report()` callback. These are all done in the `cte_conn.c` file found in the ***app/bluetooth/common_host/aoa_cte*** directory.

In the background, the `aoa_cte_on_iq_report()` calls the `aoa_calculate()` API, which leverages the RTL library to calculate the angle. The angle data is then published to the MQTT report by calling `mqtt_publish()`. These are done in the `app.c` file of the ***bt_aoa_host_locator*** sample project.

3.2.2 Connectionless CTE Receiving Locator Host

CTEs can also be received in connectionless mode using Bluetooth periodic advertisements. In this case, the locator must:

- Find the asset tag with a CTE service by its advertisement.
- Sync on the periodic advertisement.
- Enable CTE receiver in connectionless mode using periodic advertising.

- Collect and convert IQ samples into a common IQ report format.

Once the IQ report is available, the angle is calculated and published to the MQTT broker using the `aoa_cte_on_iq_report()` callback. These are all done in the `cte_conn_less.c` file found in the ***app/bluetooth/common_host/aoa_cte*** directory.

In the background, the `aoa_cte_on_iq_report()` calls the `aoa_calculate()` API, which leverages the RTL library to calculate the angle. The angle data is then published to the MQTT report by calling `mqtt_publish()`. These are done in the `app.c` file of the **bt*_aoa_host_locator*** sample project.

3.2.3 Silabs Proprietary CTE Receiving Locator Host

CTEs can also be received in extended advertising using Silabs mode. In this approach, the locator must:

- Find the asset tag by its advertisement.
- Enable receiving CTE in Silicon Lab's enhanced mode.
- Collect and convert IQ samples into common IQ report format.

Once the IQ report is available, the *angle* is calculated and published to the MQTT broker using the `aoa_cte_on_iq_report()` callback. These are all done in the `cte_silabs.c` file found in the ***app/bluetooth/common_host/aoa_cte*** directory.

In the background, the `aoa_cte_on_iq_report()` calls the `aoa_calculate()` API, which leverages the RTL library to calculate the angle. The angle data is then published to the MQTT report by calling `mqtt_publish()`. These are done in the `app.c` file of the ***bt_aoa_host_locator*** sample project.

3.3 Building the Single AoA Locator Host Sample Application

3.3.1 Exporting AoA Locator Example Project

Before starting to work with the AoA host example, Silicon Labs recommends that you generate the project using the `export` feature of the makefile. This feature allows coping all the 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 GSDK but has the same folder structure inside.

To generate your AoA locator host project into the export directory, open a terminal and change to the ***example_host/bt_aoa_host_locator*** directory.

```
cd $GSDK_DIR/app/bluetooth/example_host/bt_aoa_host_locator
```

Now call the following command:

```
make export
```

Custom export folders can be specified using the `EXPORT_DIR` variable.

```
make export EXPORT_DIR=/my/custom/export/path
```

The benefits of exporting are twofold. First, the changes to the source code during development will not affect the GSDK content. Second, multiple instances can coexist, for example, for testing different variants. In addition, with the exported folder, it is clear at glance which files belong to the project.

3.3.2 Building AoA Locator on Windows

At the moment, the only supported build environment on Windows is MinGW-64. The makefiles make sure that the gcc is used with a proper prefix (**x86_64-w64-mingw32-**). The recommended build environment on Windows is **MSYS2**.

1. Download and install MSYS2: https://www.msys2.org/.
2. Open the Mintty bash. Make sure to start Mingw-w64 64 (**mingw64.exe**) when launching Mintty. 32-bit versions of MYSYS2 will not work.
3. Install additional packages.
   ```
   pacman -S make mingw-w64-x86_64-gcc
   ```

You must also install Mosquitto MQTT broker before compiling the host application. The makefile makes sure that it copies the necessary client library files from the installation directory. Download and install Mosquitto broker if you have not done it yet: https://mosquitto.org/download/.

To build the project:

- Change to the exported project directory
  ```
  cd ~/export/app/bluetooth/example_host/bt_aoa_host_locator
  ```
- Build the project using the `make` command.

3.3.3 Building AoA Locator on Linux

The RTL library is built for Ubuntu 18 LTS 64-bit. The makefile is written so that it recognizes the Linux environment and automatically uses the Linux version of the RTL library.

To build the project:

- Install mosquitto libraries if it's not installed yet.
  ```
  sudo apt install libmosquitto-dev
  ```
- Change to the exported project folder
  ```
  cd ~/export/app/bluetooth/example_host/bt_aoa_host_locator
  ```
- Build the project using the `make` command.

3.3.4 Building AoA Locator on Embedded Linux

Cross compilation is not supported in the makefiles. Instead, there is a build target called `export` that collects all the dependencies from the Bluetooth SDK into a folder called **export** created next to the makefile. Then the **export** folder should be copied to the target Cortex-A device and compiled there. Perform the following steps from the host computer with the Bluetooth SDK installed:

- Change to the ***example_host/bt_aoa_host_locator*** directory.
  ```
  cd $GSDK_DIR/app/bluetooth/example_host/bt_aoa_host_locator
  ```
- Export GSDK files.
  ```
  make export
  ```
- Copy exported GSDK files to the home folder of the target Cortex-A device (for example, Raspberry Pi).
  ```
  scp -r export pi@raspberrypi.local:~
  ```
- Start SSH connection with Raspberry Pi.
  ```
  ssh pi@raspberrypi.local
  ```
- Install mosquitto libraries if it's not installed yet.

```
sudo apt install libmosquitto-dev
```

- Change to the exported project folder
  ```
  cd ~/export/app/bluetooth/example_host/bt_aoa_host_locator
  ```
- Build the project using the `make` command.

3.4 AoA Locator Configuration

The AoA locator configuration is an important step in angle estimation. The host application, of course, can be started without a configuration when using the default settings with Silicon Lab's dual polarized antenna board (i.e BRD4191A). In all other cases, modifying the configuration before or during runtime becomes mandatory.

The SDK host example project provides a template JSON config file *locator_config.json* inside the ***config*** folder. The host application can parse the config JSON passed through CLI input. The file has a simple JSON format containing key/value pairs of different configuration option, including the antenna type (mode), CTE configuration, angle estimation parameters, the allowed list of tags, and angle masks. A prototype of a single locator's configuration is shown below.

```
{
   "version": 1,
   "aoxMode":"SL_RTL_AOX_MODE_REAL_TIME_BASIC",
   "antennaMode":"SL_RTL_AOX_ARRAY_TYPE_4x4_DP_URA",
   "antennaArray":[0,1,2,3,4,5,6,7,8,9,10,11,12,13,14,15],
   "angleFiltering":true,
   "angleFilteringWeight":0.6,
   "angleCorrectionTimeout":5,
   "angleCorrectionDelay":3,
   "cteMode":"SILABS",
   "cteSamplingInterval":3,
   "cteLength":20,
   "slotDuration":1,
   "reportMode":"ANGLE",
   "allowlist":[
      "ble-pd-AAAAAAAAAAAA",
      "ble-pd-BBBBBBBBBBBB"
   ],
   "azimuthMask":[
      {
         "min":0.0,
         "max":10.0
      },
      {
         "min":20.0,
         "max":30.0
      }
   ],
   "elevationMask":[
      {
         "min":0.0,
         "max":10.0
      },
      {
         "min":20.0,
         "max":30.0
      }
   ]
}
```

3.4.1 Antenna Mode

The **antennaMode** specifies the type of antenna array board supported by the RTL library. The default antenna type is SL_RTL_AOX_ARRAY_TYPE_4x4_DP_URA — Silicon Lab's 4x4 dual polarized antenna array (i.e BRD4191A). Refer to the API reference manual for all other antenna types/modes supported by the RTL library.

3.4.2 Antenna Array

The **antennaArray** defines the antenna switching sequence according to the antenna numbers shown on the board, with index starting from 0. To understand the relation between the antenna switching sequence and the board's pin logic, refer to A Silicon Labs Dual Polarized Antenna Array Board (BRD4191A).

3.4.3 CTE Mode

The **cteMode** determines the type of CTE receiving mode of the locator. The default CTE mode is SILABS. To track asset tags transmitting CTE in other modes, you must set the value of this config parameter accordingly. Set this attribute to:

- CONN for connection based CTE mode.
- CONN_LESS for connectionless CTE mode.

3.4.4 Report Mode

The **reportMode** determines whether the host application calculates angle or publishes the raw IQ data received from the NCP target directly to the MQTT broker. The default report mode is ANGLE.

The report mode is an important configuration parameter especially if you want to use your own custom algorithm to calculate angle from raw IQ data. In this case, enable IQ sample report mode using IQREPORT value for this parameter.

3.4.5 Allowed List

The **allowList** tells the locator the list of asset tags to track. This improves the performance of the system by searching only asset tags that are only in the list.

Note that the template config JSON file has pseudo addresses of tags for the allowed list. If you start the host application using the config option, please make sure that you have modified the list with the right addresses of your tags. You can also empty the "allowList" like below to track every tag that the locator can find.

```
"allowList":[
  ]
```

3.4.6 Angle Masks

The **azimuth and elevation masks** in the locator's config file tell the estimator in which range NOT to search for the tag. For instance, if a locator is next to a wall, and the asset tag is to be searched in the room, then the locator should look for an azimuth Angle of Arrival in a 180° range instead of a 360° range. Similarly, if the locator is in the corner, it may be enough to search in a 90° angle range only. This improves both reliability and computation time. When the azimuth and elevation masks are configured, the RTL lib will set the weighting of the angle values inside the masked region to zero and will not return values inside the masked region. Instead, it chooses the next highest value outside the masked region. A locator may have more than one angle masks, for both azimuth and elevation.

The azimuth and elevation masks are independent of the coordinate system and are related to angle directions of each individual boards. 0°, 45°, 90°, 135°, 180°, -135°, -90°, and -45° directions are indicated on the antenna array board. Once you place the board, you can easily tell in which direction you want to search for the tag. Consider the following setup where the blue area is the desired tracking space. The red lines on the locators indicate the masked regions where the estimator will not search for the tag.

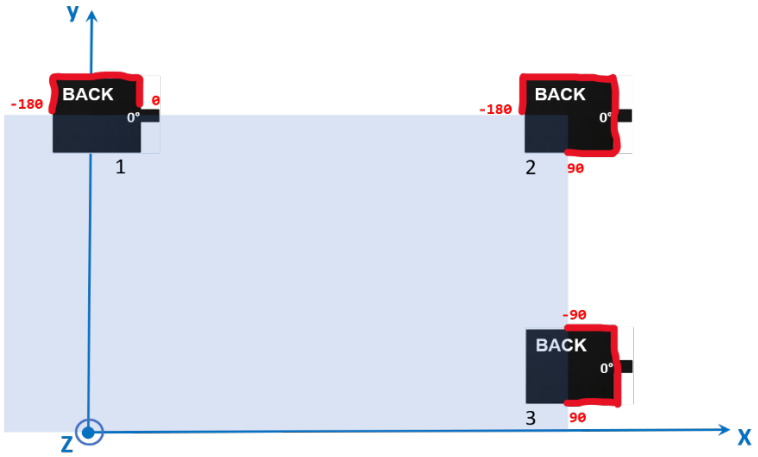


Figure 3-2. Azimuth mask configuration

For this setup, the config files for each locator should look like this:

config_locato1.json

```
{
  "azimuth_mask": {
      "min": -180.0,
      "max": 0.0
  }
}
```

config_locato2.json

```
{
  "azimuth_mask": {
      "min": -180.0,
      "max": 90.0
  }
}
```

config_locator3.json

```
{
  "azimuth_mask": {
      "min": -90.0,
      "max": 90.0
  }
}
```

Note that the template config JSON file contains a range of azimuth and elevation angle mask values. If you start the host application using the config option, please make sure that you have modified the values according to your need. You can also empty the arrays as below to track the tags on a full range.

```
"azimuthMask":[
    ],
    "elevationMask":[
    ]
```

3.4.7 Angle Correction Time Out and Angle Correction Delay

These parameters control the logic for how long the corrections from the angle feedback feature of the positioning host last after issuing one. Refer to Section 4.3 Running the Positioning Sample App to learn more about the angle feedback mechanism.

The **angleCorrectionTimeOut** value indicates that the angle correction will be cleared if angleCorrectionTimeOut amount of IQ reports are received without receiving a new correction message.

The **angleCorrrectionDelay** value indicates that the correction values with a sequence number more than angleCorrectionDelay apart from the last report are considered outdated and will be ignored.

The angle parameters, including angle filtering, correction timeout, etc, override the default values of the corresponding configuration parameters defined in *app/Bluetooth/common_host/aoa_angle/aoa_angle_config.h.*

The locator configuration can also be defined with the visual tool *AoA Analyzer*, provided in Simplicity Studio. The purpose of AoA Analyzer is twofold:

- It can create a configuration file without the need to manually edit the .json file, to ease the configurations process.
- It can apply the configuration for a single locator board and run the RTL library to estimate angles from the incoming data (without the need of running any host sample application).

To learn more about AoA Analyzer, refer to *UG514: Using the Bluetooth® Direction Finding Tool Suite*. Note that the locator configuration file created by the AoA Analyzer must be exported to be compatible with the host applications, as described in the same document.

3.5 Updating AoA Locator Configuration Through MQTT

The whole configuration file or specific configuration parameters of the AoA locator host application can be updated through MQTT on the fly without rebuilding and rerunning the application. This is done by publishing the configuration update with the specific **config** topic that the broker can use to forward the message to interested subscribers.

The AoA locator subscribes to two configuration topics, which can be used to update configuration changes during runtime. The formats of these topics are:

- `silabs/aoa/config/<locator_id>`
- `silabs/aoa/config`

The `silabs/aoa/config/<locator_id>` topic can be used when configuring just one locator, whereas the `silabs/aoa/config` topic can be used to configure more than one locator in a multi-locator setup.

3.6 Running a Single AoA Locator Host Sample Application

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

```
bt_aoa_host_locator -t <address> | -u <serial_port> [-b <baud_rate>] [-m <address>[:<port>]]
[-f <handshake>] [-c <config>]
```

Options:

`-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:

`-b` Baud rate can be given if the connection is established via serial port.

`<baud_rate>` Baud rate of the serial connection (default: 115200)

`-m` MQTT broker connection parameters.

`<address>` Address of the MQTT broker (default: localhost)

`<port>` Port of the MQTT broker (default: 1883)

`-f` Target flow control.

`<handshake>`: 0/1 (disabled/enabled). (default: 1)

`-c` Locator configuration file. It contains the locator configuration parameters.

`<config>`: Path to the configuration file. An example config file is provided in *bt_aoa_host_locator/config/locator_config.json*.

If you are using the MQTT Explorer to monitor the MQTT messages on your PC, make sure the port and host are configured as shown in the following figure:

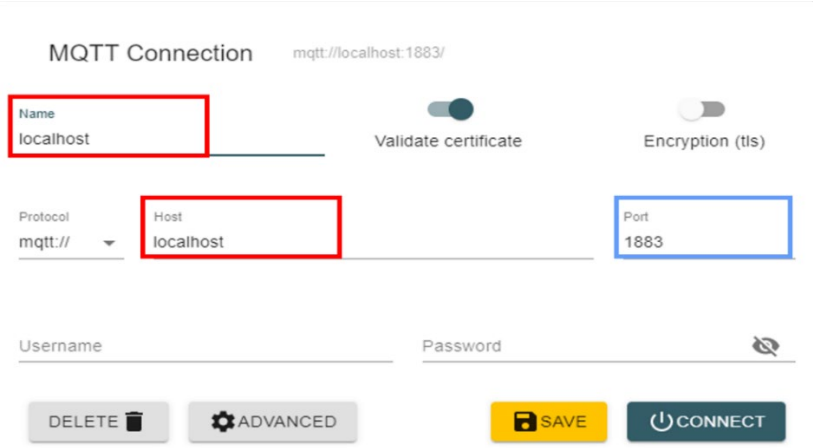


Figure 3-3. MQTT Explorer Configuration

The MQTT topic for the angle/iq data has the format `silabs/aoa/<reportMode>/<locator_id>/<tag_id>`. The `locator_id` and `tag_id` are formed as `ble-<ADDRESS_TYPE>-<BLE_ADDRESS>`, where `<ADDRESS_TYPE>` is either `sr` (for static random) or `pd` (for public device). `<BLE_ADDRESS>` is the 6-byte address without any separators, strictly using UPPERCASE letters.

Regardless of the CTE receiving mode, the AoA host application provides a common interface for the angle data structure to the MQTT broker. A prototype of the angle data looks like:

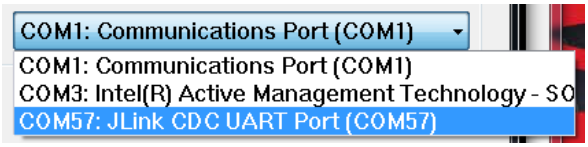
```
{
  "azimuth": 60.0,
  "azimuth_stdev": 5.0
  "elevation": 120.0,
  "elevation_stdev": 10.0
  "distance": 2.5,
  "distance_stdev": 0,
  "sequence": 123
}
```

When IQREPORT mode is enabled, the MQTT message containing the raw IQ data looks like:

```
{
  "channel": 13,
  "rssi": -50,
  "sequence": 123,
  "samples": [23, 105, 106, -10, 2, -108, ...]
}
```

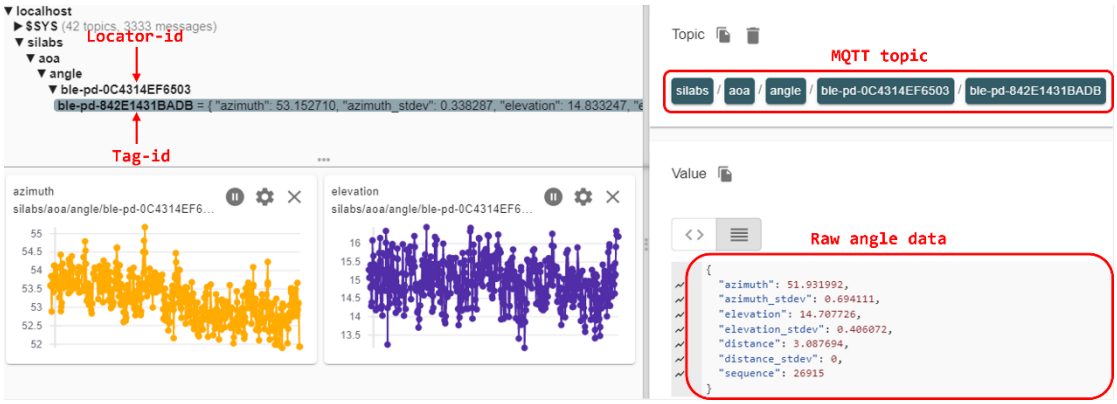
To test the AoA locator host sample application:

1. Flash a tag device (such as a Thunderboard BG22) with a bootloader and:
   - Connection-based sample app as described in 2.1 Connection-Based Asset Tag Sample Application to test CONN mode.
   - Connectionless sample app as described in 2.2 Connectionless Asset Tag Sample Application to test CONN_LESS mode.
   - Silabs sample app as described in 2.3 Silicon Labs Proprietary Asset Tag Sample Application to test SILABS mode.
2. Flash a NCP AoA locator board with the antenna array attached to a WSTK (such as BRD4191A) with a bootloader and the NCP AoA locator firmware as described in 3.1 NCP AoA Locator Sample Application. Please refer to *QSG175: Silicon Labs Direction Finding Solution Quick-Start Guide* to learn how to program BRD4191A using Simplicity Studio 5.
3. Make sure you have the correct build environment: 3.3 Building the Single AoA Locator Host Sample Application.
4. Make sure you have installed Mosquitto MQTT broker: 1.4 Prerequisites.
5. Navigate to the export/app/bluetooth/example_host/bt_aoa_host_locator/exe folder
6. Attach the WSTK to the PC and find the port number of the virtual COM port over JLink, for example by opening a terminal program that lists serial ports:



7. Start the host application from a command line with the COM port number, for example: `./bt_aoa_host_locator -u COM57`.
8. Alternatively, connect to your WSTK via Ethernet and start the application using its IP address, for example, `./bt_aoa_host_locator -t 192.168.1.2`.
9. If the application gets stuck at the beginning, push the reset button on the WSTK.
10. If the application exits at the beginning with an MQTT error, make sure that the mosquitto service is running in the background. For example, on Windows

    a) Open the Task Manager.
    b) If you see the simplified view, click **More details**.
    c) Open the Services tab.
    d) Find the mosquitto service.
    e) If it is stopped, right-click it, and click **Start**. If it is running, right-click it, and click **Restart**.
11. Open MQTT Explorer for a structured overview of the MQTT messages (angle data). Now you should see something like this:



Note that steps 7 and 8 start the application using the default settings. By default, the host application starts the locator using SILABS CTE mode. If you want to test other CTE modes, you can do so on the fly through the MQTT broker using the config topic, without recompiling or restarting the application. Alternatively, you can modify the config JSON file according to your need, and start the application using the config option like this: `./bt_aoa_host_locator -u COM57 -c ../config/locator_config.json`.

12. Now, try updating the configuration of your locator by modifying any config parameters you want to see take effect. You can do so easily using the MQTT Explorer "Publish" window. For example, when testing the connection-based CTE, change the CTE mode to CONN, simply by publishing the following MQTT message using the `silabs/aoa/config/<locator_id>` topic.

Note that the Silicon Labs proprietary approach supports scalability up to hundreds of asset tags. This approach uses the proprietary Silicon Labs CTE protocol, which improves the scalability and has low memory consumption on the AoA receiver, even with hundreds of tags. In practice this approach can support an unlimited number of tags, although a large number of tags may result in collisions on the advertising channels.

In contrast, the Connection-based and Connectionless standard solutions need more memory for the stack to keep information related to the connection status and periodic advertising syncs, respectively. Also, establishing connections or periodic advertising syncs can be time consuming, and therefore puts an absolute limit on the number of tags (about 1-50 tags on a xG22 device).

When tracking more than one tag, it is strongly recommended to disable application debug logging, as it can accumulate latencies and significantly impact the system's real-time tracking performance.

To disable the logs, open `app_log_config.h` in *<SDK Installation Location>\app\bluetooth\common_host\app_log\config* and make the following change:

```
#define APP_LOG_ENABLE      0
```

The RTL also provides IQ sample quality analysis report. The sample quality is indicated as a jitter if there is huge deviation on the antennas' signal level between snapshots. The feature is disabled by default. To enable the feature, change the log level filter to debug level in `app_log_config.h`.

```
#define APP_LOG_LEVEL_FILTER_THRESHOLD              APP_LOG_LEVEL_DEBUG
```

It is recommended to enable IQ sample quality analysis only for debugging purpose as it could have performance implications on the host machine.

In addition, when tracking more than one tag, it is important to increase the CTE advertising interval from the default value (which is 20 ms) to at least 100 ms to prevent the locator's UART from being congested, and more importantly avoid packet collisions. The advertising interval can be changed using the project configurator via *Bluetooth > GATT > Constant Tone Extension GATT Service (Silabs proprietary)* and changing the value from 32 to 160.

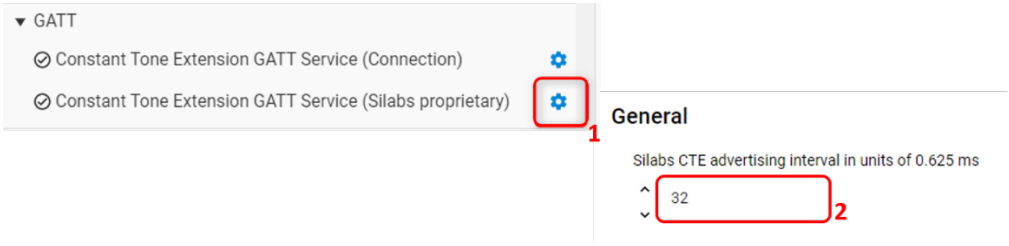


Figure 3-4. Configuring Silicon Labs CTE advertising interval

4 Positioning Sample Application

A single antenna array can give a rough estimation of the position of an asset, given that the distance can be determined from the RSSI or is constrained. However, to determine the location of an asset tag with high accuracy multiple locators are needed. Each of the locators can determine the Angle-of-Arrival of the asset, from which the position of the asset tag can be calculated using triangulation.

The Bluetooth SDK v3.x provides a unified positioning host application (***example_host/bt_host_positioning***) to demonstrate direction finding using multiple locators.

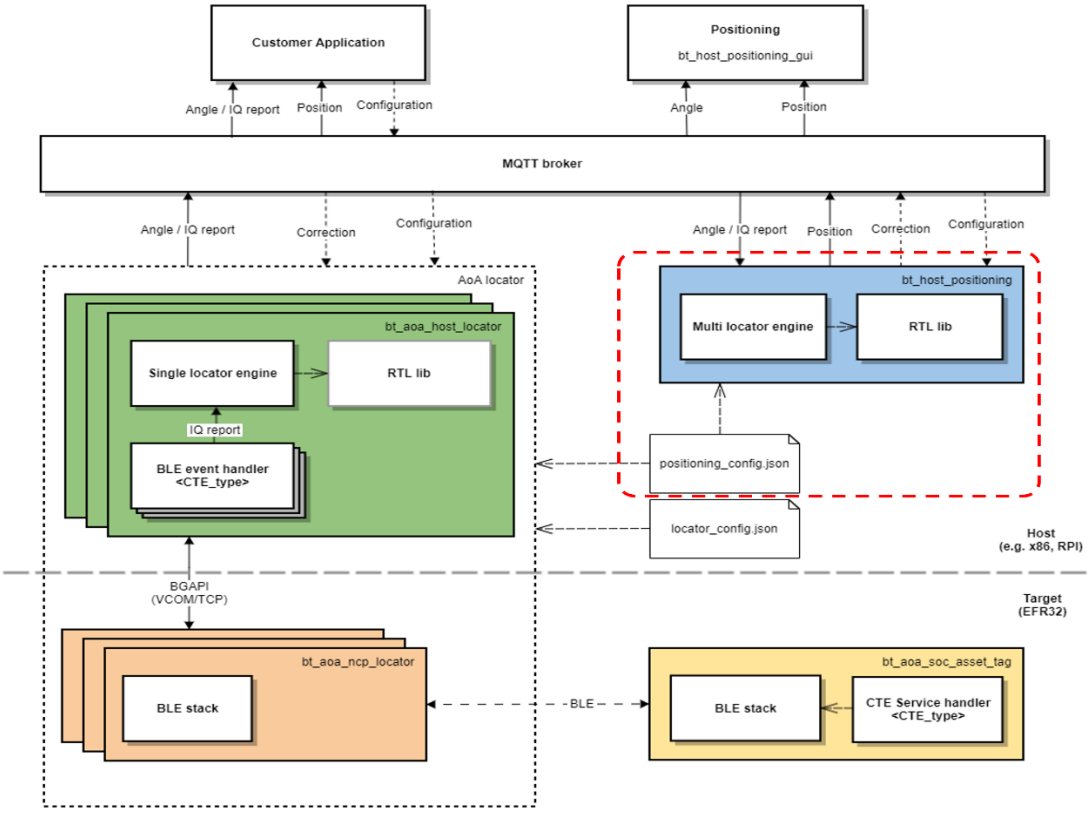


Figure 4-1. Positioning Software Architecture

The positioning host application must:

- Parse a JSON configuration file containing the IDs, orientation, and coordinates of each individual locators.
- Initialize an MQTT client and subscribe to topics created by each locator.
- Upon arrival of MQTT messages:
  - feed the IQ reports to the RTL library (optional).
  - feed the angle objects to the RTL library.
- Get the position (x, y, z) of the tag and publish it to the MQTT broker.
- Publish "correction" feedback MQTT messages about the expected angles.
- Subscribe to configuration topic.

These are all done in the `app.c` file of the ***example_host/bt_host_positioning*** project. Note that the angle data is agnostic to the type of the CTE transmission mode used. This offers great flexibility and possibilities to have different locator types in the same infrastructure. From the positioning host perspective, ideally, a tag can send CTEs for two different locator types (for instance, connection-based locator and Silicon Labs proprietary locator), and the angle data coming from both can be used to estimate the position of the tag accurately.

Like the single locator case, where the NCP AoA locator and AoA locator host can be decoupled in space, the positioning host also does not necessarily need to run on the same machine. It can basically run on any machine, a PC or Cloud (example AWS EC2). In fact, the positioning host does not need to know the IP addresses of each locator, as the communication takes place using the MQTT protocol. The MQTT publish/subscribe communication model provides great flexibility and has none of the disadvantages inherent to the client/server architecture.

However, to locate a tag, the positioning host application needs to know the IDs of each locator, their positions relative to a local coordinate system, and orientations with respect to the X, Y, and Z axes.

4.1 Positioning Configuration

The positioning configuration file has a simple JSON format that has name/value pairs of location configuration parameters, and a group of JSON objects containing name/value pairs of each individual locator's configuration parameters. A prototype of the positioning configuration file is shown below. A complete config file can be found inside ***example_host/positioning /config*** folder.

```
{
    "version": 1,
    "id":"positioning-test_room",
    "estimationModeLocation":"SL_RTL_LOC_ESTIMATION_MODE_THREE_DIM_HIGH_ACCURACY",
    "validationModeLocation":"SL_RTL_LOC_MEASUREMENT_VALIDATION_MEDIUM",
    "estimationIntervalSec":0.02,
    "locationFiltering":true,
    "locationFilteringWeight":0.1,
    "numberOfSequenceIds":6,
    "maximumSequenceIdDiffs":20,
    "locators":[
       {
          "id":"ble-pd-111111111111,
          "config": {

             Place for first locator configuration. See 3.4 AoA Locator Configuration.

          },
          "coordinate": {
             "x":0.0,
             "y":0.0,
             "z":0.0
          },
          "orientation":{
             "x":0.0,
             "y":0.0,
             "z":0.0
          }
       },
       {
          "id":"ble-pd-22222222222,
          "config": {

             Place for second locator configuration. See 3.4 AoA Locator Configuration.

      },
          "coordinate":{
             "x":0.0,
             "y":0.0,
             "z":0.0
          },
          "orientation":{
             "x":0.0,
             "y":0.0,
             "z":0.0
          }

       }
    ]
}
```

The configuration file contains the whole AoA configurations used by the positioning and single AoA locator host examples. Thus, when running the positioning example, the single AoA locator hosts can use the same config file: *app/bluetooth/example_host/bt_host_positioning/config/positioning_config.json*. In this case, each AoA locator host collects and parses the "config" JSON object that is related to it by searching for its ID.

The positioning configuration can also be defined with the visual tools *AoA Configurator* and *Positioning Tool*, provided with Simplicity Studio. AoA Configurator defines the topology of the multi-locator setup (coordinates and orientation of each locator), while the purpose of the Positioning Tool is twofold:

- It can create a top-level configuration file that defines top level settings and assigns the locators with a topology and with locator configurations.
- It can consume the top-level configuration, connect to each listed locator, and run the RTL library to estimate positions from the incoming data (without the need of running any host sample application).

To learn more about AoA Configurator and the Positioning Tool, refer to *UG514: Using the Bluetooth® Direction Finding Tool Suite*. Note that the topology file, the top level configuration file, and the locator configuration files created by the tools must be exported to be compatible with the host applications, as described in the same document.

4.1.1 Positioning ID

The positioning ID can be any string used to identify your setup, example "positioning-test_room".

4.1.2 Location Parameters

The location parameters, including estimation model, estimation interval, etc, override the default values of the corresponding configuration parameters defined in *app\bluetooth\common_host\aoa_loc\aoa_loc.c.*

To estimate the position of an asset tag accurately, the positioning engine must also know the IDs of each locator, their positions relative to a local coordinate system, and their orientations with respect to the X, Y, and Z axes as explained in the following subsections.

4.1.3 Locator IDs

The IDs of the locators are generated from their Bluetooth address and are formed as `ble-<ADDRESS_TYPE>-<BLE_ADDRESS>`, where `<ADDRESS_TYPE>` is either `sr` (for static random) or `pd` (for public device). `<BLE_ADDRESS>` is the 6-byte address without any separators, strictly using UPPERCASE letters.

The Bluetooth address of the locator can be learned in many ways. One way is to start the *bt_aoa_host_locator* sample app, which logs the Bluetooth address of the locator at the beginning.

4.1.4 Locator Coordinates

The coordinates are relative to a local coordinate system. The origin of the coordinate system and the orientation of the coordinate system are arbitrary. The only constraint is that it must be right-handed Cartesian (so that if x is pointing right, y is pointing up, as in the following image), and the unit of distance must be meter. The position of each locator board must be understood as the position of the center of

the antenna array relative to the origin. The following image provides an example how the positions of the boards must be given (each board is assumed to be at the same height, z=0):

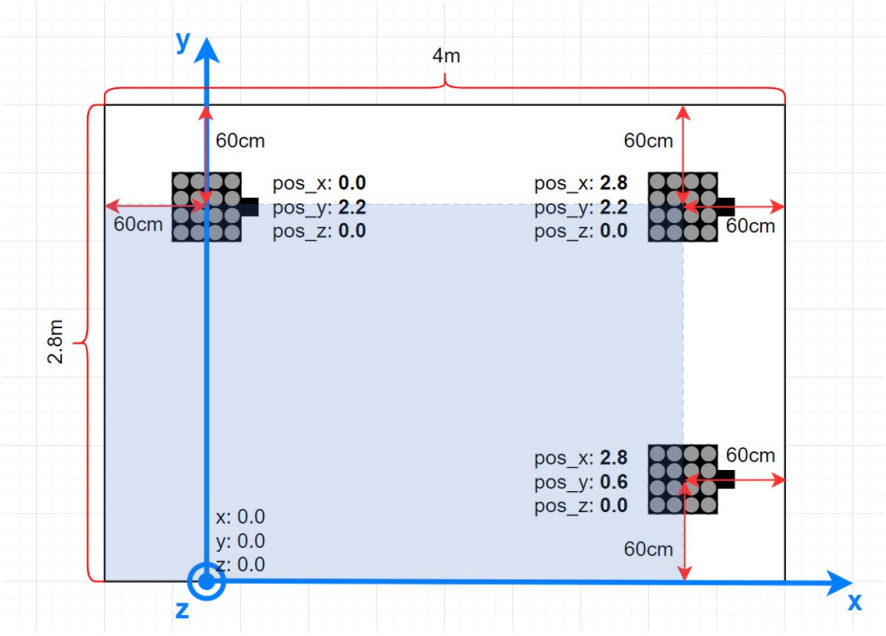


Figure 4-2. Locator Coordinate Configuration

The coordinate configurations for the above setup looks like this:

```
{
  "id": "positioning-test_room",
  "locators": [
    {
      "id": "ble-sr-111111111111",

      "coordinate": {
        "x": 0.0,
        "y": 2.2,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-222222222222",
      "coordinate": {
        "x": 2.8,
        "y": 2.2,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-333333333333",
      "coordinate": {
        "x": 2.8,
        "y": 0.6,
        "z": 0.0
      }
    }
  ]
}
```

4.1.5 Locator Orientations

After defining the coordinates of the locator boards, it is also very important to define their orientation (rotation). The calculated Angle of Arrival is always relative to the coordinate system of the board, not to the local coordinate system. Therefore, the orientation of the boards must be known, so that the RTL library can transform the angles to align with the local coordinate system.

Defining the orientation of the locators may sometimes get quite complicated, because the rotations are defined in the local coordinate system of the board, which is also rotated together with the board. Therefore, it is highly recommended to set the orientation of each board with the 3D tool *AoA Configurator*. This tool provides instant feedback about how the boards are oriented in 3D for the given rotation angles, and misbehavior caused by wrong configuration can be avoided. To learn more about the AoA Configurator tool, refer to *UG514: Using the Bluetooth® Direction Finding Tool Suite*. Note that the topology file created by the tool must be exported to be compatible with the host applications, as described in the same document.

If instead you want to define the orientations manually, information is provided in the rest of this section.

The coordinate system of the locator board looks like this:

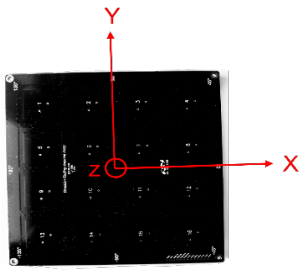


Figure 4-3. Locator Board Orientation on the X, Y, Z Axes

If the board is positioned so that the axes of this coordinate system are parallel to the axes of the local coordinate system, the rotation (orientation) does not need to be defined. In any other case, the orientation relative to this default state must be defined. (Note that in the previous image, the patch antennas are facing upward, the *z* axis is also pointing upward, and the *x* axis is pointing to the 0° direction as marked on the board.)

The orientation of the board is defined by three values: *x, y,* and *z*. Here *x* means that the board is rotated around the X axis by *x* degrees. Similarly, *y* and *z* mean that the board is rotated around the Y / Z axis by *y* / *z* degrees. Positive values mean that the board is rotated to the positive direction, that is counterclockwise, when the given axis is pointing towards you. The center of the antenna array board should stay at the same point while rotating.

The following image gives a very simple example of how the orientation should be defined. Here two boards are rotated around the z axis by 90°and -90°. (Note that patch antennas of the boards are facing upward.)

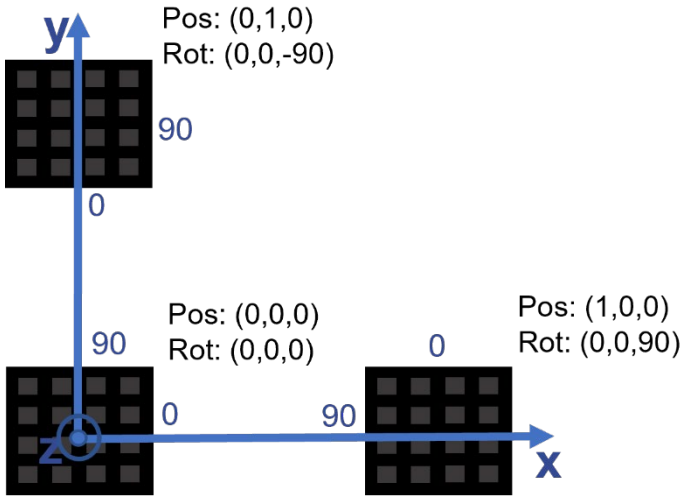


Figure 4-4. Orientation Definition Example

In this case, the configuration file should look like this:

```
{
  "id": "positioning_test_room",
  "locators": [
    {
      "id": "ble-sr-111111111111",
      "orientation": {
        "x": 0.0,
        "y": 0.0,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-222222222222",
      "orientation": {
        "x": 0.0,
        "y": 0.0,
        "z": 90.0
      }
    },
    {
      "id": "ble-pd-333333333333",
      "orientation": {
        "x": 0.0,
        "y": 0.0,
        "z": -90.0
      }
    }
  ]
}
```

It may easily happen that the board must be rotated around multiple axes to get from the default orientation to its actual orientation. In this case it is important to consider that:

- The rotations must be done in Z – Y – X order while proceeding from the default orientation to the actual orientation.
- The axes are also rotated with the board, so at the 2nd rotation the new state of axes must be considered.

The following images give some examples for this scenario. Assume the board is facing upside down, and its 0° direction is pointing in the direction of the y axis of the local coordinate system. To get this orientation from the default orientation, the board has to be rotated 90° around its z axis first, then 180° around its x axis:

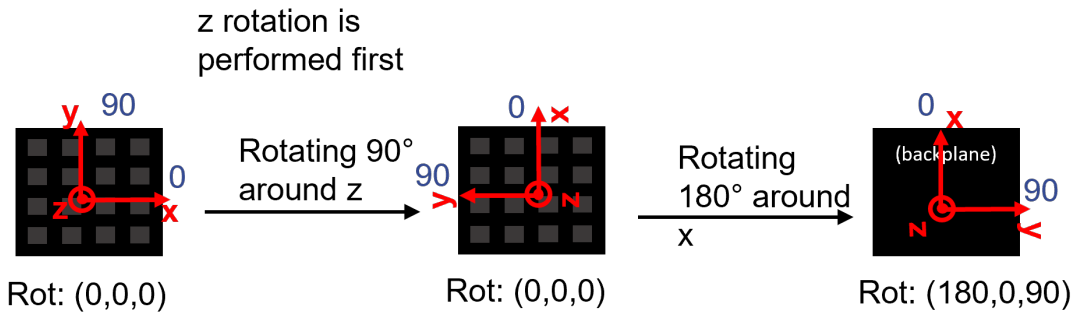


Figure 4-5. Locator Board Orientation for an Upside-Down Position— 0° Direction Pointing in the Direction of the Y Axis

Similarly, if the board is facing upside down, and its 0° direction is pointing in the opposite direction as the y axis of the local coordinate system, then the board has to be rotated -90° around its z axis first, then 180° around its x axis:

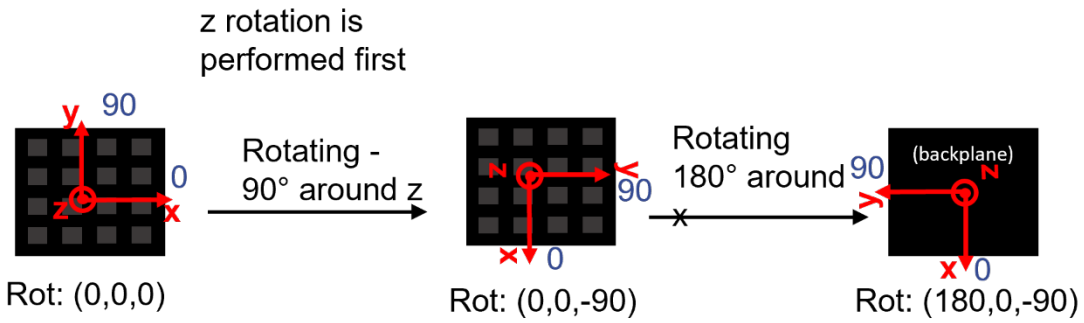


Figure 4-6. Locator Board Orientation for and Upside-Down Position— 0° Direction Pointing in the Direction Opposite to the Y Axis

The following image gives a simple example of how the orientation should be defined for a system with antenna arrays facing down:

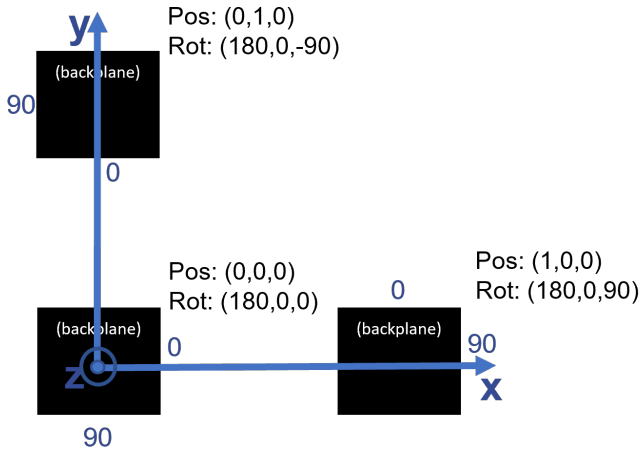


Figure 4-7. Example Showing the Values of Orientation Parameters of Three Locators Positioned Upside Down

The following two images show two recommended setups for testing purposes.

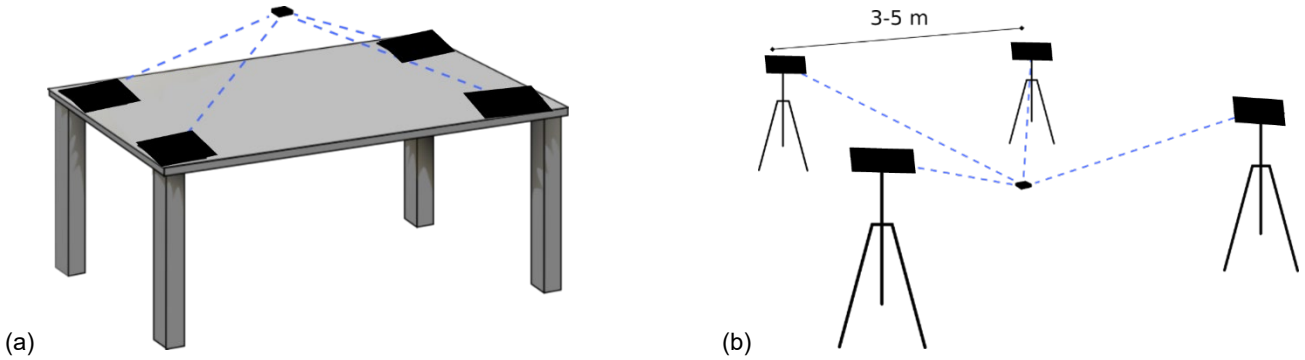


Figure 4-8. Recommended Setup of Multiple Locators

The simplest setup (a) is when four locators are placed on the four corners of a table, facing up. In this case the tag should be located above the plane of the table, since the antenna arrays "see" only upward. If the locators are rotated to the same direction, then all orientation parameters will be 0 in the config file.

A slightly more realistic use case is the second case (b), when the antennas are either fixed on the ceiling or standing on tripods, facing down. This ensures a better line of sight in any room. The recommended distance between the antennas is 3-5 m. In this case the tag

should be found under the antenna array, and since the antenna arrays are facing down, the orientation parameter for either *x* or *y* of each of them should be set to 180° in the config file.

The following configuration shows a practical setup for the above scenario. The setup consists of four locators which are in square formation at a height of 2+ meters, rotated upside-down. The locators can be, for example, mounted onto the ceiling or on stands. The evaluation setup has four locators to provide best possible positioning and ensure that effects of multipath are minimal in the position calculation, as most often at least three of the locators give correct angle estimations.

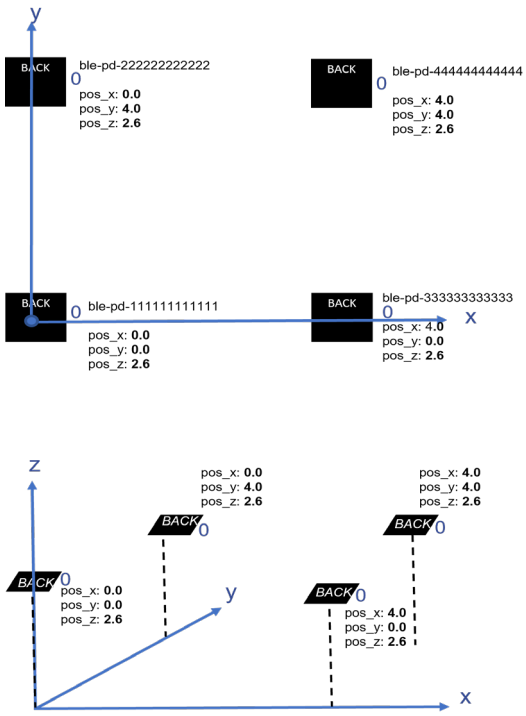


Figure 4-9. Example Setup for Evaluation of Position Calculation Using Four Locators Mounted on the Ceiling of a Room

```
{
  "id": "positioning_evaluation_setup",
  "locators": [
    {
      "id": "ble-pd-111111111111",
      "coordinate": {
        "x": 0.0,
        "y": 0.0,
        "z": 2.6
      },
      "orientation": {
        "x": 180.0,
        "y": 0.0,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-222222222222",
      "coordinate": {
        "x": 0.0,
        "y": 4.0,
        "z": 2.6
      },
      "orientation": {
```

```
        "x": 180.0,
        "y": 0.0,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-333333333333",
      "coordinate": {
        "x": 4.0,
        "y": 0.0,
        "z": 2.6
      },
      "orientation": {
        "x": 180.0,
        "y": 0.0,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-444444444444",
      "coordinate": {
        "x": 4.0,
        "y": 4.0,
        "z": 2.6
      },
      "orientation": {
        "x": 180.0,
        "y": 0.0,
        "z": 0.0
      }
    },

  ]
}
```

4.2 Updating AoA Configuration Through MQTT

The whole configuration file or specific configuration parameters of the positioning and/or AoA locator host applications can be updated through MQTT on the fly without rebuilding and rerunning the applications. This is done by publishing the configuration update with the specific **config** topic that the broker can use to forward the message to interested subscribers.

The Positioning host subscribes to two configuration topics, which can be used to update configuration changes during runtime. The formats of these topics are:

- `silabs/aoa/config/<positioning_id>`
- `silabs/aoa/config`

In summary, when running a multi-locator setup, the following configuration topics can be used to update the configuration of the locators and position hosts on the fly.

- AoA locator: `silabs/aoa/config<locator_id>`
- Positioning: `silabs/aoa/config/<positioning_id>`
- AoA locator and/or Positioning: `silabs/aoa/config`

The first two topics can be used when updating the configuration of a specific locator with a given <locator_id> ID and the positioning host identified with `<positioning_id>` ID, respectively. The third topic is a universal configuration topic which can be used for updating the configuration of one or more locators, and/or the positioning host at a time. In this case, it is possible to make all the configuration changes in one place (for example, inside the positioning_config.json file), copy the content and push that via MQTT to make all the configuration updates simultaneously.

4.3 Running the Positioning Sample App

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

```
bt_host_positioning.exe -c <config> [-m <address>[:<port>]] [-n]
```

Options:

`-c` positioning configuration file. It contains the positioning configuration parameters.

> `<config>`: Path to the configuration file. An example config file is provided in *bt_host_positioning/config/positioning_config.json.*

`-n` Turn off the bad angle feedback feature. Default is on.

The positioning host can be started with/without the configuration option. However, to calculate position of the tags, the application requires the configuration of the locators which can be passed via MQTT during runtime as explained in Section 4.2.

Regardless of the individual locators' types and their configuration, the positioning host application provides a common interface for the position data structure of the tag/s to the MQTT broker. A prototype of the position data looks like:

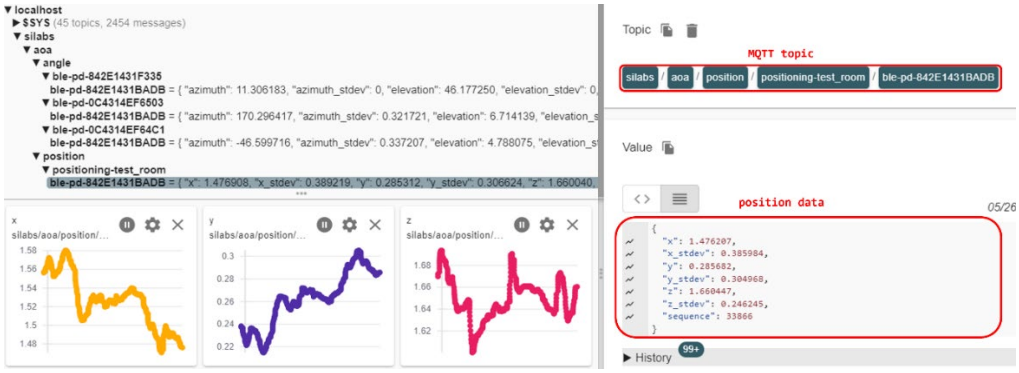
```
{
  "x": 1.509856,
  "x_stdev": 0.321395,
  "y": 0.329126,
  "y_stdev": 0.309377,
  "z": 1.650664,
  "z_stdev": 0.218991,
  "sequence": 29076
}
```

To test the positioning host application

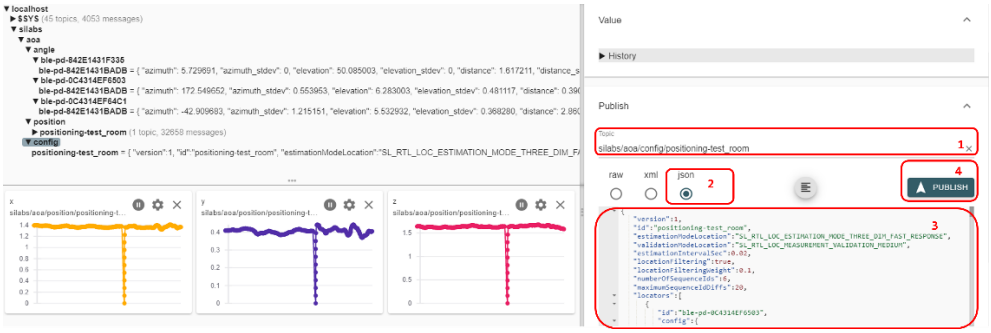
1. Flash a tag device (such as a Thunderboard BG22) with a bootloader and the asset tag sample project of your choice as described in 2 Bluetooth AoA- SoC Asset Tag.
2. Flash one or more NCP AoA locator boards with the antenna arrays attached to WSTKs (such as BRD4191A) with a bootloader and the NCP AoA locator as described in 3.1 NCP AoA Locator Sample Application.
3. Make sure you have the correct build environment: 3.3 Building the Single AoA Locator Host Sample Application.
4. Make sure you have installed Mosquitto MQTT broker: 1.4 Prerequisites.
5. Attach the WSTKs to the PC either by USB or Ethernet.
6. Open ***export/app/bluetooth/example_host/bt_aoa_host_locator/exe*** folder and start the host application for each locator with their USB COM or IP address, for example `.\bt_aoa_host_locator.exe -u COM57`. Each locator can be started with or without a config file.

   ```
   .\bt_aoa_host_locator.exe -u COM57 -c ../config/locatorX_config.json |
   $GSDK_DIR/app/bluetooth/example_host/bt_host_positioning/config/positioning_config.jso
   n
   ```

   Each locator host must run simultaneously, so you will need more command prompts if you are running all the host applications on the same PC. For more accurate results, use at least four locators in your setup.
7. Navigate to the ***/app/bluetooth/example_host/bt_host_positioning*** folder.
8. Open Mintty bash and build the project by executing `make`.
9. Find the default positioning configuration file under */app/bluetooth/example_host/bt_host_positioning/config/positioning_config.json* and modify it according to the IDs, coordinates, and orientations of your locators.
10. Start the positioning host application from a command line with the config file as parameter, for example `.\exe\bt_host_positioning.exe -c .\config\positioning_config.json`.
11. If the MQTT broker is not using the default host:port parameters, configure it with the -m option: `-m <address>:<port>`.
12. Open the MQTT Explorer to monitor the angle and position data. You should see something like the following.



13. Now, try updating the configuration of either the positioning or one of your locators by modifying any config parameters you want to see take effect. You can do so easily using the MQTT Explorer "Publish" window. For example, change the location estimation mode to SL_RTL_LOC_ESTIMATION_MODE_THREE_DIM_FAST_RESPONSE in the positioning_config.json, copy and paste the content of the file into the MQTT Explorer, and publish it using the `silabs/aoa/config/<positioning_id>` topic.



Note that each CTE packet contains a 16-bit event counter. The positioning host application uses this value to calculate the positions of an asset tag by matching IQ samples with similar event counter. In connection mode, however, since event counters for different connections are different, the positioning host will not give any results. Thus, the connection mode should not be used to test the positioning application.

By default, the positioning application runs with a feedback mechanism enabled. The feedback mechanism provides a "correction" for a deviating locator whose angle reports are far off the expected one, and thus is not pointing to the same direction where all other locators see the tag. The correction feedback provides the single locator with the information in which direction it is supposed to see the tag. This helps locators that locked up on a reflection signal which is stronger than the one coming in the line-of sight direction. The feedback mechanism works with four or more locators in a setup.

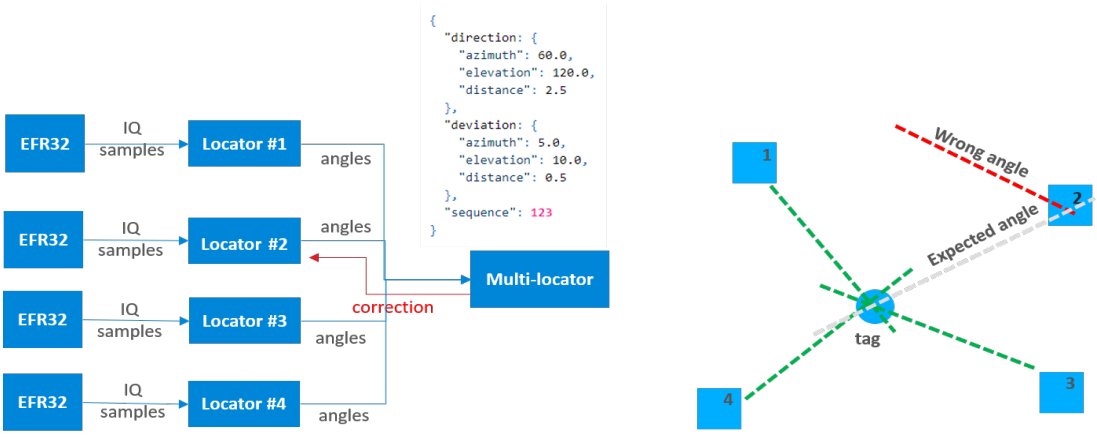


Figure 4-10. Positioning Feedback Mechanism

If you want to run the application without the feedback mechanism, start the positioning application on the command line with the "`-n`" flag:

```
.\exe\bt_host_positioning.exe -c .\config\positioning_config.json -n.
```

The positioning application does not print logs to the console. This was opted out intentionally to avoid the overhead of printf which can cause significant delay, especially if there are several tags updating angle and position information to multiple locators.

When tracking more than one tag, it is important to increase the CTE advertising interval from the default value (which is 20 ms) to at least 100 ms as shown below to prevent the locators' UART from being congested, and more importantly avoid packet collisions. In this case, the estimation interval of the positioning application in `app_config.h` should also be changed accordingly.

```
// Estimation interval in seconds.
// This value should approximate the time interval between two consecutive CTEs
#define ESTIMATION_INTERVAL_SEC      0.1f // changed from default value 0.02f
```

4.4 Visualization Script for Silicon Labs Positioning Sample Application

The Bluetooth SDK also provides a script to visualize the results of the positioning application. The visualization script is a separate Python application that collects position data by subscribing to the topics that the *positioning* sample app publishes to the MQTT broker. Therefore, it is also an example of the *user application* depicted in Figure 4-1.

The images below show how the 2D and 3D modes should look in the visualization.

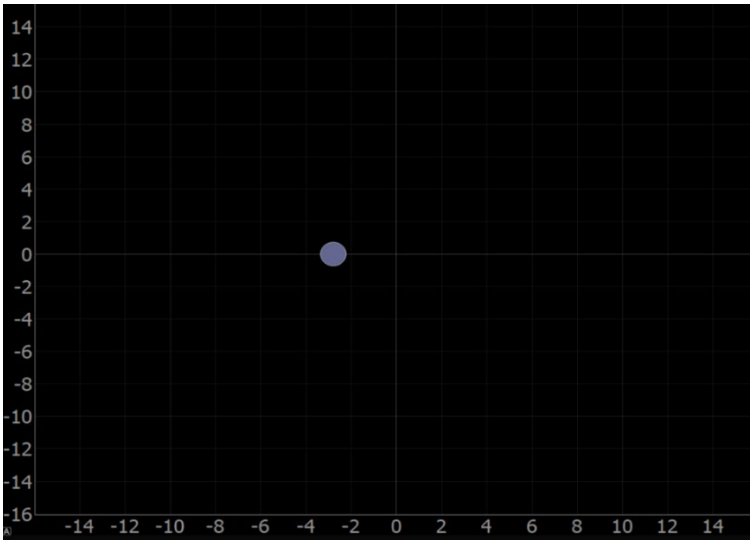


Figure 4-11. The Position of One Asset Tag Being Plotted in the 2D View

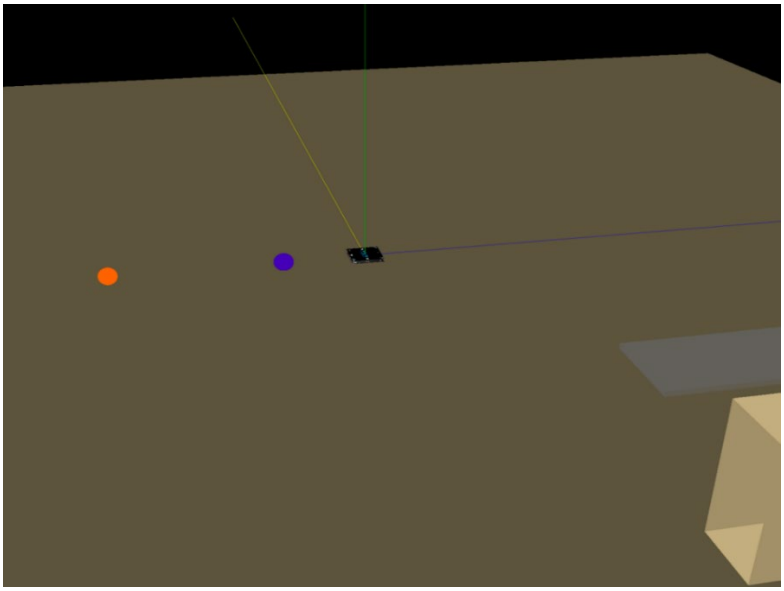


Figure 4-12. Positions of two tags and one locator shown in the 3D view

The 3D view also supports drawing simple 3D objects such as tables and shelves, as seen on the right in the above figure.

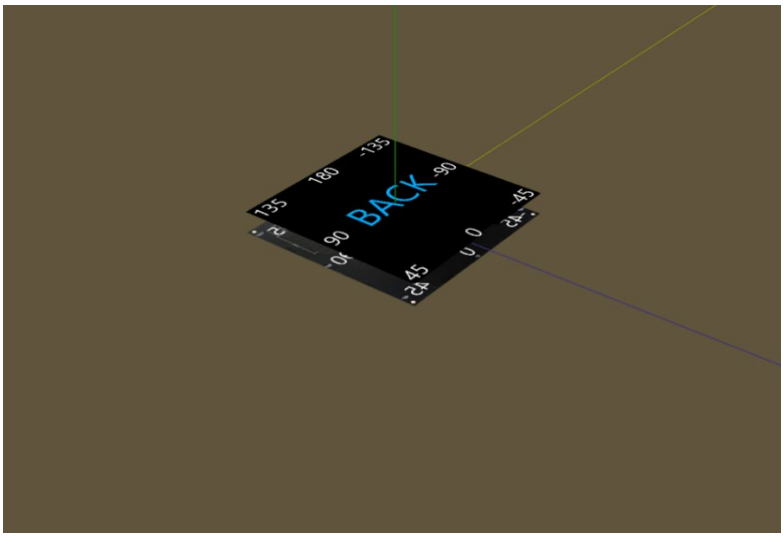


Figure 4-13. One of the Locators Shown in the 3D View

The locators are placed in the environment as given in the locator configuration file, so the visualization can be used to confirm that the configuration file matches the real-world setup. The front of the locator array (the side on which the patch antennas are located) is labeled as FRONT and the back of the locator is labeled BACK.

The visualization script can be found in the Gecko SDK Suite under:

/app/bluetooth/example_host/bt_host_positioning_gui

The following settings are configurable in the visualization script:

- **PLOT_VIEW_3D** **(0/1)**: Toggle 3D view. If disabled, uses 2D view. Note that the 3D view requires more processing and can slow down the tags' visual update rate. It is suggested to use the 2D mode for better responsiveness and the 3D mode for debugging or better but slower visualization.
- **PLOT_TAG_ADDRESSES** **(0/1)**: Toggle plotting of the tags' Bluetooth addresses next to their dots.
- **PLOT_ROOM** **(0/1)**: Toggle plotting of objects defined in `plot_room()` function in 3D mode.
- **PLOT_DEBUG_LINES** **(0/1)**: Toggle plotting visual lines from locators to tags in 3D mode. The number of lines to be drawn can be configured with the parameters MAX_NUM_TAG_LINES and MAX_NUM_LOCATOR_LINES.

- **PLOT_DEBUG_LOCATORS (0/1)**: Toggle plotting the locator arrays into the environment in 3D mode. This can be useful to ensure the locator configuration file matches with the actual setup.
- **PLOT_MARKER_TRACES (0/1)**: Toggle plotting marker traces. Marker traces are trailing dots that show the previous positions of tags. Number of trace markers to plot at once can be configured by setting self.numMarkerTraces to the desired value.

Setting up the visualization environment

1. Make sure Python 3.7 is installed in your environment. The version should be exactly 3.7, as Python 2.x or 3.8 will not work.
2. Install the following packages to your Python 3.7 installation using for example pip `py -3.7 -m pip install <package name>` or `python3.7 -m pip install <package name>`
   - pyqtgraph
   - pyqt5==5.14.0 (this exact version for Linux)
   - pyopengl
   - numpy
   - Pillow
   - paho-mqtt
3. If the previous package installations fail, try running the follow commands:
   - `python3.7 -m pip install –upgrade pip`
   - `python3.7 -m pip install –upgrade setuptools`
4. If installation of one of the previous packages cannot be done, the Python visualization may not work properly. Consider running in pipenv or docker in that case.

Running the positioning sample app with visualization

1. Follow the instructions in section 4.2 Testing the Positioning Sample App to get the positioning app up and running.
2. Ensure that the Python environment is set up correctly as per the instructions above.
3. Open a command prompt and navigate to the folder /apps/bluetooth/example_host/bt_host_positioning_gui
4. Run the visualization script with either of the following commands:

   ```
   py -3.7 app.py
   ```

   or

   ```
   python3.7 app.py
   ```

5. Note that the visualization script also needs the positioning config file to be able to display the locators and subscribe to the appropriate topics at the MQTT broker. If you do not use the default positioning_config.json file found under /app/buetooth/example_host/positioning/config, then define its location with the -c switch as for the positioning sample app:

   ```
   py -3.7 app.py -c ./../bt_host_positioning/config/my_config.json
   ```

6. Similarly, if you are not using the default MQTT host:port settings, then define them using the -m switch.
7. Now you should be able to see the GUI start, displaying the locators and tags.

Note that 3D mode may have significantly worse performance with more than three locators and more than five tags, and default intervals are used. In this case it is strongly recommended to use the 2D mode or to significantly increase the connection/advertisement intervals.

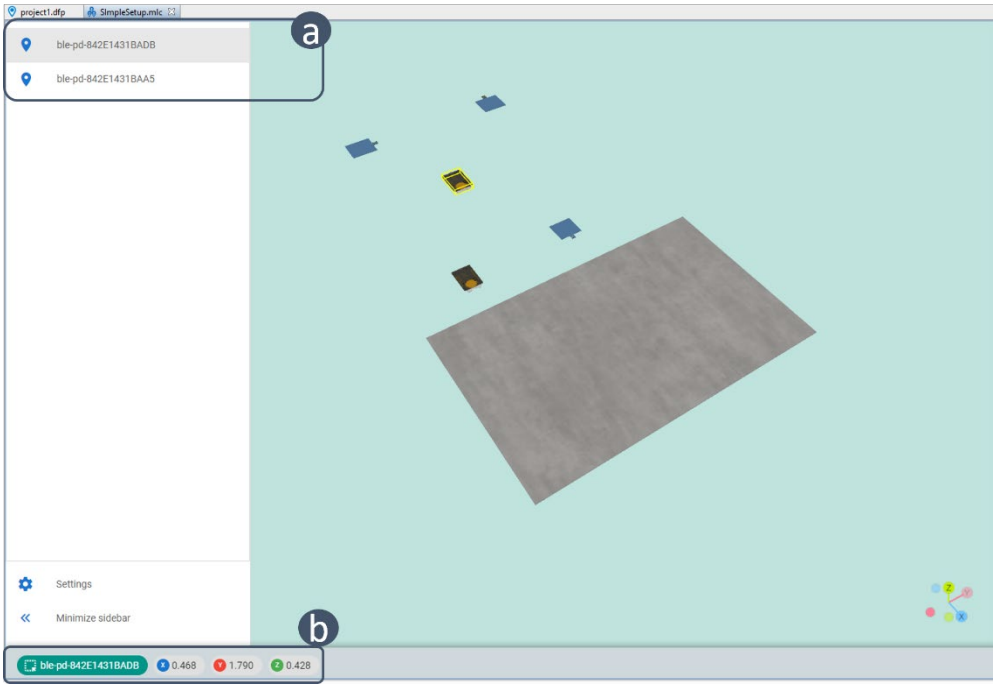
When tracking more than one tag using multiple locators, it is strongly recommended to disable the debug logs of the host applications, as logging can accumulate latencies and significantly impact the system's real-time tracking performance. This is particularly evident if the *bt_aoa_host_locator* and *bt_host_positioning* applications are running on the same machine.

To disable the logs, open `app_log_config.h` in *<SDK Installation Location>\app\bluetooth\common_host\app_log\config* and make the following change:

```
#define APP_LOG_ENABLE      0
```

4.5 Visualization Tool for Positioning

It is also possible to display the calculated 3D position of each asset tag in Simplicity Studio using the Positioning Tool. The Positioning Tool does not take the results of the positioning sample app, but instead takes the configuration files defined with the AoA tools, connects to each locator, collects IQ samples, calculates angles and finally calculates positions – all in one. The Positioning Tool can be used for evaluation of the RTL library without the need to build host applications. To learn more about the Positioning Tool, see *UG514: Using the Bluetooth® Direction Finding Tool Suite.*



A. Silicon Labs Dual Polarized Antenna Array Board (BRD4191A)

A.1 Reference Design Overview

Geometrically, the BRD4191A is a 4x4 rectangular antenna array in shape with dimensions 160 x 160 mm. The board has 16 Dual Polarized (DP) antenna patches, each consisting of one vertical and another horizontal polarized antenna elements, resulting in a total of 32 antennas (shown in gold in the figure below).

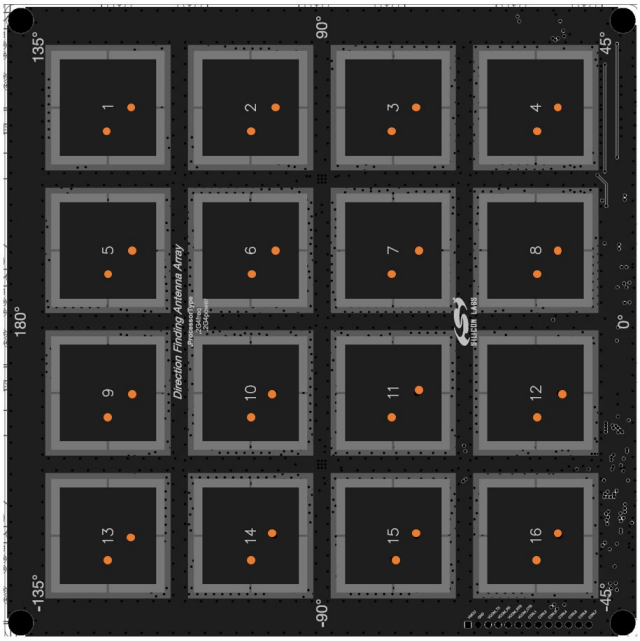


Figure: BRD4191A DP antenna array board.

In addition to receiving signals with single orientation—either horizontal or vertical polarization, the DP antenna board also allows for the reception of Circular Polarized (CP) signals on two of the antenna patches, namely antenna 6 and 13. The CP capability is important for accurate estimation of the optimal Automatic Gain Control (AGC) value at the receiver, ensuring optimal signal strength and minimal signal distortion. For this reason, one of these antennas is switched in CP mode to receive the Bluetooth packets.

The CP is realized by driving the signals from horizontal and vertical components of either antenna 6 or antenna 13 simultaneously and combining them using a hybrid coupler while creating a 90-degree phase shift between them. For more details, please refer to *AN1195: Antenna Array Design Guidelines for Direction Finding*.

A.2 Antenna RF Switch Structure

The switching of the 32 antennas is done using a chain of RF switching blocks shown below.

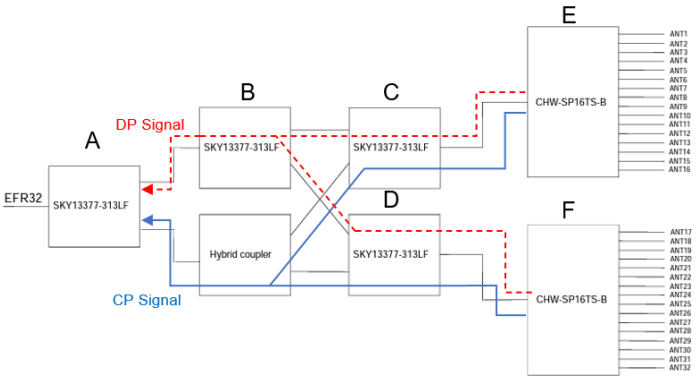


Figure 2 BRD4191A RF switch structure.

- SKY switch A is used to select between CP and DP signal sources.

- SKY switch B is used to select one of the CHW switches in DP mode (i.e E or F).
- SKY switches C & D are used to switch two antennas simultaneously in CP mode.
- CHW switches E&F are used to select one of the 16 antennas on their respective sides.

A.3 Antenna Control Pins

In the reference design hardware, there are 7 control (CTRL) pins that can be toggled to switch antennas, select between CP and DP modes, and choose the mode of operation to AoA or AoD with ease. The functionality of the CTRL pins are:

- Control pin signals CTRL1 to CTRL4 (i.e bits 0 to 3) select one of the 16 antennas on switches E & F simultaneously.
- Control pin signal CTRL5 (i.e bit 4) selects either switch E or F in DP mode.
- Control pin signal CTRL6 (i.e bit 5) selects either CP or DP mode (0 for DP & 1 for CP mode).
- Control pin signal CTRL7 (i.e bit 6) selects either AoA or AoD mode (0 for AoA & 1 for AoD mode).

A.4 Antenna Switching

The antenna switching is initiated by the Bluetooth stack when a BLE packet with *cteInfo* field present is received. A BLE packet with a CTE has the following structure.

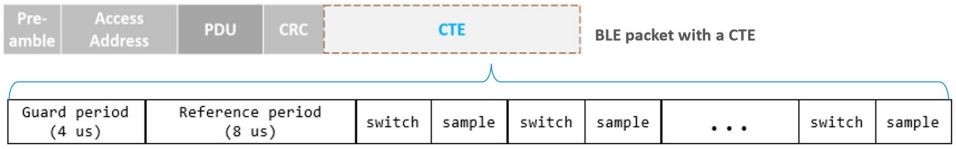


Figure 3 Bluetooth packet with CTE.

The CTE samples during the reference period are captured using one antenna, namely the reference antenna. The reference period samples are used to determine the frequency offset between the transmitter and receiver. It is important that the reference antenna is not cross polarized with the transmitter antenna. To avoid cross polarization, the reference antenna should always be selected as CP antenna. Once the phase compensation is determined, the samples in the reference period are discarded and will not be included in the angle calculation.

After the reference period, the IQ sampling is done by switching the antennas according to the switching pattern provided to the Bluetooth stack. From the estimator perspective, the RTL library expects the samples be provided so that the vertical element comes first followed by the corresponding horizontal counterpart. For example, if the switching is to be done in sequence starting from antenna ANT1 to ANT16, the samples should be fed to the algorithm in the order: {ANT_V1, ANT_H1, ANT_V2, ANT_H2, …, ANT_V16, ANT_H16}, where ANT_Vi, and ANT_Hi represent the vertical and horizontal elements of antenna patch i on the board.

The design decisions made for optimal performance and simplicity of the antenna board, such as maintaining equal lengths of the antenna feeding lines and the CP mode of operation, came with a tradeoff in the pin mapping complexity. As a result, there is no logical sequential relation between the antenna numbering and the BGAPI pin logic sequence. Figure 4 illustrates the feed lines from the RF switches E & F to the 32 antennas (16 vertical and 16 horizontal polarized antennas) with respect to their logical pin numbers. Based on Figure 4, the switching pin pattern for the default antenna switching sequence ANT1 to ANT16 becomes:

{2, 3, 0, 1, 14, 15, 12, 13, 4, 5, 8, 24, 20, 21, 10, 11, 6, 7, 22, 23, 16, 17, 19, 18, 9, 25, 26, 27, 28, 29, 31, 30}

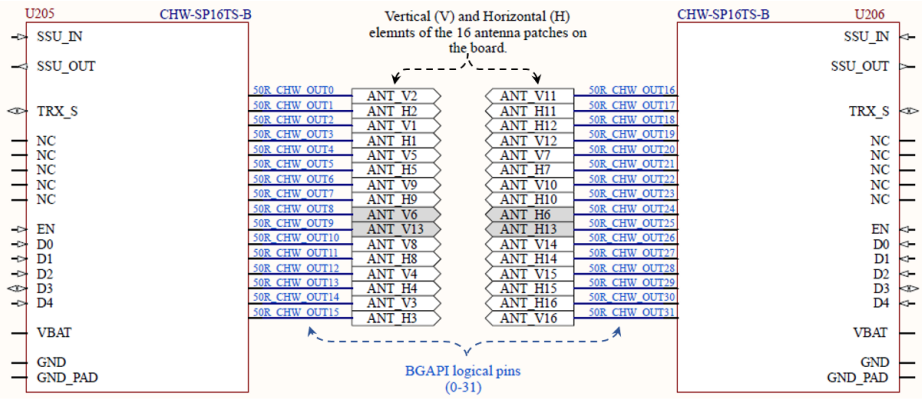


Figure 4 Pin mapping of the board with respect to the physical antenna numbers.

In summary, the relationship between the antenna switching sequence (index starting from 0) and the logical switching pattern can be represented by the following table.

pin logic (V,H):	2,3,	0,1,	14,15,	12,13,	4,5,	8,24,	20,21,	10,11,	6,7,	22,23,	16,17,	19,18,	9,25,	26,27,	28,29,	31,30
Switching Sequence:	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15

There is a helper function defined for mapping the antenna switching sequence to the switching pin control logic of this board in the RTL library: sl_rtl_aox_convert_switch_pattern(). Refer the API reference manual to learn more about its usage.

The logical pin for the CP antenna is defined based on the bitwise operation of CTRL6 (i.e bit 5) and the logical pin of either antenna 6 or 13 —which results 40 for ANT6 and 41 for ANT13.

#define CP_MODE (1 << 5)

#define ANT6_CP (CP_MODE | 8)

#define ANT13_CP (CP_MODE | 9)

Simplicity Studio

One-click access to MCU and wireless tools, documentation, software, source code libraries & more. Available for Windows, Mac and Linux!

IoT Portfolio
www.silabs.com/IoT

SW/HW
www.silabs.com/simplicity

Quality
www.silabs.com/quality

Support & Community
www.silabs.com/community

SILICON LABS

Silicon Laboratories Inc.
400 West Cesar Chavez
Austin, TX 78701
USA

www.silabs.com