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.

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
- AoA locator
- AoA multilocator
- Aoa multilocator GUI

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

The Bluetooth SDK in Gecko SDK Suite v3.x provides an AoA asset tag sample project (***Bluetooth - SoC AoA 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

The Bluetooth SDK v3.x 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:

- NCP AoA 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 compiled for the desired type of CTE receiver (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 using multiple locators, the Bluetooth SDK also provides a sample project for supporting multiple locators—the multi-locator AoA locator host application.

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

- Studio Examples
  - soc_aoa_asset_tag— ***Bluetooth – SoC AoA Asset Tag***
  - ncp_aoa_locator—***Bluetooth - NCP AoA locator***
- Host Examples (app/Bluetooth/example_host)
  - aoa_locator—*AoA locator host (single locator)*
  - aoa_multilocator—*AoA locator host (multi-locator)*

Additionally, two more host sample apps are provided for visualization purposes:

- aoa_compass (see *QSG175: Silicon Labs Direction Finding Solution Quick-Start Guide* for a detailed description)
- aoa_multilocator_gui

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.

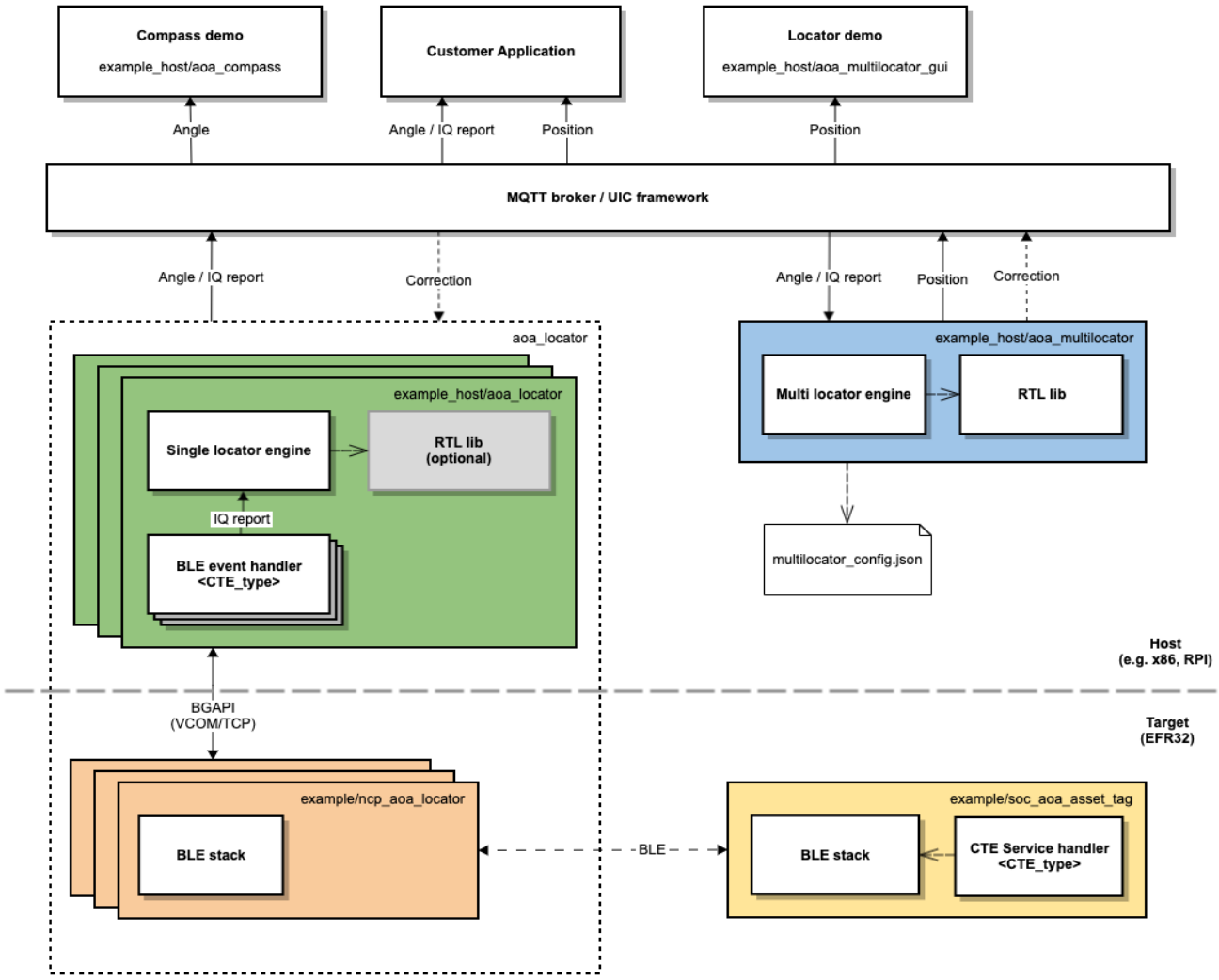


Figure 1-1. Software Architecture for the Asset-Tag, NCP AoA Locator, and Host Sample Applications

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 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. 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 box in blue represents the multi-locator 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-based device serving as a tag (for example, Thunderboard BG22).
- 4x4 antenna array board: one for single locator sample apps, more than one for multi-locator 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.

2 Bluetooth - SoC AoA 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 - SoC AoA 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.

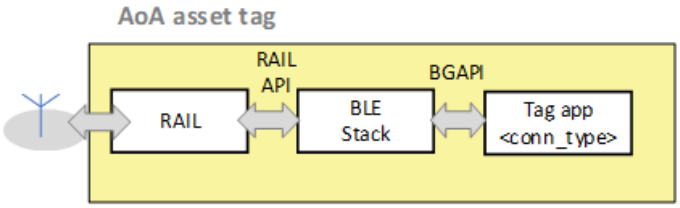


Figure 2-1. Bluetooth – SoC Asset Tag

At its core, the ***Bluetooth - SoC AoA 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 - SoC AoA 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.

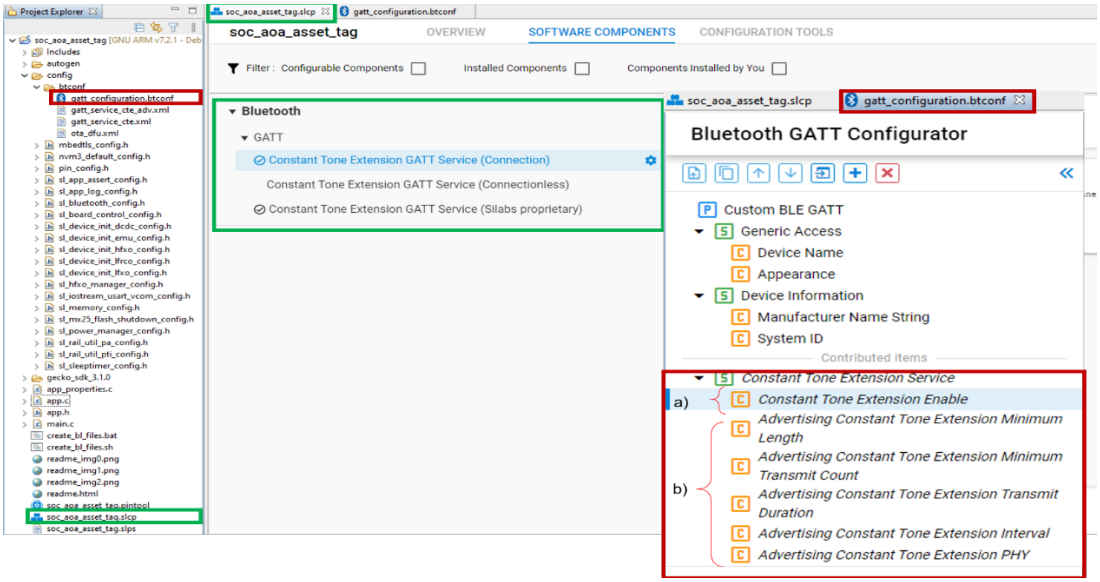


Figure 2-2. Bluetooth SoC – AoA Asset Tag Sample Project

The ***Bluetooth - SoC AoA 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.

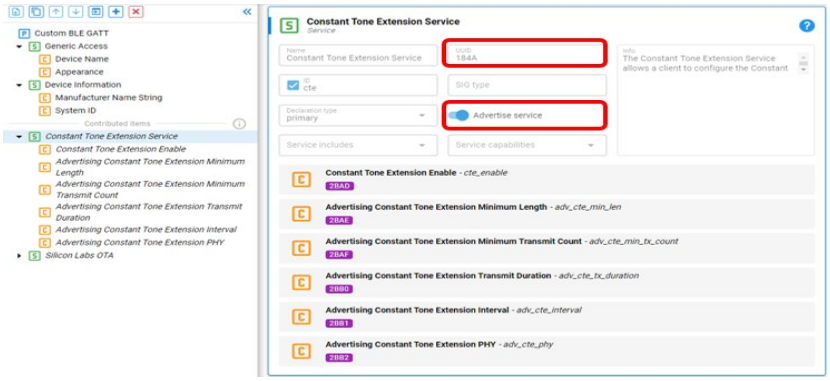


Figure 2-3. Bluetooth SoC – AoA Asset Tag CTE Service

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 - SoC AoA 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 - SoC AoA 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 - SoC AoA Asset Tag project in Simplicity Studio 5.
2. Build the project.
3. Flash it to an EFR32xG22 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. If you have not flashed any bootloader into your device, flash an xG22 bootloader sample app (for example *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.1\platform\bootloader\sample-apps\bootloader-storage-internal-single-512k\efr32mg22c224f512im40-brd4182a\ bootloader-storage-internal-single-512k.s37*).

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 - SoC AoA 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 - SoC AoA 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 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. If you have not flashed any bootloader into your device, flash an xG22 bootloader sample app (for example *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.1\platform\bootloader\sample-apps\bootloader-storage-internal-single-512k\efr32mg22c224f512im40-brd4182a\ bootloader-storage-internal-single-512k.s37*).

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 - SoC AoA 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 - SoC AoA Asset-Tag* sample app. As such, for the Silicon Labs proprietary tag, the ***Bluetooth - SoC AoA 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 - SoC AoA Asset-Tag*** project in Simplicity Studio 5.
2. Build the project.
3. Flash it to an EFR32xG22 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. If you have not flashed any bootloader into your device, flash an xG22 bootloader sample app (for example *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.1\platform\bootloader\sample-apps\bootloader-storage-internal-single-512k\efr32mg22c224f512im40-brd4182a\ bootloader-storage-internal-single-512k.s37*).

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 - NCP AoA locator***) and one sample app for a locator host. The NCP target sample project can be found in Simplicity Studio, and the host sample app can be found in the SDK folder inside app/bluetooth/*example_host/aoa_locator*. The full path is:

C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\example_host\aoa_locator

While the NCP AoA locator application is unified for all variants (connection-based, connectionless and Silicon Labs proprietary), the locator host should be compiled for each variant separately.

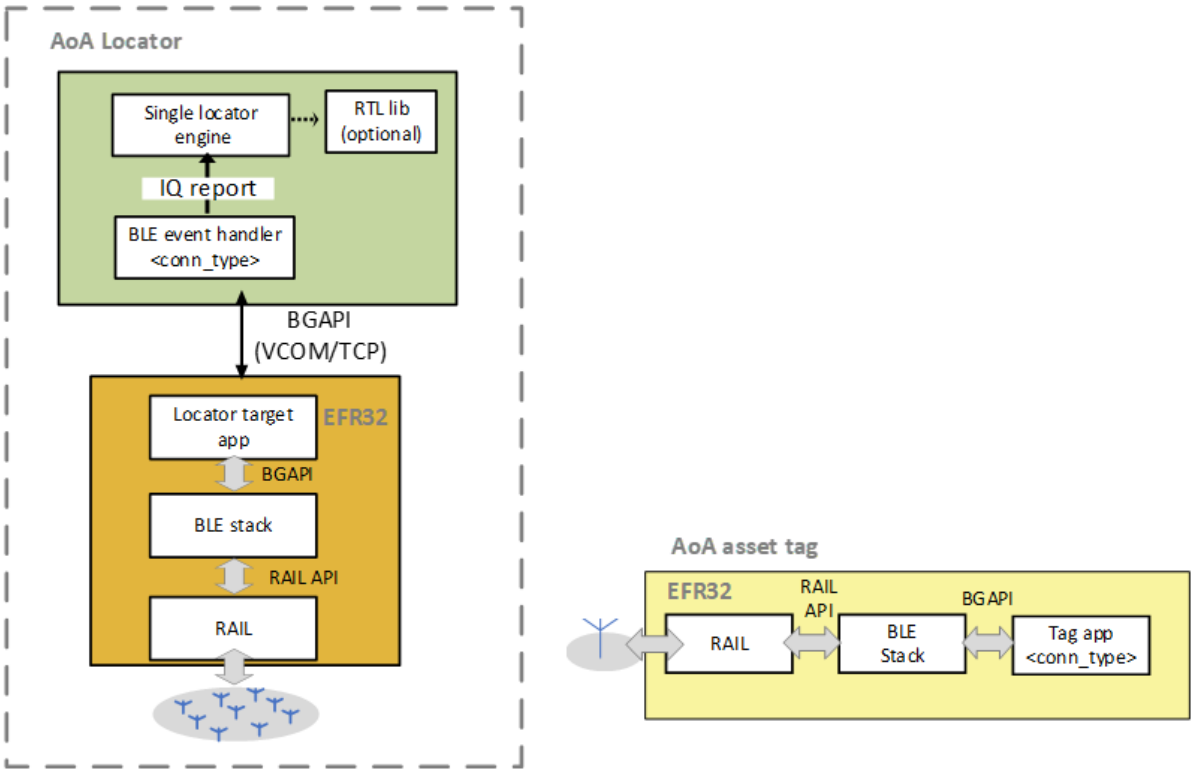


Figure 3-1. AoA Locator

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, as long as the host reaches the NCP AoA locator using its IP address.

3.1 NCP AoA Locator Sample Application

The Bluetooth SDK v3.x provides the ***Bluetooth - NCP AoA 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
- Initializing the CTE transmitter and receiver classes
- IQ sampling
- Antenna switching
- Mirroring the BGAPI interface to UART

The CTE transmitter and receiver are initialized by calling the `sl_bt_class_cte_transmitter_init()` and `sl_bt_class_cte_receiver_init()` APIs. These APIs are automatically added in the Bluetooth initialization by the installation of the *AoA Transmitter* and *AoA Receiver* components, which are default in the ***Bluetooth - NCP AoA locator*** project.

```
void sl_bt_init(void)
{
 ...
  sl_bt_class_cte_receiver_init();
  sl_bt_class_cte_transmitter_init();
 ...
}
```

Also, the ***Bluetooth - NCP AoA*** 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 - NCP AoA 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. If you have not flashed any bootloader into your device, flash an xG22 bootloader sample app (for example *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\v3.1\platform\bootloader\sample-apps\bootloader-storage-internal-single-512k\efr32mg22c224f512im40-brd4182a\ bootloader-storage-internal-single-512k.s37*).

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 the angle/IQ report data to the MQTT broker.
- Subscribing to a "correction" MQTT topic that contains its address.

The host application can be compiled in three variants: connection-based, connectionless, and Silicon Labs proprietary. Regardless of its type, 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,
  "elevation": 120.0,
  "distance": 2.5,
  "quality": 0,
  "sequence": 123
}
```

3.2.1 Building a Single AoA Locator Host Sample Application

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 ***example_host/aoa_locator*** directory.
  ```
  cd $GSDK_DIR/app/bluetooth/example_host/aoa_locator
  ```
- Build the project using the following commands.

Connection type	Command
connection-based	`make APP_MODE=conn`
connectionless	`make APP_MODE=conn_less`
Silicon Labs proprietary	`make (default)`

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 ***example_host/aoa_locator*** directory.
  ```
  cd ~$GSDK_DIR/app/bluetooth/example_host/aoa_locator
  ```
- Build the project using the following commands.

Connection type	Command
connection-based	`make APP_MODE=conn`
connectionless	`make APP_MODE=conn_less`
Silicon Labs proprietary	`make (default)`

Raspberry Pi

The RTL library is built for Raspbian 9. 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 only the export folder should be copied to the RPI, for example via an ssh connection by using scp. Follow these steps from the host computer with the Bluetooth SDK installed:

- Change to the ***example_host/aoa_locator*** directory.
  `cd $GSDK_DIR/app/bluetooth/example_host/aoa_locator`
- Export GSDK files.
  `make export`
- Copy exported GSDK files to the RPI's home folder.
  `scp -r export pi@raspberrypi.local:~`
- Start SSH connection with RPI
  `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/aoa_locator`
- Build the project using the following commands.

Connection type	Command
connection-based	`make APP_MODE=conn`
connectionless	`make APP_MODE=conn_less`
Silicon Labs proprietary	`make (default)`

The single AoA locator can be detached from the RTL library and thus does not have to calculate angle values. Instead, it can publish IQ reports directly to the MQTT broker. This can be done using the ANGLE make variable.

```
make ~$GSDK_DIR/app/bluetooth/example_host/aoa_locator  APP_MODE=<conn_type> ANGLE=0
```

A prototype of the MQTT message in this case looks like:

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

3.2.2 Running a Single AoA Locator Host Sample Application

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

```
aoa_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

`-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 azimuth mask, and asset tag allowlist.

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

The **azimuth mask** in the locator's config file tells 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 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 mask is 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.

The azimuth 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.

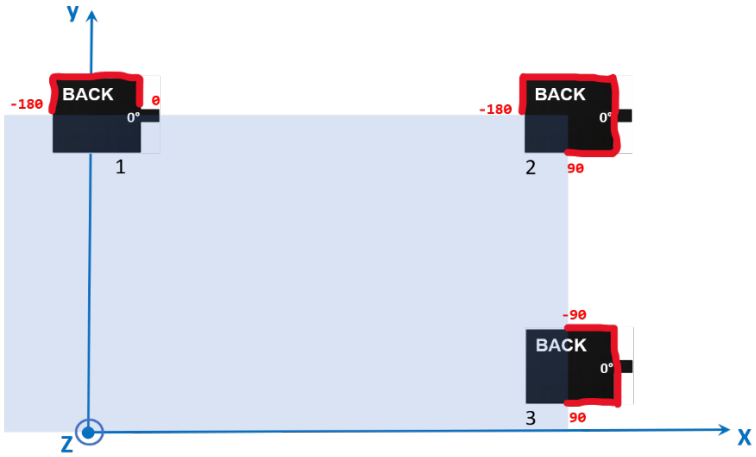


Figure 3-2. Azimuth mask configuration

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

aoa_locator/config_locato1.json

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

aoa_locator/config_locato2.json

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

aoa_locator/config_locator3.json

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

On the other hand, the **tag allowlist** in the config file 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.

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:

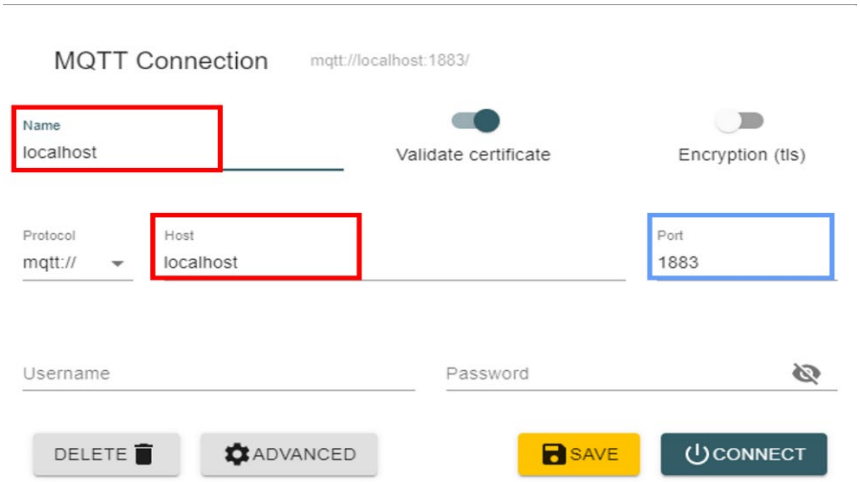


Figure 3-2. MQTT Explorer Configuration

3.2.3 Connection-Based Single Locator Host Sample Application

A connection-based locator must:

- Find the asset tag by its advertisement.
- 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.
- Sample the CTE and store the IQ sample for processing.

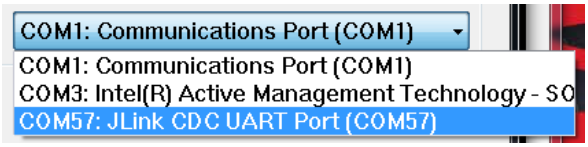
These are all done in the `app_conn.c` file of the ***example_host/aoa_locator*** sample project.

Once the IQ samples are available, the *angle* is calculated and published to the MQTT broker by calling `app_on_iq_report()`. This processing is done in `app.c` file. In the background, the `app_on_iq_report()` calls the `aoa_calculate()` API, which leverages the RTL library to calculate the angle. This is done in the `aoa.c` file.

The MQTT topic for the angle data has the format `silabs/aoa/angle/<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.

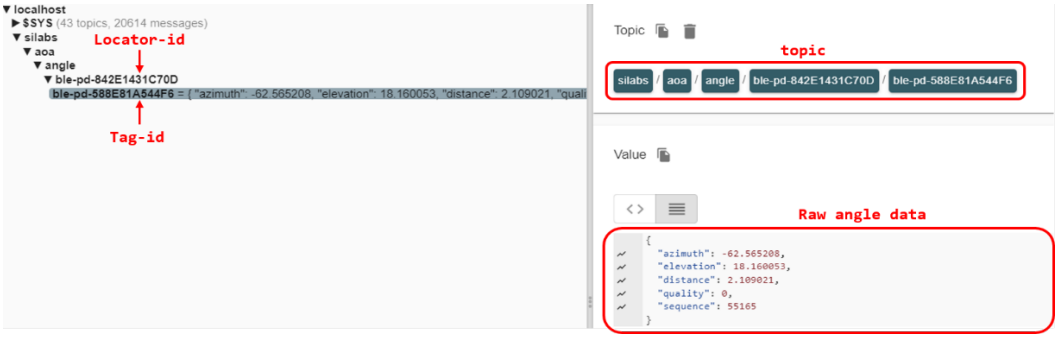
To test a connection-based single locator host sample application:

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.
2. Flash the NCP AoA locator board with the antenna array attached to a WSTK 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: 3.2.1 Building a Single AoA Locator Host Sample Application.
4. Make sure you have installed Mosquitto MQTT broker: 1.4 Prerequisites.
5. Navigate to the /app/bluetooth/example_host/aoa_locator folder
6. Open Mintty bash and build the project by executing `make APP_MODE=conn` inside the project directory.
7. Open the */exe* folder.
8. 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:



9. Start the host application from a command line with the COM port number, for example: `.\aoa_locator.exe -u COM57.`
10. Alternatively, connect to your WSTK via Ethernet and start the application using its IP address, for example `./aoa_locator.exe -t 192.168.1.2`
11. If the application gets stuck at the beginning, push the reset button on the WSTK.
12. 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.
13. Open MQTT Explorer for a structured overview of the MQTT messages (angle data). Now you should see something like this:



3.2.4 Connectionless Single Locator Host Sample Application

CTEs can also be received in Bluetooth periodic advertisements. In this case, the CTE receiver (the locator) must:

- Find the asset tag by its advertisement.
- Sync on the periodic advertisement.
- Sample the CTE and store the IQ sample for processing.

These are all done in the `app_conn_less.c` file of the ***example_host/aoa_locator*** sample project.

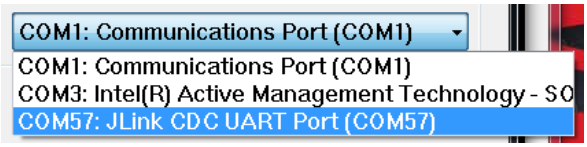
Once the IQ samples are available, the *angle* is calculated and published to the MQTT broker by calling `app_on_iq_report()`. This processing is done in `app.c` file. In the background, the `app_on_iq_report()` calls `aoa_calculate()` API, which leverages the RTL library to calculate the angle. This is done in `aoa.c` file.

The MQTT topic for the angle data has the format `silabs/aoa/angle/<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.

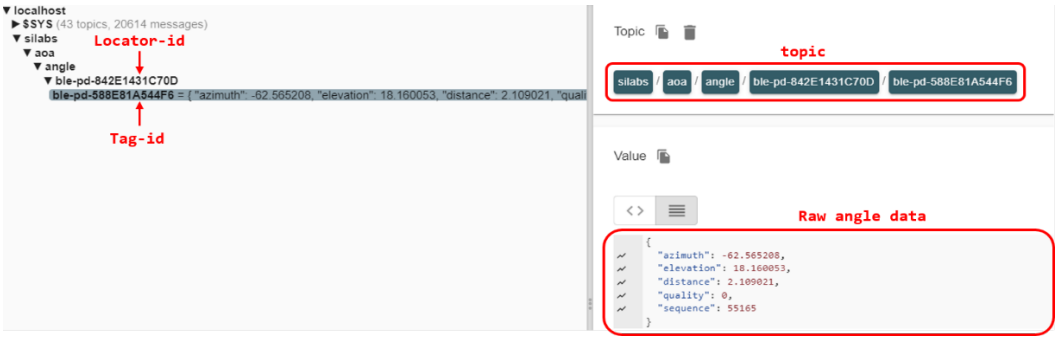
To test a connectionless single locator host application:

1. Flash a tag device (such as Thunderboard BG22) with a bootloader and connectionless sample app as described in 2.2 Connection-less Asset Tag Sample Application.
2. Flash the NCP AoA locator board with the antenna array attached to a WSTK 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: 3.2.1 Building a Single AoA Locator Host Sample Application.
4. Make sure you have installed Mosquitto MQTT broker: 1.4 Prerequisites.
5. Navigate to the /app/bluetooth/example_host/aoa_locator folder.
6. Open Mintty bash and build the project by executing `make APP_MODE=conn_less` inside the project directory.

7. Open the */exe* folder.
8. Attach your 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:



9. Start the host application from a command line with the COM port number, for example: `.\aoa_locator.exe -u COM57.`
10. Alternatively, connect to the WSTK via Ethernet and start the application using its IP address, for example:
    `./aoa_locator.exe -t 192.168.1.2`
11. If the application gets stuck at the beginning, push the reset button on the WSTK.
12. 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.
13. Open MQTT Explorer for a structured overview of the MQTT messages (angle data). Now you should see something like this:



3.2.5 Silicon Labs Proprietary Single Locator Host Sample Application

CTEs can also be received in extended advertising. In this approach, the receiver (that is, the locator) must:

- Find the asset tag by its advertisement.
- Sample CTE and store the IQ sample for processing.

These are all done in the `app_silabs.c` file of the ***example_host/aoa_locator*** sample project.

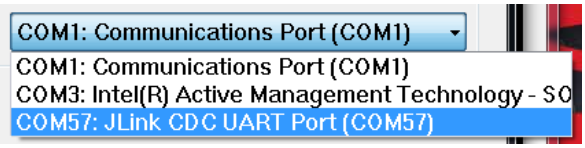
Once the IQ samples are available, the *angle* is calculated and published to the MQTT broker by calling `app_on_iq_report()`. This processing is done in `app.c` file. In the background, the `app_on_iq_report()` calls `aoa_calculate()` API, which leverages the RTL library to calculate the angle. This is done in `aoa.c` file.

The MQTT topic for the angle data has the format `silabs/aoa/angle/<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.

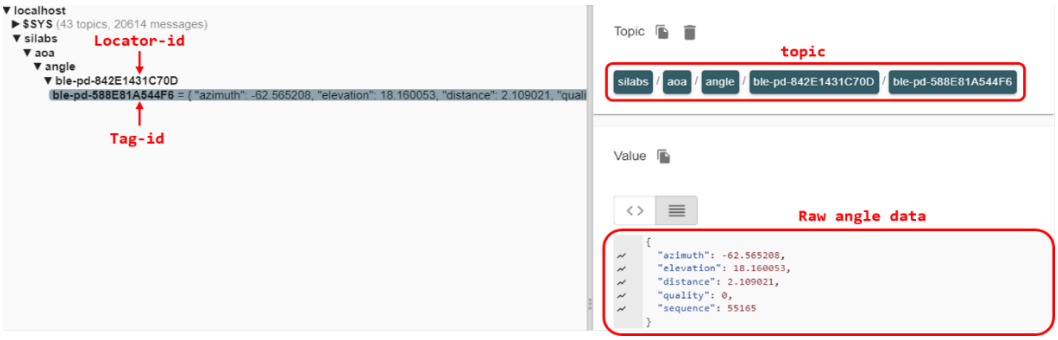
To test Silicon Labs proprietary single locator host application:

1. Flash a tag device (such as a Thunderboard BG22) with a bootloader and connectionless sample app as described in 2.3 Silicon Labs Proprietary Asset Tag Sample Application.
2. Flash the NCP AoA locator board with the antenna array attached to a WSTK 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: 3.2.1 Building a Single AoA Locator Host Sample Application.
4. Make sure you have installed Mosquitto MQTT broker: 1.4 Prerequisites.

5. Navigate to the /app/bluetooth/example_host/aoa_locator folder.
6. Open Mintty bash and build the project by executing `make` inside the project directory.
7. Open the */exe* folder.
8. Attach your 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:



9. Start the host application from a command line with the COM port number, for example: `.\aoa_locator.exe -u COM57`
10. Alternatively, connect to the WSTK via Ethernet and start the application using its IP address, for example:
    `./aoa_locator.exe -t 192.168.1.2`
11. If the application gets stuck at the beginning, push the reset button on the WSTK.
12. 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.
13. Open MQTT Explorer for a structured overview of the MQTT messages (angle data). Now you should see something like this:



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 *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\common_host\app_log\config* and make the following change:

```
#define APP_LOG_ENABLE      0
```

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.

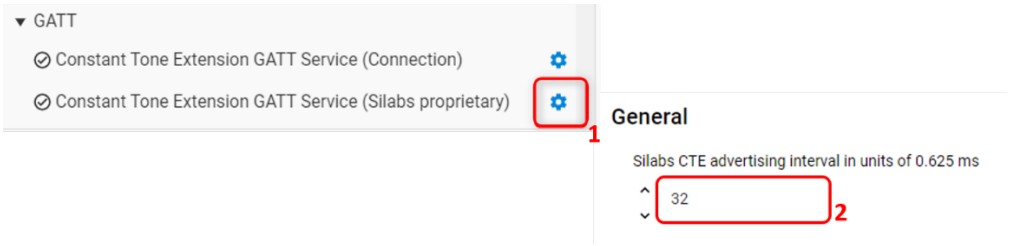


Figure 3-2. Configuring Silabs CTE advertising interval

4 Multi-Locator 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 host application (***example_host/aoa_multilocator***) to demonstrate direction finding using multi-locators.

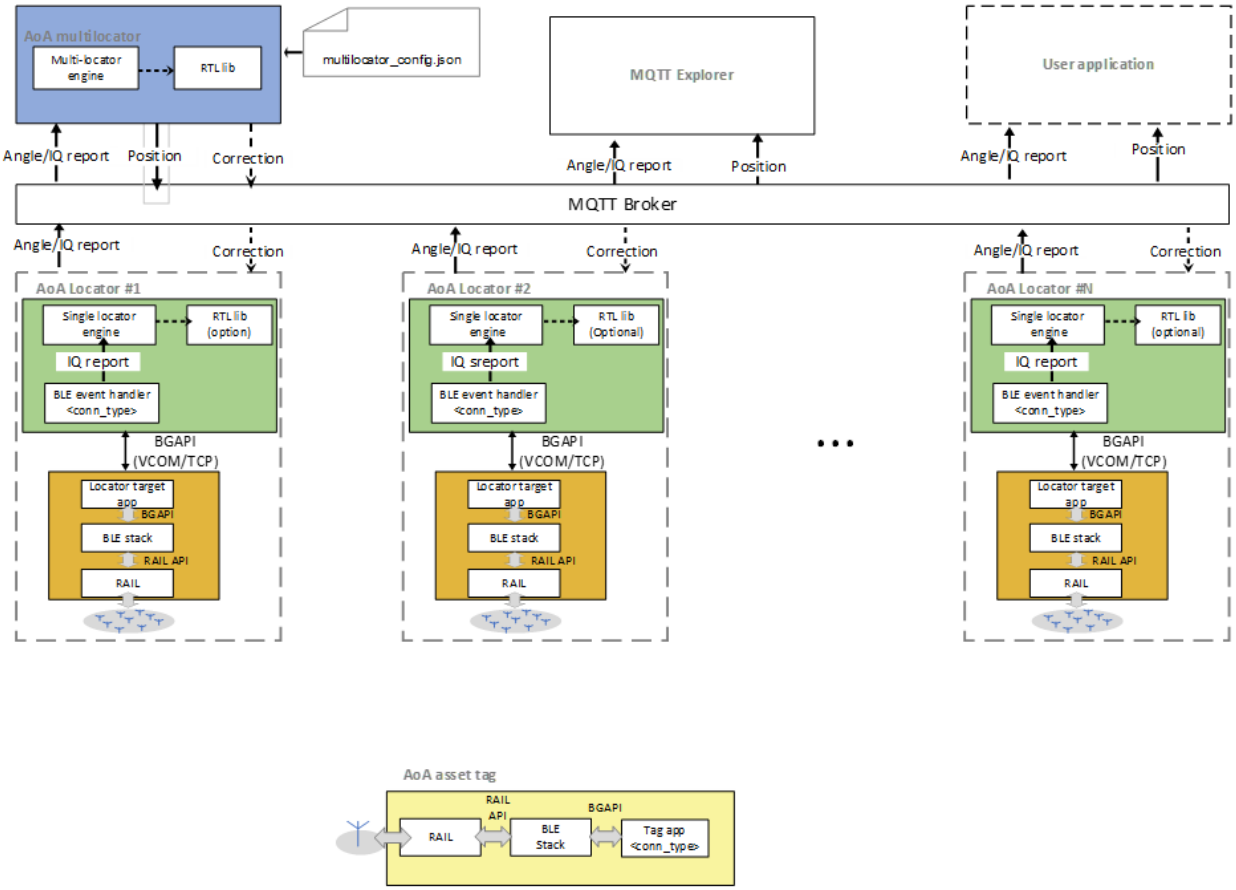


Figure 4-1. Multi-Locator AoA Architecture

The multi-locator host application must:

- Parse a JSON configuration file containing the 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 so that deviating locators can take into account the feed-back and apply the corrections.

These are all done in the `app.c` file of the ***example_host/aoa_multilocator*** project. Note that the angle data is agnostic to the type of the CTE transmission mode used by the locators (that is, connection-based, connectionless, or Silicon Labs proprietary). This offers great flexibility and possibilities to have different locator types in the same infrastructure. From the multi-locator 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 multi-locator host also does not necessarily need to run on the same machine. It can basically run on any machine (a PC or Cloud). In fact, the multi-locator 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 multi-locator host application needs to know the ID of each locator, their position relative to a local coordinate system, and their orientation with respect to the X, Y, and Z axis. This information is provided to the multi-locator host using a JSON config file during runtime. Using the locators' ID, the multi-locator host determines the topics that it will subscribe to at the MQTT broker.

4.1 Locator Configuration

The multi-locator configuration file has a simple JSON format that has a name/value pair of the locators' IDs, coordinates, and orientations.

The locator's ID is generated from its Bluetooth address, as described in section 3.2.2 Connection-Based Single Locator Host Sample Application. The Bluetooth address of the locator can be learned in many ways. One way is to start the *aoa_locator* sample app, which logs the Bluetooth address of the locator at the beginning.

In addition to the locators' ID, the multi-locator application must know the positions (coordinates), and orientation of each locator on the local coordinate system. Moreover, azimuth angle mask can also be provided via each locator's configuration file to improve both reliability and computation time, as described later in this section.

4.1.1 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 the same height, z=0):

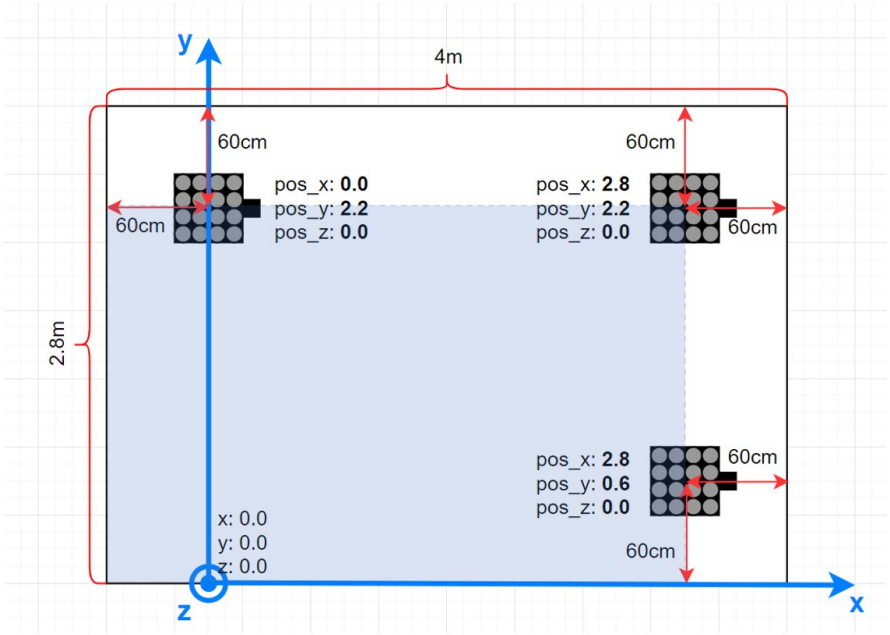


Figure 4-2. Locator Coordinate Configuration

The configuration file for the above setup looks like this:

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

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

4.1.2 Locator Orientations

After defining the position (that is, coordinates) of the locator boards, it is very important to define their orientation (rotation), as well. 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. The coordinate system of the locator board looks like this:

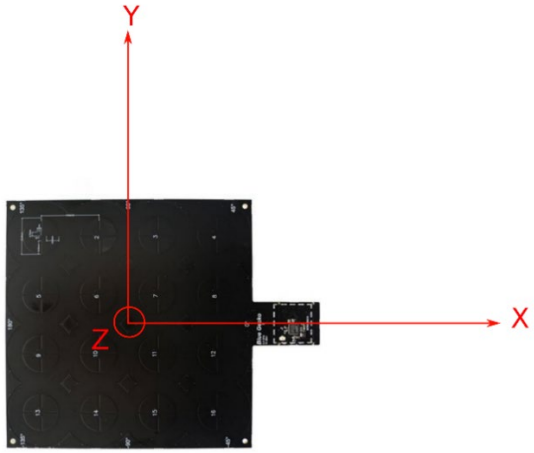


Figure 4-3. Locator Board Orientation on the X, Y, Z Axes

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.)

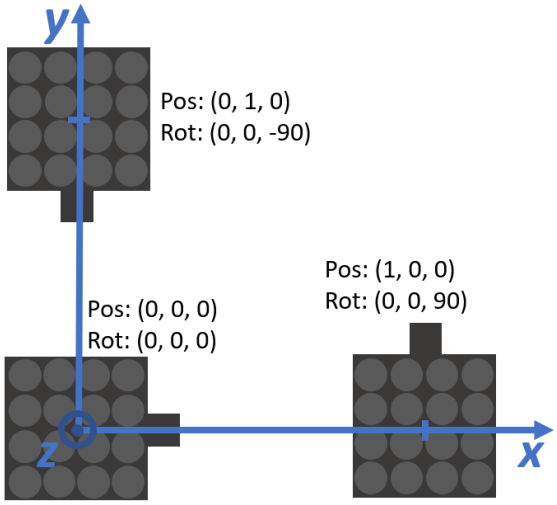


Figure 4-4. Orientation Definition Example

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

```
{
  "id": "multilocator_test_room",
  "locators": [
    {
      "id": "ble-sr-111111111111",
      "coordinate": {
```

```
        "x": 0.0,
        "y": 0.0,
        "z": 0.0
      },
      "orientation": {
        "x": 0.0,
        "y": 0.0,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-222222222222",
      "coordinate": {
        "x": 1.0,
        "y": 0.0,
        "z": 0.0
      },
      "orientation": {
        "x": 0.0,
        "y": 0.0,
        "z": 90.0
      }
    },
    {
      "id": "ble-pd-333333333333",
      "coordinate": {
        "x": 0.0,
        "y": 1.0,
        "z": 0.0
      },
      "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:

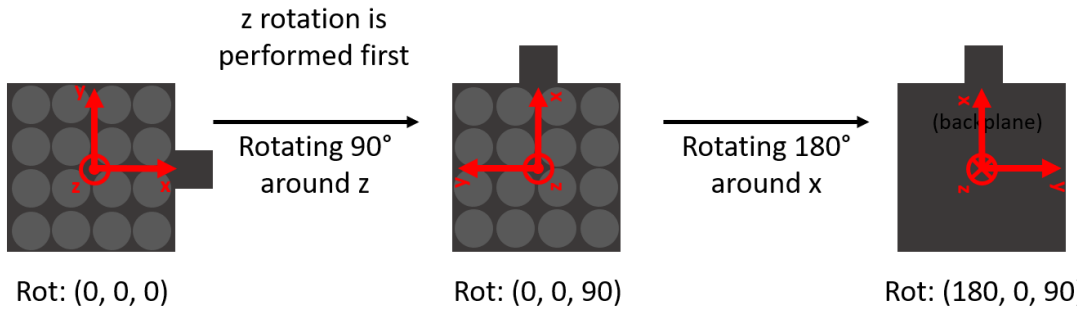


Figure 4-5. Locator Board Orientation for an Upside-Down Position— 0° Direction Pointing in the Direction of the Y Axis

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:

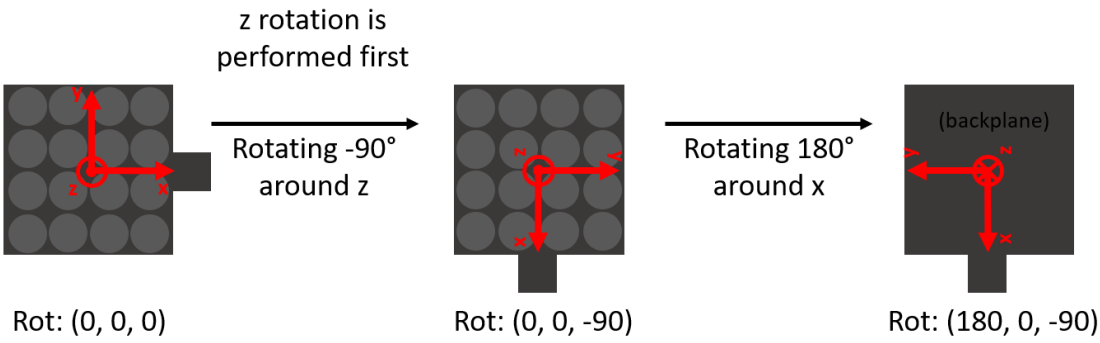


Figure 4-6. Locator Board Orientation for and Upside-Down Position— 0° Direction Pointing in the Direction Opposite to the Y Axis

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

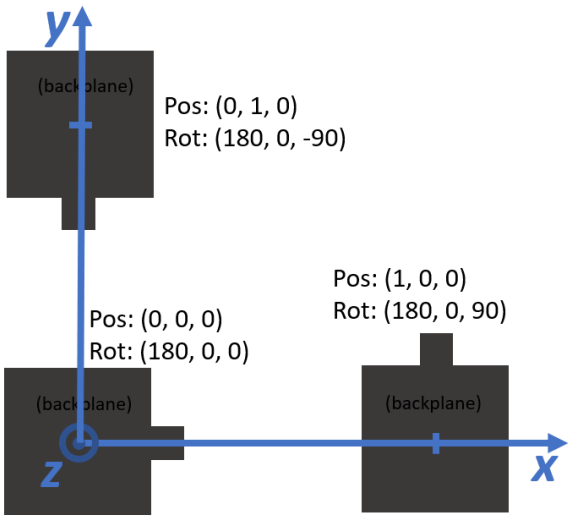


Figure 4-7. Example Showing the Values of Orientation Parameters of Three Locators Positioned Upside Down

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

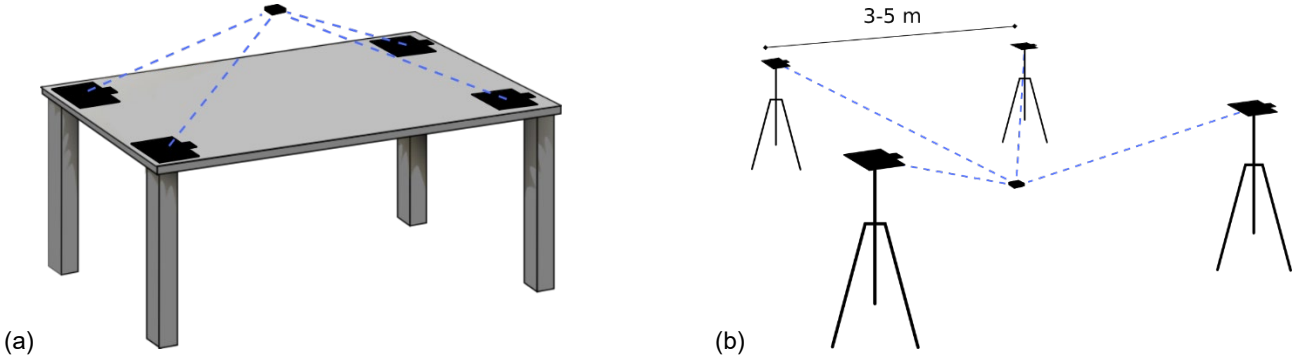


Figure 4-8. Recommended Setup of Multi-Locators

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 in order 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.

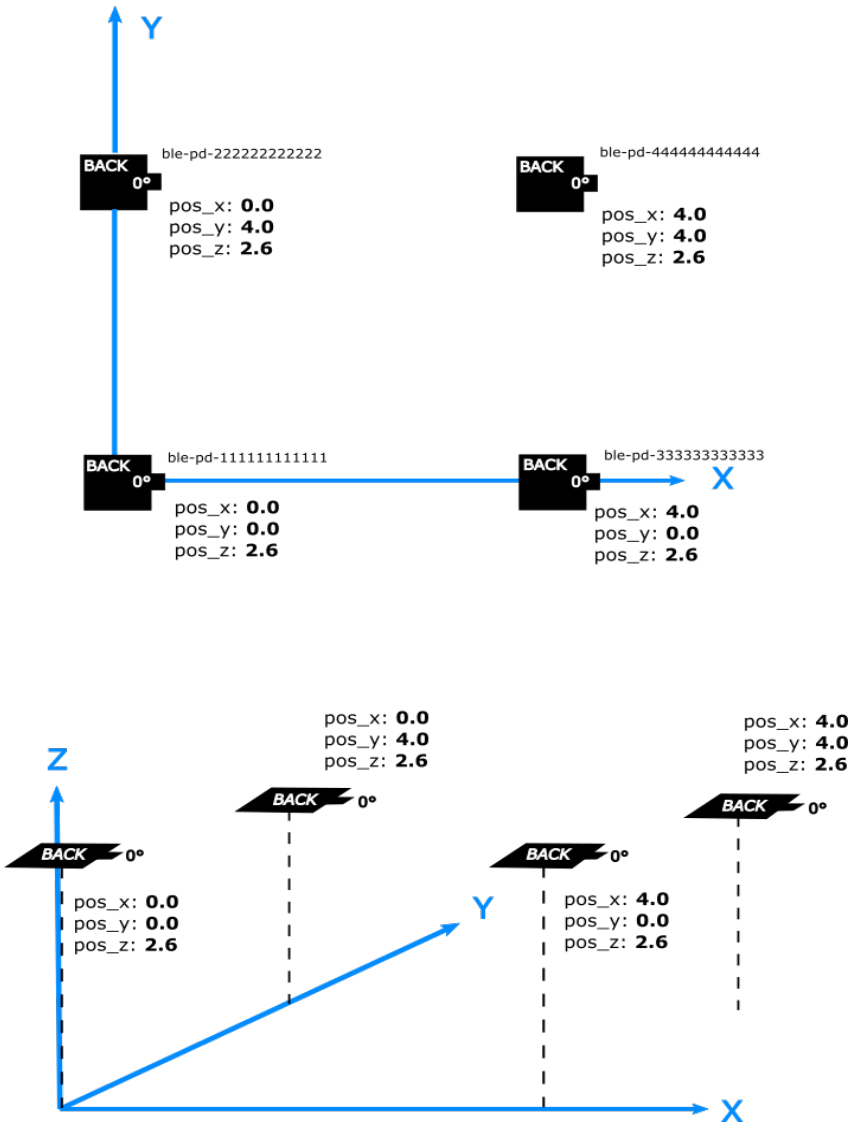


Figure 4-9. Example Setup for Evaluation of Position Calculation Using Four Locators Mounted on the Ceiling of a Room

```
{
  "id": "multilocator_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
      }
    },
```

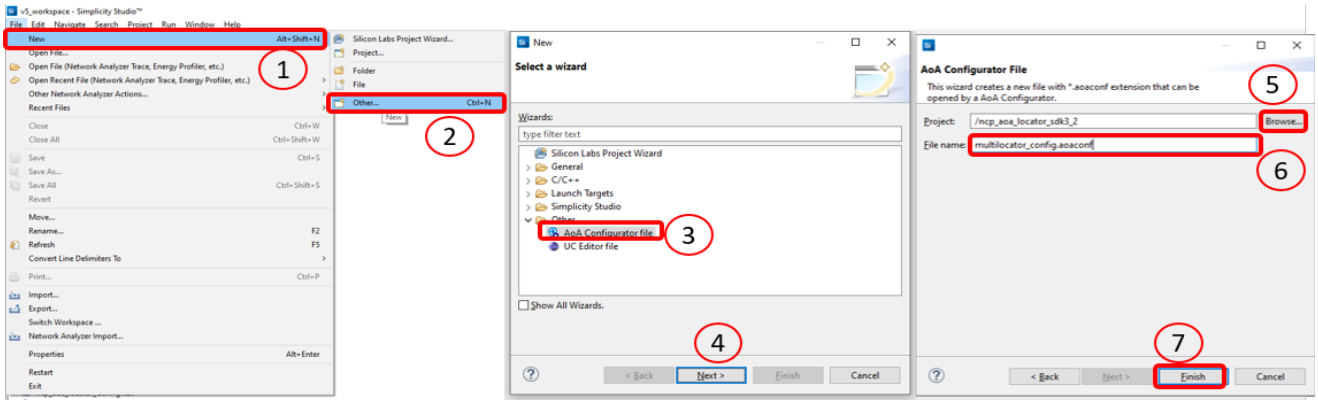
```
    {
      "id": "ble-pd-222222222222",
      "coordinate": {
        "x": 0.0,
        "y": 4.0,
        "z": 2.6
      },
      "orientation": {
        "x": 180.0,
        "y": 0.0,
        "z": 0.0
      }
    },
    {
      "id": "ble-pd-333333333333",
      "coordinate": {
        "x": 4.0,
        "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 AoA Configurator Tool

It can, at times, be a daunting task to determine the orientations of locator boards, especially when the boards are rotated over multiple axes to suit the set-up environment. To simplify this process, Bluetooth SDK v3.2 has introduced the AoA Configurator Tool. Using this tool, you can add as many locators as you have, drag them to the position you want, and even rotate them to the best orientation that is suitable for your setup. The output of the tool can be saved to a JSON file which can then be used by the multi-locator application.

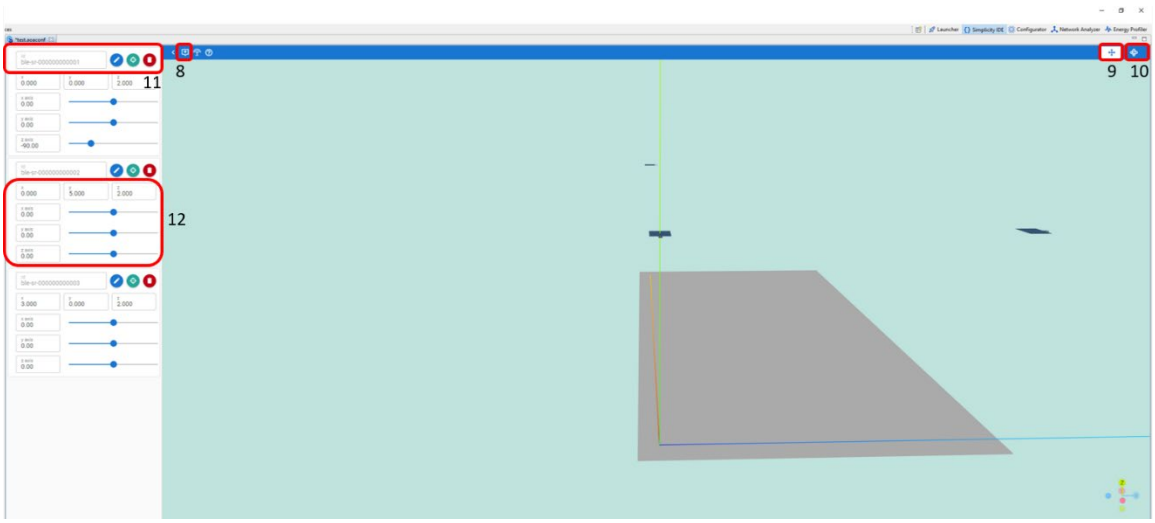
To generate a multi-locator configuration file using the AoA Configurator Tool

1. Create a project using SSv5 (for instances, *Bluetooth - NCP AoA locator*).
2. Go to **File > New > Other** (steps 1 and 2 in the figure below).
3. When the new wizard opens, select **AoA Configurator file** and click **Next** (step 3 and 4).
4. Browse to the project created above (step5). Note that the project needs to be writable and should be opened in SSv5.
5. Enter a filename (it should have the *.aoaconf* extension) and click **Finish** (step 6 and 7).



After the AoA configuration file is created SSv5 opens it in a new window automatically. In this window, you can:

6. Add a new locator button 8.
7. Translate the axes of a locator using button 9.
8. Rotate a locator using button 10.
9. Select a locator, edit a locator's ID, delete a locator using buttons in 11.
10. Manually edit a locator's coordinates, and orientations using button in 12.



11. Save and close the file.
12. Open the file in a text editor, and copy its content to your multilocator_config.json file. Alternatively, you can copy the *.aoaconf file to your /aoa_multilocator/config director and rename it with *.json extension.

4.3 Testing the Multi-Locator Sample App

To test the multi-locator host application

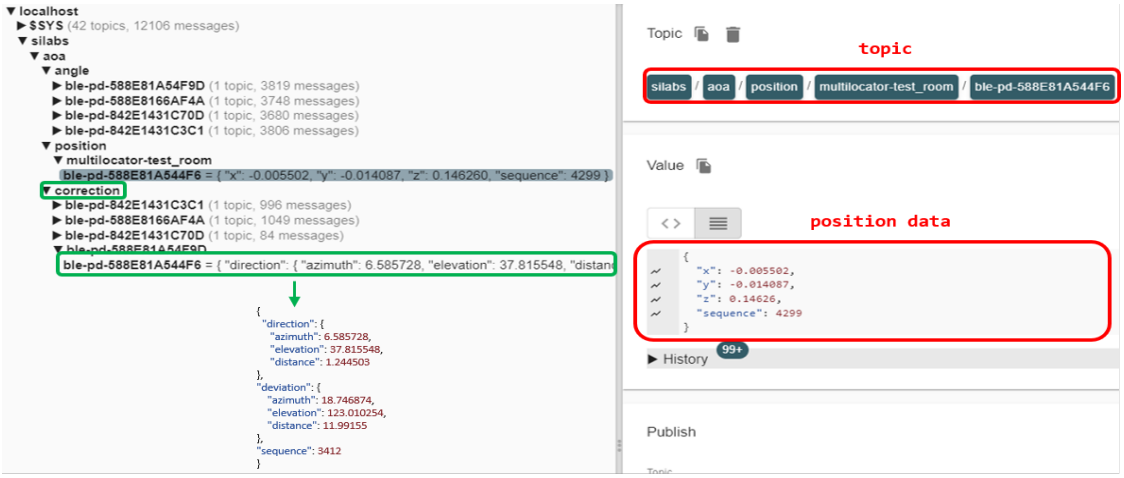
1. Flash a tag device (such as a Thunderboard BG22) with a bootloader and the asset tag sample project of your choice as described in 2 Bluetooth - SoC AoA Asset Tag. In a multi-locator scenario connectionless or Silicon Labs proprietary mode is highly recommended.
2. Flash one or more NCP AoA locator boards with the antenna arrays attached to WSTKs 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: 3.2.1 Building a Single AoA Locator Host Sample Application
4. Make sure you have installed Mosquitto MQTT broker: 1.4 Prerequisites.
5. Navigate to the /app/bluetooth/example_host/aoa_locator folder.
6. Open Mintty bash and build the project by executing `make APP_MODE=<conn_type>` inside the project directory. Where `<conn_type>` is the connection type of your choice: `conn`, `conn_less`, or `silabs`.

   If you want the aoa_locator to publish IQ reports to MQTT broker but do not calculate angle value, replace the above command with:

```
make APP_MODE=<conn_type> ANGLE=0
```

7. Attach the WSTKs to the PC either by USB or Ethernet.
8. Open the ***example_host/aoa_locator/exe*** folder and start the host application for each locator with their USB COM or IP address, for example `.\aoa_locator.exe -u COM57`.

   If you want to apply constraints for a given locator, as specified in a locator_config.json file, also provide the locator config file as a parameter, for example `.\aoa_locator.exe -u COM57 -c ../config/locator_config1.json`. The apps started for each locator must run simultaneously, so you will need more command prompts. For more accurate results, use at least four locators in your setup.
9. Navigate to the **/app/bluetooth/example_host/aoa_multilocator** folder.
10. Open Mintty bash and build the project by executing `make`. If you want the aoa_multilocator application to receive IQ reports, and calculate angle and position values, execute `make ANGLE=1` instead.
11. Find the default multi-locator configuration file under */app/bluetooth/example_host/aoa_multilocator/config/multilocator_config.json* and modify it according to the IDs, coordinates, and orientations of your locators.
12. Start the multi-locator host application from a command line with the config file as parameter, for example `.\exe\aoa_multilocator.exe -c .\config\multilocator_config.json`.
13. If the MQTT broker is not using the default host:port parameters, configure it with the -m switch, just like in the case of the *aoa_locator* sample app.
14. Open the MQTT Explorer to monitor the angle and position data. You should see something like the following.



By default, the multi-locator 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.

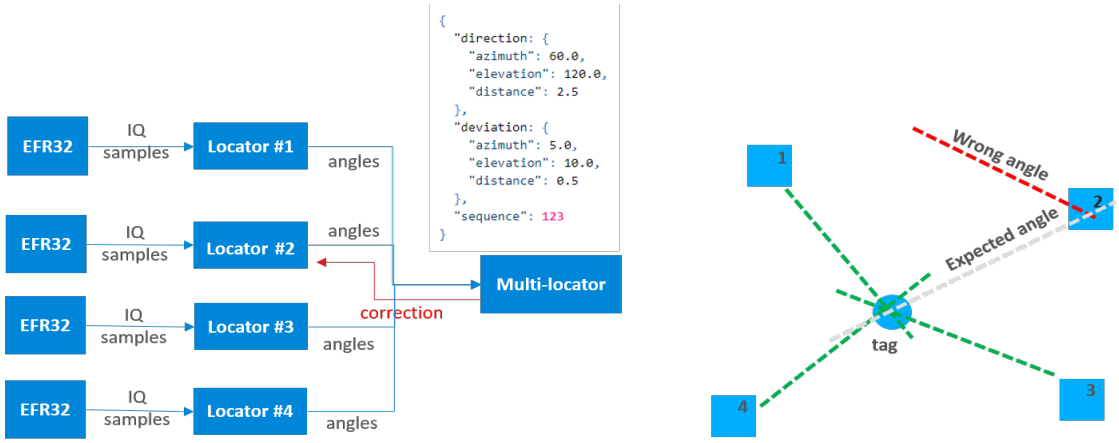


Figure 4-10. Multi-locator Feedback Mechanism

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

```
.\exe\aoa_multilocator.exe -c .\config\multilocator_config.json -n.
```

The multi-locator 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 multi-locator 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
```

Note that each CTE packet contains a 16-bit event counter. The multi-locator 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 multi-locator will not give any results. Thus, the connection mode should not be used to test the multi-locator application.

4.4 Visualization Script for Silicon Labs Multi-Locator Sample Application

The Bluetooth SDK also provides a Python script to visualize the results of the multi-locator application, since it would be extremely hard to see through the textual results when following many tags.

The visualization script is a separate application that collects the data (that is the x, y, z coordinates of the tags) by subscribing to the topic where the *aoa_multilocator* sample app publishes the position estimation results (x, y, z coordinates). Therefore, it is also an example for the *user application* as depicted in the figure at the beginning of section 4 Multi-Locator Sample Application.

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

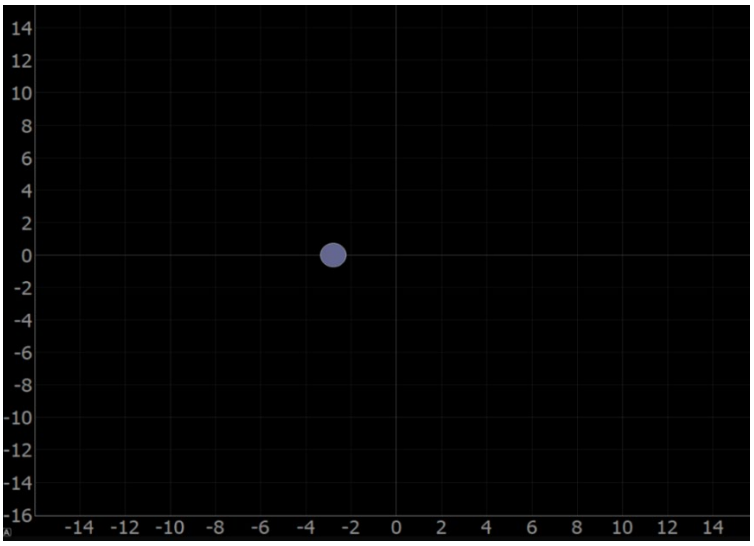


Figure 4-11. The Position of One Asset Tag Being Plotted in the 2D View

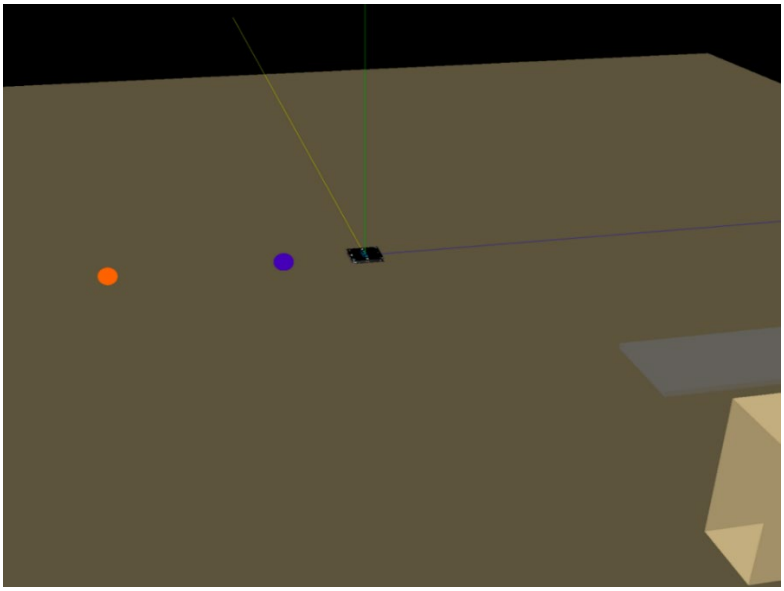


Figure 4-12. Positions of two tags and one locator shown in the 3D view

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

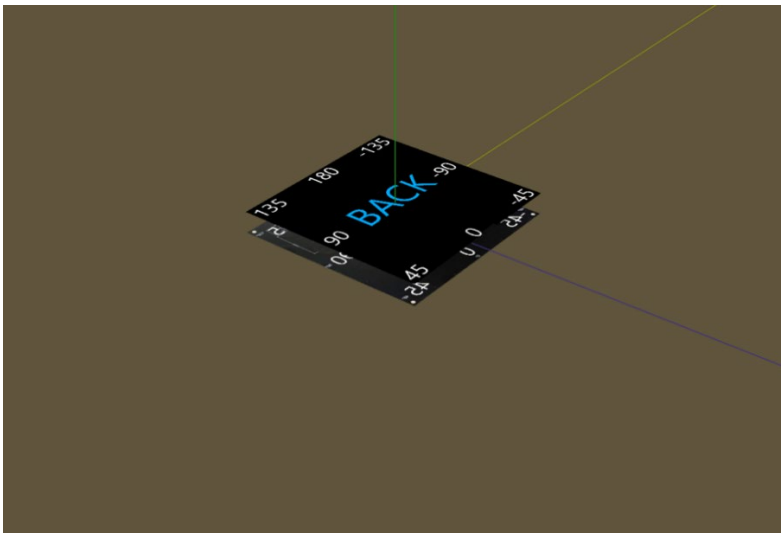


Figure 4-13. One of the Locators Shown in the 3D View

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/aoa_multilocator_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 multi-locator sample app with visualization

1. Follow the instructions in section 4.2 Testing the Multi-Locator Sample App to get the multi-locator 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/aoa_multilocator_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 multi-locator config file to be able to display the locators and to be able to subscribe to the appropriate topics at the MQTT broker. If you do not use the default multilocator_config.json file found under /app/buetooth/example_host/aoa_multilocator/config, then define its location with the -c switch as for the aoa_multilocator sample app:

```
py -3.7 app.py -c ./../aoa_multilocator/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 *aoa_locator* and *aoa_multilocator* applications are running on the same machine.

To disable the logs, open `app_log_config.h` in *C:\SiliconLabs\SimplicityStudio\v5\developer\sdks\gecko_sdk_suite\<version>\app\bluetooth\common_host\app_log\config* and make the following change:

```
#define APP_LOG_ENABLE      0
```

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