

Amazon Sidewalk

[Developing with Amazon Sidewalk](#)

[Release Notes](#)

[Amazon Sidewalk](#)

[Out-of-the-Box Demo](#)

[Overview](#)

[Kit Preparation](#)

[Sidewalk Network Access](#)

[Run the Demo](#)

[Going Further](#)

[Getting Started](#)

[Overview](#)

[Prerequisites](#)

[Create and Compile an Application](#)

[Provision your Device](#)

[Interacting with the Cloud](#)

[Protocol Overview](#)

[Overview](#)

[Frustration Free Networking](#)

[FSK Configuration](#)

[CSS Configuration](#)

[Multi-link and Auto Connect](#)

[Developer's Guide](#)

[Overview](#)

[Stack Structure](#)

[Application Development Walkthrough](#)

[Amazon Sidewalk API](#)

[Testing and Debugging](#)

[Power Consumption Analysis](#)

[Performance](#)

[Multiprotocol with Sidewalk](#)

[Troubleshooting](#)

[PAL API Reference](#)

[Interfaces](#)

[SAL](#)

[Common Interface](#)

[Type definitions](#)

[platform_parameters_t](#)

[Critical Region Interface](#)

Logging Interface

[sid_pal_log_buffer](#)

[Type definitions](#)

Peripheral Interfaces

GPIO

[Type definitions](#)

Serial Bus Interface

[Type definitions](#)

[sid_pal_serial_bus_client](#)

[sid_pal_serial_bus_iface](#)

[sid_pal_serial_bus_factory](#)

Serial Client Interface

[Type definitions](#)

[sid_pal_serial_callbacks_t](#)

[sid_pal_serial_params_t](#)

[sid_pal_serial_ifc_s](#)

[sid_pal_serial_client_factory_t](#)

Temperature

Radio Interfaces

FSK Interface

[Type definitions](#)

[sid_pal_radio_fsk_cad_params_t](#)

[sid_pal_radio_fsk_modulation_params_t](#)

[sid_pal_radio_fsk_packet_params_t](#)

[sid_pal_radio_fsk_phy_hdr_t](#)

[sid_pal_radio_fsk_pkt_cfg_t](#)

[sid_pal_radio_fsk_rx_packet_status_t](#)

[sid_pal_radio_fsk_phy_settings_t](#)

LoRa Interface

[Type definitions](#)

[sid_pal_radio_lora_modulation_params_t](#)

[sid_pal_radio_lora_packet_params_t](#)

[sid_pal_radio_lora_rx_packet_status_t](#)

[sid_pal_radio_lora_cad_params_t](#)

[sid_pal_radio_lora_phy_settings_t](#)

Sub-GHz Interface

[Type definitions](#)

[sid_pal_radio_rx_packet_t](#)

[sid_pal_radio_packet_cfg_t](#)

[sid_pal_radio_tx_packet_t](#)

[sid_pal_radio_state_transition_timings_t](#)

SWI Interface

Security and Crypto

[Type definitions](#)

- [sid_pal_hash_params_t](#)
- [sid_pal_hmac_params_t](#)
- [sid_pal_aes_params_t](#)
- [sid_pal_aead_params_t](#)
- [sid_pal_dsa_params_t](#)
- [sid_pal_ecdh_params_t](#)
- [sid_pal_ecc_key_gen_params_t](#)

[Storage Interface](#)

- [KV Storage](#)

- [Manufacturing Page](#)

- [Type definitions](#)

- [sid_pal_mfg_store_region_t](#)

[Timer Interfaces](#)

- [Delay](#)

- [Timer](#)

- [Type definitions](#)

- [Uptime](#)

[Sidewalk PAL](#)

- [BLE Adaptation](#)

- [BLE adapter](#)

- [Type definitions](#)

- [sid_pal_ble_profile_config_t](#)

- [sid_pal_ble_adapter_ctx_t](#)

[GPIO](#)

- [Type definitions](#)

- [GPIO_PinConfig](#)

- [GPIO_LookupItem](#)

[NVM3 Manager](#)

- [Type definitions](#)

[Timer Types](#)

- [Type definitions](#)

- [sid_pal_timer_impl_t](#)

[Overview](#)

[Sidewalk Sample Applications](#)

[Platform Resources](#)

[Overview](#)

[Bootloading](#)

- [Overview](#)

- [Bootloader Fundamentals \(PDF\)](#)

- [Silicon Labs Gecko Bootloader User's Guide \(PDF\)](#)

[Non-Volatile Memory Use](#)

- [Overview](#)

- [Non-Volatile Data Storage Fundamentals \(PDF\)](#)

- [Using NVM3 Data Storage \(PDF\)](#)

Security

[Overview](#)

[IoT Security Fundamentals \(PDF\)](#)

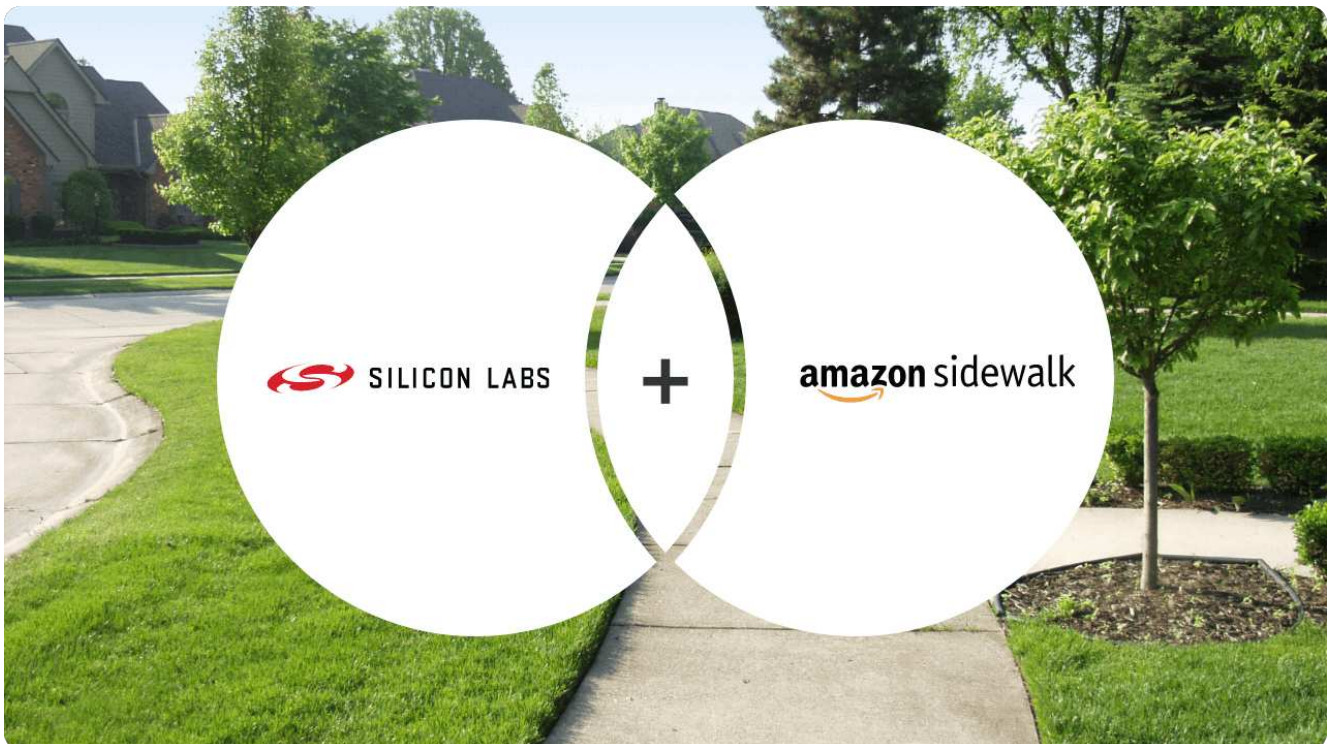
[Integrating Crypto Functionality with PSA Crypto vs. Mbed TLS \(PDF\)](#)

[Manufacturing a Product](#)

Developing with Amazon Sidewalk

Developing with Amazon Sidewalk

Amazon Sidewalk is a shared wireless network that uses Amazon Sidewalk Bridges, such as compatible Amazon Echo and Ring devices, to enable communication among devices operating on the network. Amazon Sidewalk enables reliable, low-bandwidth, and long-range connectivity at home and beyond. It connects IoT devices and applications such as outdoor lights, motion sensors, and location-based devices. It uses Bluetooth Low Energy for short-distance communication and CSS (Chirp Spread Spectrum) and FSK (Frequency-Shift Keying) radio protocols at 900 MHz frequencies to cover longer distances.



In an Amazon Sidewalk product, the provisioning takes place at the factory with keys created specifically for Amazon Sidewalk. Upon its first boot, the device will automatically join the network as long as there is an Amazon gateway within range. Registration can occur either via BLE or FSK, depending on the configuration and radios supported by your hardware. While Sidewalk supports BLE, FSK, and CSS radio links, registration cannot occur on CSS. Registration happens only once in the device's lifetime and allows it to join the Sidewalk network, rather than a specific user network. Sidewalk is a collaborative and shared network that does not facilitate device-to-device communication. Instead, all communications are routed to the cloud and then sent back to the appropriate destination, with network routing managed in the cloud.

These pages are intended for those who are actively exploring or already developing an application using the Silicon Labs integrated solution for Amazon Sidewalk. These pages focus on application development using Silicon Labs integrated solution for Amazon Sidewalk. For a high-level overview of the entire process, from buying parts to manufacturing and deployment, see the [Amazon Sidewalk Developer Journey with Silicon Labs](#).

For details about this release: Sidewalk extension [release notes](#) are available on Github along with the known issues list.

For our recommended Amazon Sidewalk development platform: See the [Silicon Labs Pro Kit for Amazon Sidewalk](#).

To see the Silicon Labs Amazon Sidewalk integrated solution in action: Follow the Pro Kit [out-of-the-box demo instructions](#).

For background about the protocol: The [Amazon Sidewalk Protocol Overview](#) discusses important elements of the specification.

To start your Amazon Sidewalk development using Simplicity Studio example applications: See the [Getting Started Guide](#).

If you are already in development: See the [Developer's Guide](#) for resources to support crucial aspects of your development effort. You can also refer to the [Amazon Sidewalk Sid API Developer Guide](#).

Amazon Sidewalk

Amazon Sidewalk Version 2.5.0 (June 30 2025) - Release Notes

[Simplicity SDK Version 2025.6.0](#)

Amazon Sidewalk is a secure, low-bandwidth, long-range wireless protocol designed to connect smart devices and extend their range beyond standard Wi-Fi or Bluetooth. It enables seamless connectivity for IoT devices in homes and neighborhoods, supporting features like device tracking, smart lighting, and sensor networks. For information about previous releases, see the [release notes archive](#).

Release Summary

[Key Features](#) | [API Changes](#) | [Bug Fixes](#) | [Chip Enablement](#)

Key Features

- Supporting SiSDK 2025.6.0.
- Supporting Semtech's LR1110.
- RAIL, sx1262, and Ir1110 as a SW component.

API Changes

None.

Bug Fixes

- Fixed a bug where the RAIL calibration error was not handled correctly on xG28 with DMP using FSK link.
- Fixed a bug where warning log messages about TX power were displayed when using RAIL radio.

Chip Enablement

xGM260P support.

Key Features

[New Features](#) | [Enhancements](#) | [Removed Features](#) | [Deprecated Features](#)

New Features

None.

Enhancements

- SLCP files are merged for every sample application.
- BLE-only mode can be selected for chips which support BLE. SubGHz component can be disabled.
- Out of the Box demo binaries are updated, now using latest Sidewalk stack and SiSDK version.

Removed Features

None.

Deprecated Features

None.

API Changes

[New APIs](#) | [Modified APIs](#) | [Removed APIs](#) | [Deprecated APIs](#)

New APIs

None.

Modified APIs

None.

Removed APIs

None.

Deprecated APIs

None.

Bug Fixes

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1410675	RAIL calibration error on xG28 with DMP using FSK link.	None.	<ul style="list-style-type: none">Amazon Sidewalk - SoC Dynamic Multiprotocol LightBRD4401C, BRD4400CSoC
1452596	Warning log messages about TX power when using RAIL radio.	None.	<ul style="list-style-type: none">Amazon Sidewalk - SoC Hello Neighbor, Amazon Sidewalk - SoC CLIAll the xg28, xg23 radio boardsSoCPAL

Chip Enablement

Chip Family	OPNs / Boards / OPN Combinations	Supported Software Variants (if applicable)	Supported Modes	Supported Host Interfaces
MGM260P	<ul style="list-style-type: none">OPN: EFR32MG26B420F3200IM48-B, EFR32MG26B410F3200IM48-B, EFR32MG26B421F3200IM48-B, EFR32MG26B411F3200IM48-B, EFR32BG26B411F3200IM48-B, EFR32BG26B421F3200IM48-BBoards: BRD4350A, BRD4351A, BRD2713A	N/A	SoC	UART, RTT

Application Example Changes

[New Examples](#) | [Modified Examples](#) | [Removed Examples](#) | [Deprecated Examples](#)

New Examples

None.

Modified Examples

None.

Removed Examples

None.

Deprecated Examples

None.

Known Issues and Limitations

ID	Issue or Limitation Description	GitHub / Salesforce Reference (if any)	Workaround (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1469745	Wrong RSSI value on xG28 when using FSK.	N/A	No workaround is possible.	<ul style="list-style-type: none">EFR32ZG28B322F1024, EFR32ZG28B320F1024, EFR32ZG28B312F1024BRD4401C, BRD4400C, BRD4401B, BRD2705ASoC

Impact of Release Changes

[Impact Statements](#) | [Migration Guide](#)

Impact Statements

None.

Migration Guide

None.

Using This Release

[What's in the Release?](#) | [Compatible Software](#) | [Installation and Use](#) | [Help and Feedback](#)

What's in the Release?

The Sidewalk MCU SDK v2.5.0 release introduces several new capabilities and platform expansions. This release includes updated Out-of-the-Box (OOB) binaries, support for the Semtech LR1110 chip, and enables Semtech-based solutions on EFR32xG28 SoCs. It also adds compatibility with the latest Silicon Labs Series 3 devices, including the xG301 family. Developers benefit from an easier configuration flow allowing seamless switching between Sub-GHz driver configurations and BLE-only solutions. Additionally, the SDK now supports the latest Silicon Labs SDK version 2025.6.0. This release includes Sidewalk SDK version 1.18.

Compatible Software

Software	Compatible Version or Variant
Simplicity SDK	SiSDK 2025.6.0
Amazon Sidewalk SDK	1.18

Installation and Use

To run your first demo, see our [Getting Started Guides](#).

To kick start your development, see our [Developer's Guide](#).

For information about Secure Vault Integration, see [Secure Vault](#).

To review Security and Software Advisory notifications and manage your notification preferences:

1. Go to <https://community.silabs.com/>.
2. Log in with your account credentials.
3. Click your profile icon in the upper-right corner of the page.
4. Select **Notifications** from the dropdown menu.
5. In the Notifications section, go to the **My Product Notifications** tab to review historical Security and Software Advisory notifications
6. To manage your preferences, use the **Manage Notifications** tab to customize which product updates and advisories you receive.

For recommended settings, configurations, and usage guidelines, see our [Developer's Guide](#).

To learn more about the software in this release, dive into our [online documentation](#).

Help and Feedback

- Contact [Silicon Labs Support](#).
- To use our **Ask AI** tool to get answers, see the search field at the top of [this page](#).

Note: Ask AI is experimental.

- Get help from our [developer community](#).

Feature Matrix

Unsupported Features

- Sidewalk Bulk Data Transfer (SBDT): Sidewalk BLE integrated OTA mechanism is available in our Sidewalk alpha repository as it is not publicly accessible on Amazon AWS side.
- Switched Multiprotocol (SMP): Sidewalk, BLE, Z-Wave SMP PoC is available in our alpha repository.

SDK Release and Maintenance Policy

See our [SDK Release and Maintenance Policy](#).

Out-of-the-Box Demo

Running the "Out-of-the-Box" Demo with Silicon Labs Pro Kit for Amazon Sidewalk

Congratulations on your purchase of the Silicon Labs Pro Kit for Amazon Sidewalk! This guide will help you get the most from the "out-of-the-box" demo application that is pre-installed on the radio boards in your new kit.

- **Prepare the Kit Hardware:** Describes your options with the Pro Kit and Explorer Kit hardware, and explains how to prepare your kit to run the demo.
- **Ensure Access to the Amazon Sidewalk Network:** Provides details on connecting the endpoint created by the demo application with the Amazon Sidewalk network through an in-range Sidewalk gateway.
- **Run the Demo:** Describes how, with the endpoint ready and a Sidewalk gateway nearby, you can establish your Sidewalk endpoint and access the associated AWS application from your mobile device.
- **Going Further:** Includes "next steps" to explore once you've witnessed the ease with which the Sidewalk network allows you to interact between an endpoint and the cloud, along with helpful tips if your demo did not operate smoothly.

Kit Preparation

Prepare the Kit Hardware

Silicon Labs offers two kits for Amazon Sidewalk: the Pro Kit and the Explorer Kit. These kits allow users to utilize the "out-of-the-box" demo application, which demonstrates the use of Sidewalk.

INFO: A USB-C cable, required to power the device, is *not* included in the kit. You must use a cable that is appropriate for your power source.

Mainboards and Radio Boards

In case of the Pro Kit, required kit items for the "out-of-the-box" demo are:

- Wireless Pro Kit Mainboard (BRD4002A), which provides power and peripherals to the radio boards
- Either of the two radio boards:
 - EFR32xG24 2.4 GHz 20 dBm Radio Board (BRD4187C)
 - KG100S Sidewalk Module Radio Board (BRD4332A) + 915Mhz antenna

Demo firmware is pre-flashed on both the EFR32xG24 and the KG100S Module radio boards. Silicon Labs recommends using the EFR32xG24 when you first run the demo, as this version presents a more complete experience. The differences are:

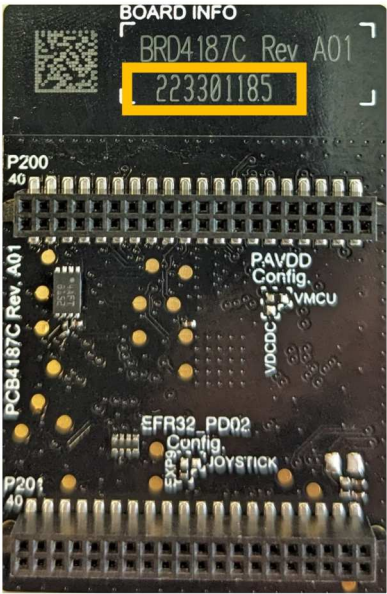
1. EFR32xG24 Radio Board: This demo app drives the LCD on the Mainboard to display a QR code for your mobile device to scan. This application uses only Bluetooth Low Energy (BLE) to communicate with the gateway.
2. KG100S Radio Board: The QR code is printed on a sticker on the KG100S radio board. Make sure to power up the board first before scanning the QR code. This application communicates with the gateway over both BLE and the sub-GHz FSK radios.

In case of the Explorer Kit, required kit items for the "out-of-the-box" demo are:

- EFR32xG28 Explorer Kit (BRD2705A)

Demo firmware is pre-flashed on the EFR32xG28.

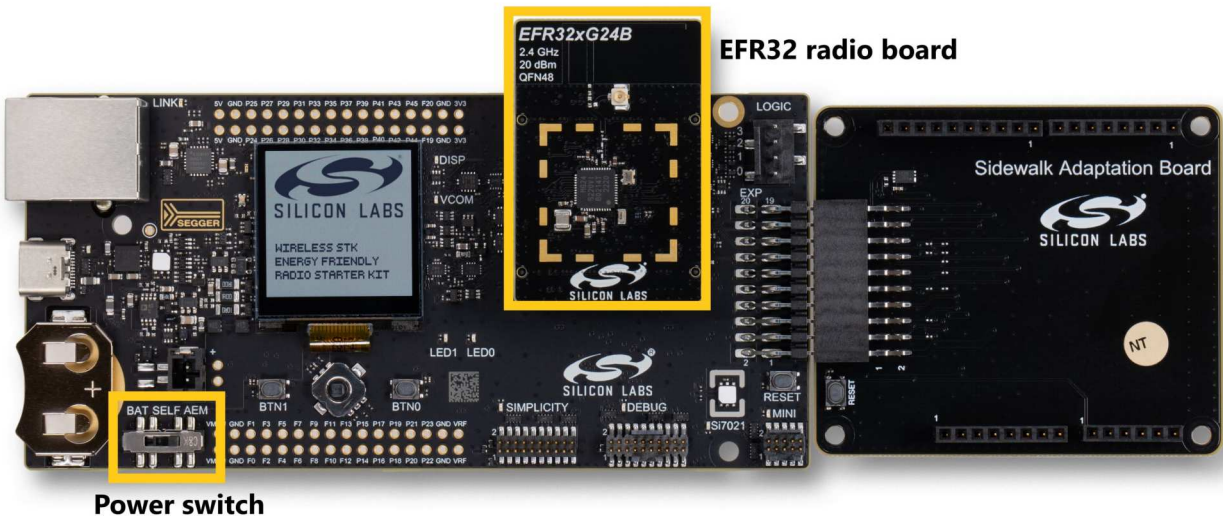
As an alternative to scanning the QR code, the radio board serial number can be entered in Silicon Labs' "out-of-the-box" demo Web interface at silabssidewalkdemo.com. The radio board serial number is located behind the board as shown in the following picture.



Assembling the Demo Hardware

① INFO ①: This step is only necessary in case of the Pro Kit.

Install the radio board on the Mainboard. Take care to align the contact points between the radio and Mainboard. For reference, the EFR32xG24 is shown mounted on a Mainboard.



Ensure that the power switch is in the **AEM** position, which supports supplying power to the Mainboard as described in the next section. The additional Sidewalk Adaptation Board shown in the image above is not needed for this demo.

If you are using the KG100S Module, you must also connect the 915 MHz antenna onto that radio board.

Powering the Demo Hardware

With the power switch in the **AEM** position, the kit hardware is powered through the Mainboard's USB-C connector. Connecting the Mainboard to a USB power source starts the demo, so first check for network availability as described in the next step.

There is no **AEM** switch for the Explorer Kit. Directly connect the board to a USB power source.

For the purposes of the "out-of-the-box" demo, only bus power is required. Enumeration with a USB host is not necessary.

Next Step: Sidewalk Network Access

With your kit assembled and a USB-C power source available, next confirm [Amazon Sidewalk Network Availability](#) at your location.

Ensure Access to the Amazon Sidewalk Network

Sidewalk Network Coverage

⚠ WARNING ⚠: Regional regulations may also restrict unauthorized usage of the Sidewalk-supported sub-GHz bands outside of the United States. Consult local laws and ensure compliance (radio isolation by RF chamber, etc.) as needed. See [Using Sidewalk Gateways Outside the USA](#) for more information.

The image displays two maps. The top map is a detailed view of the Greater Boston area, showing a grid of streets and highways. Key locations labeled include Cambridge, Boston, Brookline, and Chelsea. The bottom map is a smaller, regional view of the Eastern United States, showing state boundaries and major cities like New York, Toronto, and Quebec. The map includes labels for various states and provinces, such as Ontario, Québec, and Maine.

The Amazon Echo 4th generation is the recommended Amazon Sidewalk gateway for development purposes. Silicon Labs validates the Amazon Sidewalk software development kit against this product.

Follow the standard Echo product configuration procedures, and make sure to enable Sidewalk support on your gateway so it will access the Sidewalk network.

① INFO ①: To enable Sidewalk support, visit [Enable or Disable Amazon Sidewalk for Your Account](#) for more help.

For more information on this topic, see the [Getting Started Prerequisites](#) page.

Next Step: Run the Demo

After you have access to the Amazon Sidewalk network, you are ready to [Run the Demo](#).

Run the Demo

Running the Out-of-the-Box Demo

With your chosen radio board mounted and an Amazon Sidewalk gateway in BLE range of your device, you can run the demo by simply connecting USB-C power to the Mainboard.

At power on, the pre-installed demo application forms a Sidewalk endpoint that automatically registers with the network through the nearest gateway. Amber LED(s) on the Mainboard then blink periodically to indicate that the endpoint is awaiting time sync with the Sidewalk network. Once properly synchronized, the LED(s) stop blinking and the endpoint associates with a web application in the cloud, allowing you to interact with the endpoint over the Sidewalk network.

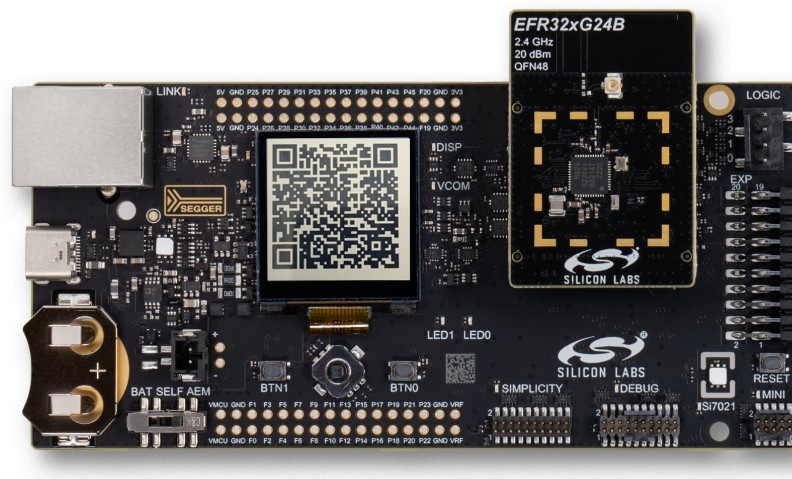
This entire process occurs within about a minute (or less) of powering on your device. If you are using the EFR32xG24 radio board, you will see a QR code displayed on the LCD. If you do not see a QR code (or if, when using the KG100S or xG28, LED1 continues to blink) after a minute or two, see [Troubleshooting the OOB Demo](#).

Accessing the Demo Application in the Cloud

The method for accessing the cloud application varies depending on the radio board. Each approach is detailed in the following sections.

When Using the EFR32xG24

In this version of the demo, the EFR32xG24 drives the LCD on the Mainboard. Once the web application is prepared, a QR code is displayed on the LCD.



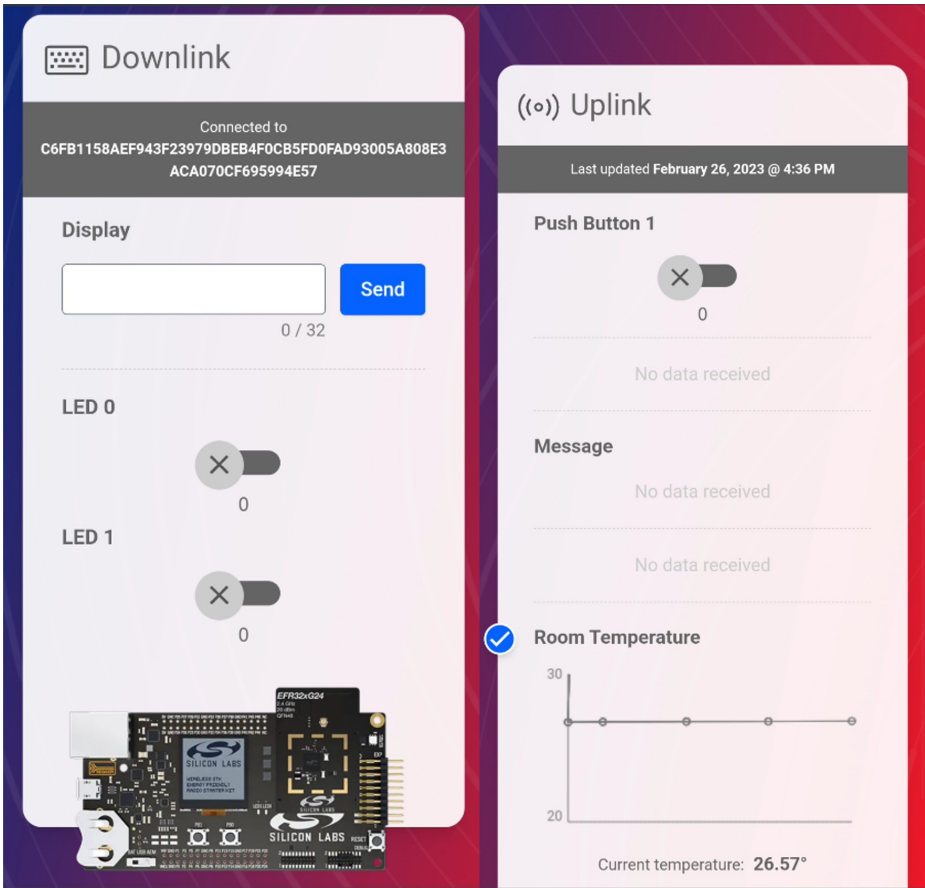
Use your mobile device or webcam to read this QR code and access the embedded URL in a browser. This page loads the demo web application already associated with your Sidewalk end device.

When Using the KG100S Sidewalk Module or xG28 Explorer Kit

A QR code sticker was applied to the device during manufacturing that already encodes the web application URL. After powering on the device, once LED1 stops blinking, you can read the embedded URL and access the web application in your browser.

How to use the Cloud Demo Application

The cloud application is divided into two main sections, each providing a different way to interact over the Sidewalk network. These features are described below.



Downlink Section

These elements send data from the cloud down to the end device. For each board, you have access to a set of capabilities:

- **Display:** If your demo is using the onboard LCD, type a message in this field and click **Send**. Your message is sent to the endpoint and appears on the LCD.
- **LED 0:** Tap to flip this toggle on or off. For the EFR32xG24 and EFR32xG28, LED0 on the Mainboard updates to your desired state. The KG100S radio board does not use this LED.
- **LED 1:** Tap to flip this toggle on or off. For the EFR32xG24 and KG100S, LED1 on the Mainboard updates to your desired state. The EFR32xG28 explorer kit does not use this LED.

Capability	Description	xG24	KG100S	xG28
Display	Type a message in the field on the web application and click Send . Your message is sent to the endpoint and appears on the LCD.	X		
LED0	Tap to flip this toggle on or off.	X		X
LED1	Tap to flip this toggle on or off.	X	X	

Uplink Section

These elements display data sent from the end device up to the cloud. For each board, you have access to a set of capabilities:

- **Push Button 1:** Press the associated button on your Mainboard. The graphic toggles between 0 and 1 with each button press.
- **Room Temperature:** Ambient temperature is sampled by the endpoint, sent through the Sidewalk network to the cloud, and plotted over time on this Room Temperature chart. The temperature sensor is the Si7021 mounted on the Mainboard.

Capability	Description	xG24	KG100S	xG28
Push Button 1	Press the associated button on your Mainboard. The graphic toggles between 0 and 1 with each button press.	X	X	X
Push Button 0	Press the associated button on your Mainboard. The graphic increments the message counter			X
Room Temperature	Ambient temperature is sampled by the endpoint, sent through the Sidewalk network to the cloud, and plotted over time on this Room Temperature chart. The temperature sensor is the Si7021 mounted on the Mainboard.	X	X	X

For the EFR32xG28 and KG100S radio boards, dummy temperature data is sampled by the endpoint.

Next Step: Going Further

When you're ready to move beyond the out-of-the-box demo and more fully explore the Amazon Sidewalk developer experience provided by Silicon Labs, visit the [Going Further](#) page. You'll find resources to help you move beyond the demo, along with guidance to troubleshoot any issues you may have.

Going Further

Going Beyond the Demo

The Silicon Labs Kits for Amazon Sidewalk demo were designed to provide users with effortless access to the kind of streamlined experience that Amazon Sidewalk can provide for your customers.

Silicon Labs supports Amazon Sidewalk with a set of tools, capability, and documentation that similarly lowers the barriers of entry to Amazon Sidewalk application development.

The following resources are great places to start if you're ready to learn more.

Developer Resources

- [Getting Started Guide](#): Step-by-step journey through AWS (deploying application, profile and device creation) and building an example Amazon Sidewalk application in Simplicity Studio.
- [Developer's Guide](#): Goes beyond the "Getting Started" introduction to deep-dive into the most important aspects of embedded development for Amazon Sidewalk with Silicon Labs.
- [Amazon Sidewalk Protocol Overview](#): Review of many relevant technical elements of the Amazon Sidewalk network.

A Note on Preserving the Demo Functionality

The out-of-the-box demo relies on device-specific credentials pre-flashed to the USERDATA page on your device. This page is not affected by mass-erase operations, but *can* be explicitly erased by a targeted page erase.

Typical erase/flash cycles as you repurpose the kit radio boards during development are of no concern. In fact, [Credentials Backup/Restore Feature](#) describes a process by which you can restore a working out-of-the-box demo application after a mass-erase. However, care should be taken to NOT perform a page erase of USERDATA, or the out-of-the-box demo cannot be restored.

The backup/restore feature from OOB demo performs as follows:

1. If you erase the main flash: The device should recover the credentials on first boot after flashing the application binary.
2. If you erase the user data partition: The device should recover by itself on first boot.
3. If you erase both the main flash and user data partition: Your device credentials are not recoverable.

This is especially important if you have one of Silicon Labs Pro Kits for Amazon Sidewalk. If you wish to use the radio boards with an application other than the demo, you can erase the main flash and use any application you like with your own manufacturing page. As long as you have not erased the user data partition, your device will recover the demo when you flash the out-of-the-box application binary back on your device. You should never erase the user data partition of your Pro Kit's radio boards.

Troubleshooting the Out-of-the-Box Demo

If you encountered problems running the demo, the following guidance may help get things back on track:

- Kit Setup
 - Ensure the Mainboard power switch is in the **AEM** position.
 - Disconnect USB-C power, then re-seat the radio board.
- Board-Specific
 - If not already, try using the EFR32xG24 radio board. This demo version relies only on BLE, and provides feedback at various stages on the LCD.
 - If using the KG100S radio board, install the 915 MHz antenna found in your kit.
- Network-Specific
 - If relying on ambient Sidewalk network coverage, obtain your own gateway to ensure network access.

- If using your own gateway, confirm that Sidewalk is **enabled**, and verify the gateway has access to the internet (try asking *"Alexa, are you online?"*) from a US-based IP address.

If these all fail, review the [Getting Started Guide](#). This will incrementally build what should become a known-good baseline (verifying kit hardware integrity and Sidewalk network access along the way).

For Silicon Labs Technical Support, contact us on our [Support Platform](#).

Getting Started

Getting Started with Amazon Sidewalk Development in Simplicity Studio

- **Prerequisites:** Describes the hardware and software prerequisites for starting your development.
- **Create and Compile your Sidewalk Application:** Once Sidewalk is added to your Simplicity SDK, you can choose a sample Sidewalk application to compile and flash on your device.
- **Provision your Amazon Sidewalk Device:** Now your application is running on your endpoint, and you can create the application server on AWS and add the corresponding certificates to your device.
- **Interacting with the Cloud:** As the last step, you can send and receive messages between your endpoint and the server application in AWS.

For Silicon Labs Technical Support, contact us on our [Support Platform](#).

Prerequisites

Prerequisites

To develop for Amazon Sidewalk on Silicon Labs SoCs and modules, you need the hardware and software resources detailed on this page.

Hardware

Silicon Labs Wireless Development Hardware

Silicon Labs produces many different kits to support development for a broad range of wireless technologies. New users can jump-start their journey with a [Silicon Labs Pro Kit for Amazon Sidewalk](#). While experienced users may have much of the required hardware on hand already, Silicon Labs recommends they too start with [Silicon Labs Pro Kit for Amazon Sidewalk](#).

For Amazon Sidewalk solutions, the typical Silicon Labs development hardware is a wireless main board paired with a radio board that supports Sidewalk. The Sidewalk-specific wireless technologies supported in this configuration (Bluetooth Low Energy (BLE), sub-GHz FSK, and sub-GHz CSS) depend on the radio board, as shown in the following table. Note that for sub-GHz support, some devices will require both an EFR32 radio board and an additional Semtech LoRa shield with the appropriate adapter board. Advanced users who have already bought a Pro Kit or have a WSTK/WPK can augment their existing hardware with additional radio boards as needed.

The following list and table summarizes these requirements. You can also read our [hardware selector guide](#) to help choose appropriate hardware for your use case.

- A Sidewalk-supported Radio board:
 - [EFR32xG21B Radio Board](#) (BRD4181C)
 - [EFR32xG23B Radio Board](#) (BRD4204D, BRD4210A, BRD4264C, BRD4263C, BRD4204C)
 - [EFR32FG23 868-915 MHz +14 dBm Dev Kit](#) (BRD2600A)
 - [EFR32MG24B Radio Board](#) (BRD4186B, BRD4186C, BRD4187B, or BRD4187C - included in the [Silicon Labs Pro Kit for Amazon Sidewalk](#))
 - [EFR32xG26B Radio Board](#) (BRD2608A, BRD4118A, BRD4121A, BRD41201, BRD4117A, BRD4116A)
 - [EFR32xG28B Radio Board](#) (BRD2705A, BRD4400A, BRD4400B, BRD4400C, BRD4401A, BRD4401B, BRD4401C)
 - [KG100S Module Radio Board](#) (BRD4332A, included in the [Silicon Labs Pro Kit for Amazon Sidewalk](#))
- A Wireless Starter Kit Main board (BRD4001x), or a Wireless Pro Kit Main Board (BRD4002x) included in the [Silicon Labs Pro Kit for Amazon Sidewalk](#). (*Note: a main board is NOT required when using a Dev Kit from the list above*)
- For sub-GHz applications: A 915MHz antenna (included in most sub-GHz kits) should be installed on the appropriate connector. If using EFR32xG21, EFR32xG24 or EFR32xG26, a [Semtech SX1262MB2CAS LoRa shield](#) and Sidewalk Adaptation Board (BRD8042A, included in the [Silicon Labs Pro Kit for Amazon Sidewalk](#)) are required. With EFR32xG24, the [Semtech LR1110MB1LCKS LoRa shield](#) and [Semtech Sidewalk Adaptation Board \(BRD8102A\)](#) can be used to enable location services.

Radio Board Model	BLE	FSK	CSS
EFR32xG21	x		
EFR32xG21 + Semtech SX1262 shield	x	x	x
EFR32xG23		x	
EFR32xG24	x		
EFR32xG24 + Semtech SX1262 shield	x	x	x
EFR32xG24 + Semtech LR1110 shield	x	x	x
EFR32xG26	x		

Radio Board Model	BLE	FSK	CSS
EFR32xG26 + Semtech SX1262 shield	X	X	X
EFR32xG28	X	X	
KG100S	X	X	X

INFO: Amazon Sidewalk support requires that target hardware have Secure Element (SE) firmware versions of at least v1.2.9 for xG21 SoCs, v2.1.7 for xG24 SoCs, and v1.2.9 for KG100S Modules. Follow instructions in Section 4.4 of [AN1222: Production Programming of Series 2 Devices](#) to update the SE firmware. xG28 and xG23 should come with an appropriate SE firmware version.

WARNING: EFR32xG21B and EFR32xG28 radio boards are available with devices of various flash sizes. Minimum flash size to run the Hello Neighbor example applications is 768 kB. This restriction does not apply to EFR32xG23 devices, which do not support BLE.

For simplicity, this *Getting Started Guide* focuses on the **EFR32MG24** and demonstrates how to use the **Amazon Sidewalk - SoC Hello Neighbor** example application provided in the SDK for Amazon Sidewalk.

Amazon Sidewalk Gateway

The Sidewalk protocol requires a gateway to provide endpoints access to the AWS cloud. Several Amazon products can act as a gateway. These products have different functions and varying support for Amazon Sidewalk network features. The list of Amazon Sidewalk gateways and their supported radio capabilities is available at the following link:

<https://docs.sidewalk.amazon/introduction/sidewalk-gateways.html>.

The Amazon Echo 4th generation is the recommended Amazon Sidewalk gateway for development purposes. Silicon Labs validates the Amazon Sidewalk software development kit against this product.

WARNING: Beware of the difference between Amazon Echo 4th gen and Amazon Echo **Dot**. The Echo 4th generation supports all 3 radios and is the reference hardware used for testing. The Echo **Dot** supports only BLE radio.

Developers are advised to use their own gateway, as this affords the greatest control over providing consistent network access during the development phase. However, you may already have access to the Amazon Sidewalk network at your location. Amazon provides a Sidewalk Network Coverage map you can use to see if this is likely:

<https://coverage.sidewalk.amazon>.

Sidewalk Gateway Setup

Follow the standard product installation procedures to set up a new gateway. Configuring your gateway with Sidewalk support depends on a few additional requirements:

- The gateway must be set up using a US-based Amazon account (see [Change your Amazon Account Country](#) to adjust the Country setting for an existing account)
- Amazon Sidewalk must be enabled on the Amazon account (see [Enable or Disable Amazon Sidewalk for Your Account](#) for more help)
- The gateway must have a US-localized IP address, and be configured with a US-based location (see [Change Your Alexa Device Location](#) if needed)

Using Sidewalk Gateways Outside the USA

⚠ WARNING ⚠: Amazon Sidewalk is available only in the United States of America. To the extent that any Sidewalk gateway functionality might be used outside of the U.S., it should be used **ONLY** for Amazon Sidewalk endpoint development purposes. In addition, Silicon Labs recommends that you consult with your local regulatory bodies and check if the gateway is allowed to operate its radio in your locale, as U.S. license-free band devices, only for development. Developers are solely responsible for ensuring compliance with local regulations. Additionally, Silicon Labs recommends a shielded chamber or similar apparatus to capture and contain wireless signals within your development environment.

To enable operation outside of the US for your development, you need to use a VPN router that supports OpenVPN Client functionality in conjunction with a cloud VPN service provider. Refer to the following [Amazon documentation](#) for more details. You are also solely responsible for compliance with any local regulations regarding VPN use.

① INFO ①: VPN clients running on a laptop or mobile device or using any other VPN protocol (such as L2TP or PPTP) are not supported.

Amazon Frustration-Free Setup (FFS)

A fully operational gateway is linked to an Amazon account, usually during initial setup using the Alexa app. However, this device-account linking can instead be initiated at time of purchase if you buy the gateway from Amazon and check the box labeled **Link device to your Amazon account to simplify setup**. This account linking has implications for Sidewalk feature support and gateway setup:

- Currently, FSK support is enabled on only one (by default, the first) compatible gateway linked to an account.
- If you purchase a gateway device using an Amazon account other than the one you plan to use for testing and in the Alexa app to configure the gateway, do NOT check the **Link device to your Amazon account** box. Doing so may prevent the successful setup of your gateway until you request Amazon Customer Support to remove the original device-account link. More info: [Amazon Frustration-Free Setup Frequently Asked Questions](#).
- The **Link device** checkbox also enables Amazon Wi-Fi Simple Setup, which can simplify getting your new gateway connected when powered on. However, if you plan to connect the gateway to a different SSID than you already use with other devices known to your Amazon account, the FFS-driven automatic selection of Wi-Fi networks can impede your efforts to connect the gateway to your preferred SSID.

Software

To get started with your Amazon Sidewalk development, you need:

- [Simplicity Studio 5](#), with the SDK extension for Amazon Sidewalk installed (see version and installation guidance below)
- [J-Link RTT Viewer](#) (see version guidance below)

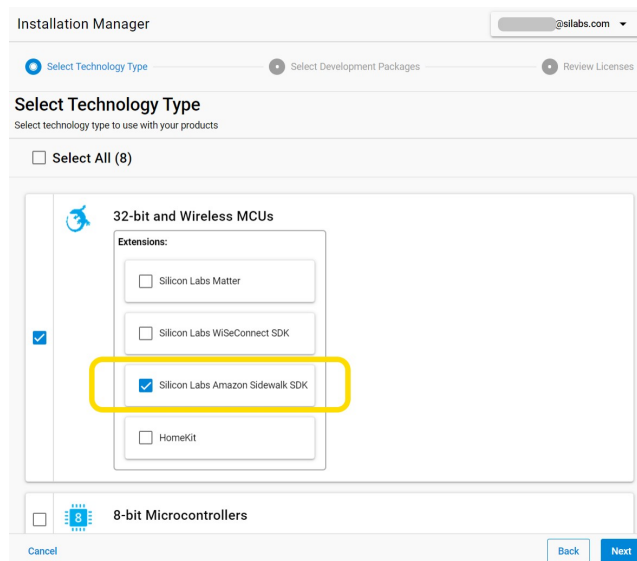
① INFO ①: Amazon Sidewalk support requires the Amazon Sidewalk SDK extension. The extension version 1.0.0 requires at least Simplicity Studio v5, the Simplicity Commander version included in that release, GSDK 4.2.2, and JLink version 7.84.

Silicon Labs SDK Extension for Amazon Sidewalk

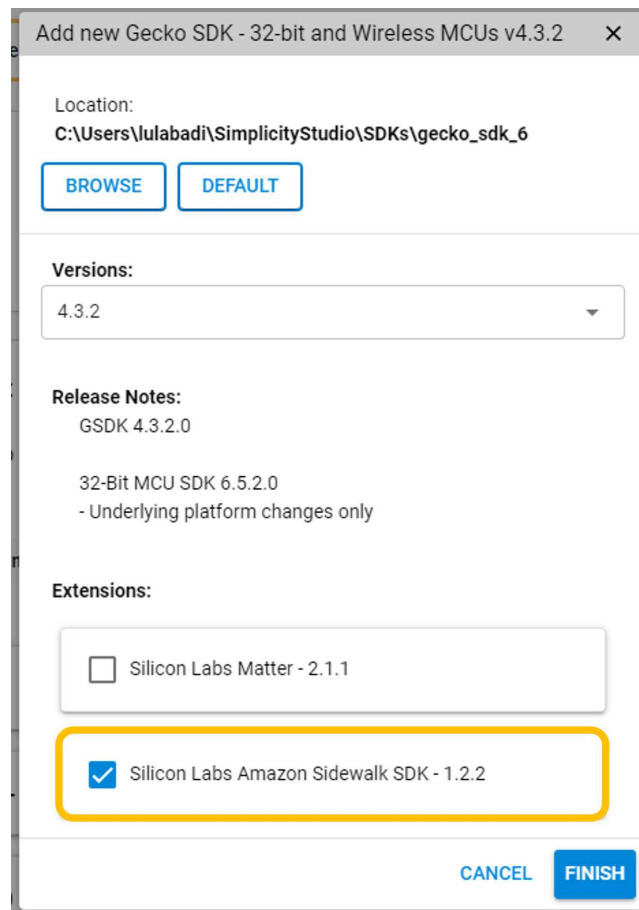
During the initial install of Simplicity Studio, support for Amazon Sidewalk can be added by checking both the **32-bit and Wireless MCUs** and the **Amazon Sidewalk SDK** extension boxes in the Installation Manager as shown in the "step 1" figure below. Click **Next** to install the latest 32-bit and Wireless MCU Simplicity SDK and the Sidewalk SDK extension.

If Simplicity Studio is already installed, follow step 1 below to add Simplicity SDK support for the first time. Alternatively, for installations that have already added at least one Simplicity SDK version, jump to step 2 to ensure you install Sidewalk support with the latest Simplicity SDK.

1. Add Simplicity SDK and SDK extension for Amazon Sidewalk to an existing Simplicity Studio installation.
 - Open [Simplicity Studio 5](#).
 - Click the **Install** icon on the toolbar and select **Install by technology type**.
 - A **Select Technology Type** dialog opens. Select **32-bit and Wireless MCUs, Amazon Sidewalk SDK**, and click **Next** (see figure below).
 - In the **Package Installation Options** dialog, select **Auto** and click **Next**.
 - Wait for the installation to complete, click **Finish**, and go to step 2 to verify the installed Simplicity SDK and Sidewalk support.



2. Add or confirm the SDK extension for Amazon Sidewalk when Simplicity SDK is already installed.
 - Open [Simplicity Studio 5](#).
 - Click the **Install** icon on the toolbar and select **Manage installed packages**.
 - In the **Installation Manager**, go to the **SDKs** tab.
 - On the **Simplicity SDK - 32-bit and Wireless MCUs** card, verify that the latest version is installed.
 - If a different version is installed, click **Add....**
 - On the **Versions** drop-down, select the latest version.
 - Note the **location** of the new Simplicity SDK folder, you may want to refer to this in a later step.
 - If not already selected, check the box next to **Amazon Sidewalk SDK**.
 - Click **Finish**.



If needed, Simplicity Studio 5 now downloads and installs the Simplicity SDK and Sidewalk SDK extension.

AWS Command Line Interface (CLI)

To perform the operations that create and manage the cloud-based elements of your Amazon Sidewalk applications, your scripts and the Sidewalk Assistant use AWS CLI. With the AWS CLI, these tools can run commands that implement functionality equivalent to that provided by the browser-based AWS Management Console. Use the resources below to set up AWS CLI for these purposes.

INFO: Note that Amazon has currently activated Sidewalk only for the North Virginia region ("us-east-1"). To support Sidewalk development (and operation), your AWS CLI and AWS web interface should be localized on this **us-east-1** region.

- [Prerequisites to use the AWS CLI](#). To prepare your system to support AWS CLI, you must create users in your AWS CLI account. For convenience, Silicon Labs recommends using *long-term IAM credentials* (via IAM) to access AWS during your evaluation:
 1. Follow the guidance in [Getting set up with IAM](#) to create an AWS account if you do not already have one.
 2. Follow steps 1 thru 3 in [Authenticate using long-term credentials](#) to create an administrator IAM account (select the *AdministratorAccess* policy), get your access keys, and update the shared credentials file. (For additional assistance on these topics, see [Getting started with IAM](#) and [Managing access keys for IAM users](#))
- [Install AWS CLI](#). If AWS CLI is not already installed, install it as described here.
- [Configure AWS CLI](#). Configure AWS CLI to leverage your IAM administrator account. (Sidewalk Assistant supports the recommended *credentials file* approach described in prior bullets and *environment variables* (except on Mac OS where environment variables are not supported).)

Getting Support

Silicon Labs supports developers in many ways, including with documentation and active community forums where other users and Silicon Labs employees offer guidance to those in need.

- [Amazon Sidewalk at Silicon Labs](#) is the home page of the site that contains this getting started guide and other helpful resources
- [Sidewalk Community Forum](#) for all things Sidewalk-related
- [Simplicity Studio 5 User's Guide](#) documents the features and usage guidance for Simplicity Studio
- [Simplicity Studio Forum](#) for assistance with the development environment and resource management provided through Simplicity Studio
- [What Do the Lights on Your Echo Device Mean?](#) can help you understand the visual feedback from your Amazon Echo device
- [Amazon Sidewalk Documentation](#) provides additional details on Amazon Sidewalk

Taking the Next Step

Once you have completed the required hardware and software setup tasks, you can begin [creating and compiling an Amazon Sidewalk Application](#).

Create and Compile an Application

Create and Compile your Sidewalk Application

Amazon Sidewalk Sample Applications

Silicon Labs SDK for Amazon Sidewalk includes multiple example applications, including:

- Amazon Sidewalk - SoC Hello Neighbor
- Amazon Sidewalk - SoC CLI
- Amazon Sidewalk - SoC Production Device Provisioner
- Amazon Sidewalk - SoC Dynamic Multiprotocol Light
- Amazon Sidewalk - SoC Qualification
- Amazon Sidewalk - SoC OOB Demo (delivered in binary only)

When a Sidewalk-supported target device is selected, additional examples are available in the Simplicity Studio **Launcher** perspective, **EXAMPLE PROJECTS & DEMOS** tab. Type "sidewalk" into the **Filter on Keywords** field at top left, and press enter to view only Amazon Sidewalk examples.

For simplicity, this guide walks through the **Amazon Sidewalk - SoC Hello Neighbor** example on an **EFR32xG24** radio board. You can then repeat this procedure with other examples in the SDK.

Additional sample applications are available in the [Silicon Labs Amazon Sidewalk Applications Github repository](#).

Create an Amazon Sidewalk Project

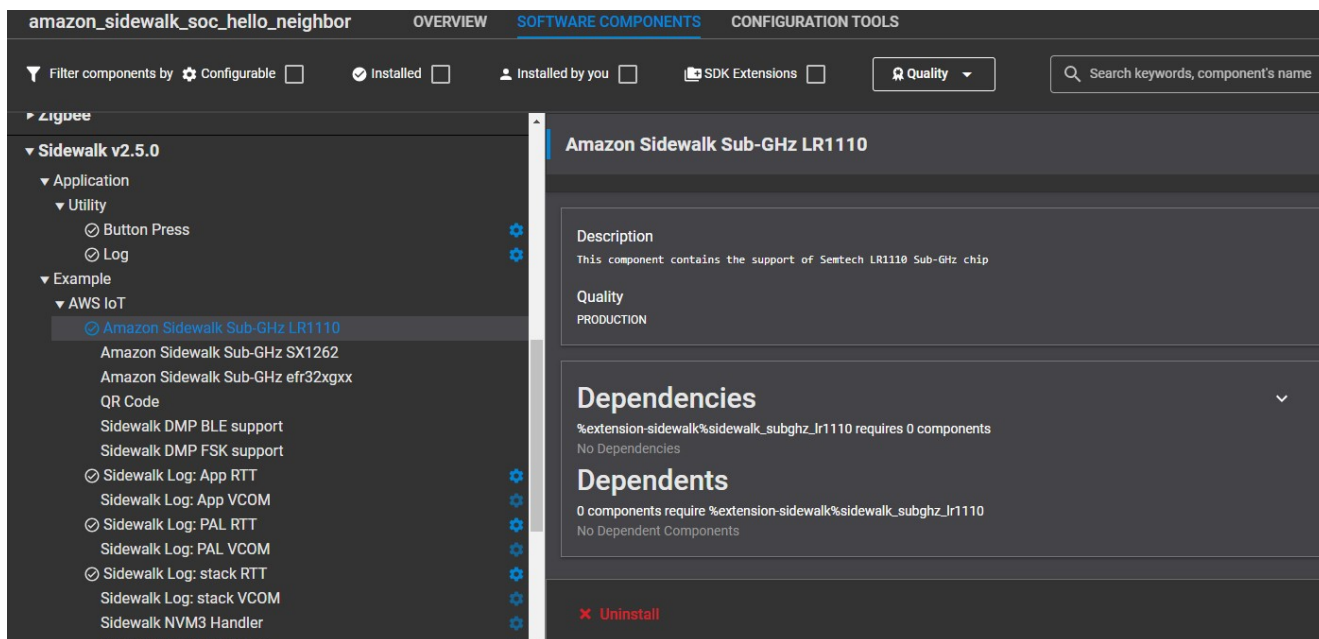
With the Sidewalk resources added to your Simplicity SDK, reopen Simplicity Studio 5. You can now use the graphical interface to create a Sidewalk project.

- Mount the EFR32xG24 radio board onto the main board, then connect the assembly to your computer through the USB connector on the main board.
- A new entry should appear in the **Debug Adapters** view.
- Select the board and make sure your Simplicity SDK with Amazon Sidewalk SDK extension installed is selected in the "Preferred SDK" section of the **General Information** card. If the **Secure FW** version depicted on the card is below the minimum SE FW for your target device specified by the [Prerequisites](#) page, click the nearby link to upgrade the SE FW before proceeding.
- Go to the **EXAMPLE PROJECTS & DEMOS** tab.
- Filter the example list by typing *sidewalk* in the "**Filter on Keywords**" field and press **Enter**.
- Click **Create** next to the **Amazon Sidewalk - SoC Hello Neighbor** example.
- In the **New Project Wizard**, select the **Copy contents** radio button, then click **Finish**.

INFO: Simplicity Studio and the Simplicity SDK support multiple toolchains in addition to the default GCC. However, for projects using the Amazon Sidewalk SDK extension, GCC is required.

For the examples leveraging a Semtech transceiver, SX1262 is used by default. It can be changed to LR1110 with a few simple steps:

1. Update the LR1110 firmware:
 - You will need the LR1110 transceiver firmware version 0x0401 downloadable [here](#).
 - Using the STM32 Nucleo board (recommended), follow the procedure [here](#).
2. In your Sidewalk application, open the the SLCP file and select the Software component panel.
3. Install the LR1110 driver as shown in the picture below. This software component will replace the one for SX1262.



Compile and Flash the Project

Simplicity Studio adds the project to the workspace folder. You can now compile and flash the project on the EFR32xG24 radio board.

- In the Simplicity IDE perspective, select the Sidewalk project (.slcp file).
- On the top menu click **Run**, then select **Debug**.
- Wait for the build and flash operations to succeed.
- In the **Debug** perspective, click **Run**.
- Select **Resume**.

INFO: It is a "best practice" to erase the main flash before writing new application firmware to your device. Simplicity Studio provides many ways to do so, including Simplicity Commander and the [Flash Programmer](#).

WARNING: If using a radio board from the Silicon Labs Pro Kit for Amazon Sidewalk, the out-of-the-box (OOB) demo relies on device-specific credentials pre-flashed to the USERDATA page on your device. This page is not affected by mass-erase operations, but can be explicitly erased by a targeted page erase. [Credentials Backup/Restore Feature](#) describes a process by which you can restore a working out-of-the-box demo application after a mass-erase. However, care should be taken to NOT perform a page erase of USERDATA, or the OOB demo cannot be restored.

View the Application Logs through J-Link RTT

The UART interface is not always available to report traces from Sidewalk example applications. Instead, the applications leverage the J-Link RTT interface. You can easily switch between J-Link RTT and UART interface. A procedure is available in the [Developer's Guide](#).

⚠ WARNING ⚠: J-Link RTT and Simplicity Studio use the same channel to communicate with the board. If you do not see logs in J-Link RTT, try closing and re-opening Simplicity Studio to reset the connection. RTT Viewer needs to be disconnected from a device to allow flashing again with commander or Simplicity Studio.

In the Amazon Sidewalk sample application, the RTT interface is split between 3 channels:

- Terminal 0 - App: contains the application logs
- Terminal 1 - Stack: contains the logs from the Amazon Sidewalk precompiled binaries
- Terminal 2 - Pal: contains the logs from Physical Abstraction Layer (PAL) available in source in the extension

To set up the communication between your PC and the EFR32, follow these instructions:

1. Install the [J-Link RTT Viewer](#).
2. Open the J-Link RTT Viewer.
3. In the **Configuration** panel, **Connection to J-Link** section, select **USB**.
4. In the **Specify Target Device** list, select the connected part. For the EFR32xG24 radio board BRD4187C, select EFR32MG24Axxx1536.
5. In the **Target Interface & Speed** panel, select **SWD** and **4000 kHz**.
6. In the **RTT Control Block** panel, select **Auto Detection**.
7. Click **OK**.

① INFO ①: For the Quectel KG100S module, select EFR32BG21Bxxx1024 as the target device.

A terminal opens and the Sidewalk application traces are output as shown (Terminal 2):

```
[00000006] <l> kvs> kv store opened with 1 object(s)
[00000007] <l> swi> interrupt init ok
[00000008] <l> mfg> mfg store opened with 0 object(s)
```

The above console log indicates that the example application is running, but found no objects in the device's NVM3 flash area. For any Sidewalk application (including this one) to run properly, Sidewalk resources in the cloud must be prepared. Move on to the next section, [Provision your Amazon Sidewalk Device](#), to provision your device and prepare your Amazon Sidewalk cloud resources to interact with it.

If you encounter any issues during this Getting Started procedure, see our [troubleshooting page](#) with solutions to common problems.

Provision your Device

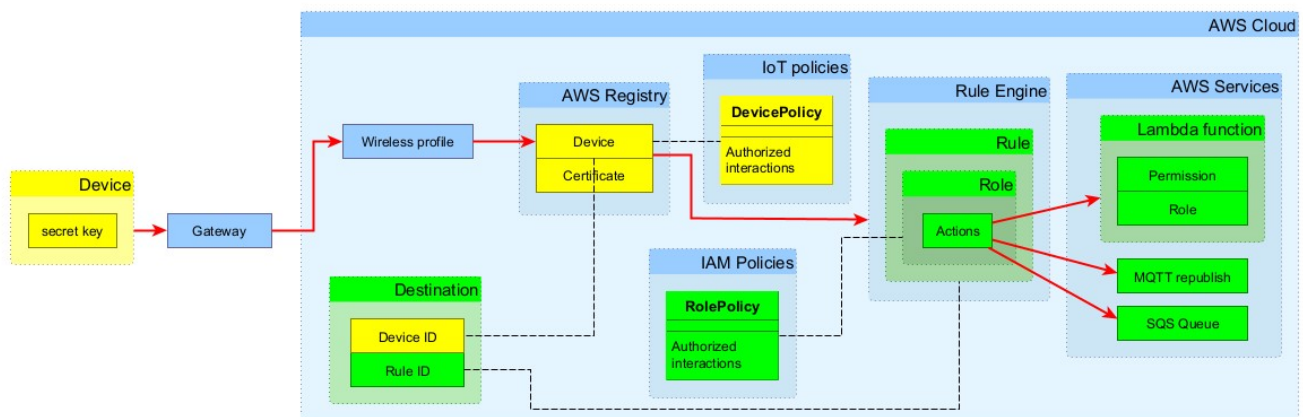
Provision your Amazon Sidewalk Device

Firmware running on an embedded device forms only part of an Amazon Sidewalk solution. Cloud resources must be created to recognize and process data moving through the network between AWS Services and your endpoint. Credentials must be generated and written to the device so that it is accepted and granted access to the Sidewalk network by in-range gateways.

AWS CloudFormation

AWS CloudFormation is a service that helps you model and set up your AWS resources so that you can spend less time managing those resources and more time focusing on your applications that run in AWS. You create a template that describes all the AWS resources that you want (like Amazon EC2 instances or Amazon RDS DB instances), and CloudFormation takes care of provisioning and configuring those resources for you. You do not need to individually create and configure AWS resources and figure out dependencies, because CloudFormation handles that on your behalf.

The Hello Neighbor example depends on AWS resources created using AWS CloudFormation. In the graphic below, green indicates the resources created by the CloudFormation script and yellow indicates the resources strictly associated with a device. The red arrows show the flow of information before an action can be performed on the cloud side.



Even with helpful task aggregator tools like CloudFormation, many steps are involved with the process to create and interlink these resources. Silicon Labs provides Sidewalk developers with a tool to rapidly accomplish these tasks for our example applications.

Sidewalk Assistant

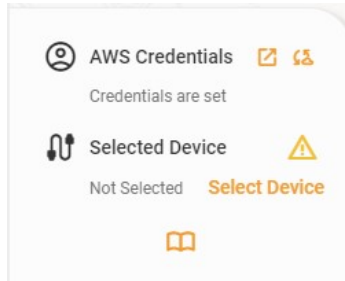
After [creating your example project](#), when your new .slcp project file opens, an additional tab **sidewalk.asconf** containing the **Sidewalk Assistant** tool also appears. If you had closed your project tabs, you can reopen them by going to the root of the project folder in Project Explorer view, and double-clicking the .slcp and .asconf files.

The Sidewalk Assistant tool abstracts multiple underlying scripted interactions via AWS CLI (or manual interactions with the AWS Management Console) into an intuitive graphical user interface that vastly simplifies these operations. Using this tool allows you to focus on the high-level results of your development, rather than the details of each step. You can delve deeper into these details later, when you are ready to move beyond the simplistic AWS constructs of our example applications.

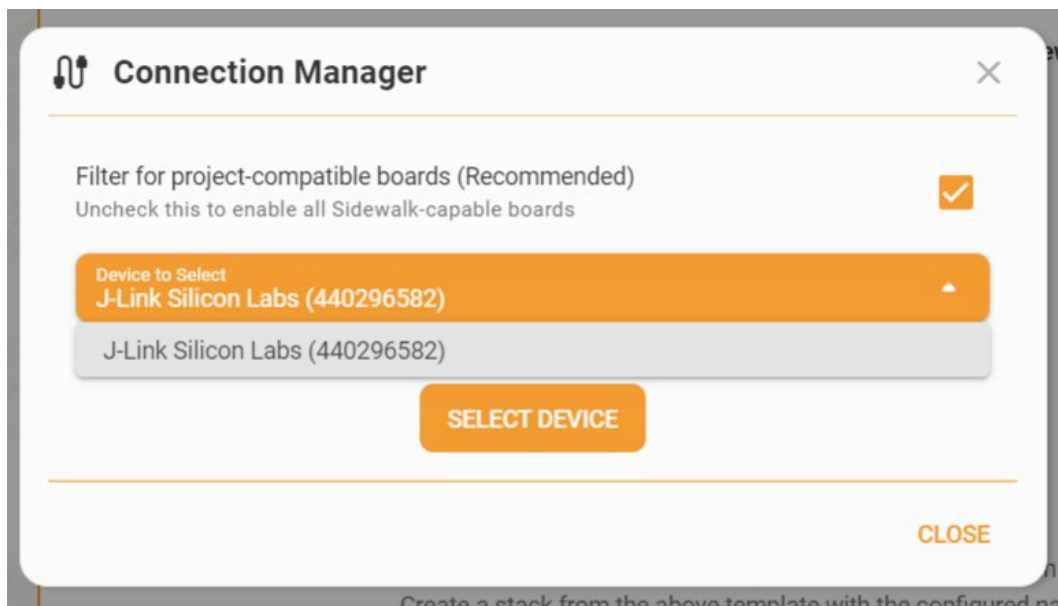
Sidewalk Assistant: AWS Credentials & Selected Device

The `sidewalk.asconf` file (in `/config/sidewalk/` within the project directory) opens in the Sidewalk Assistant on project creation, or you can access the tool by navigating to and opening the `.asconf` file.

The tool automatically scans your system for appropriate AWS credentials, established when AWS CLI was installed and AWS accounts created previously, and indicates that no target device is yet selected:



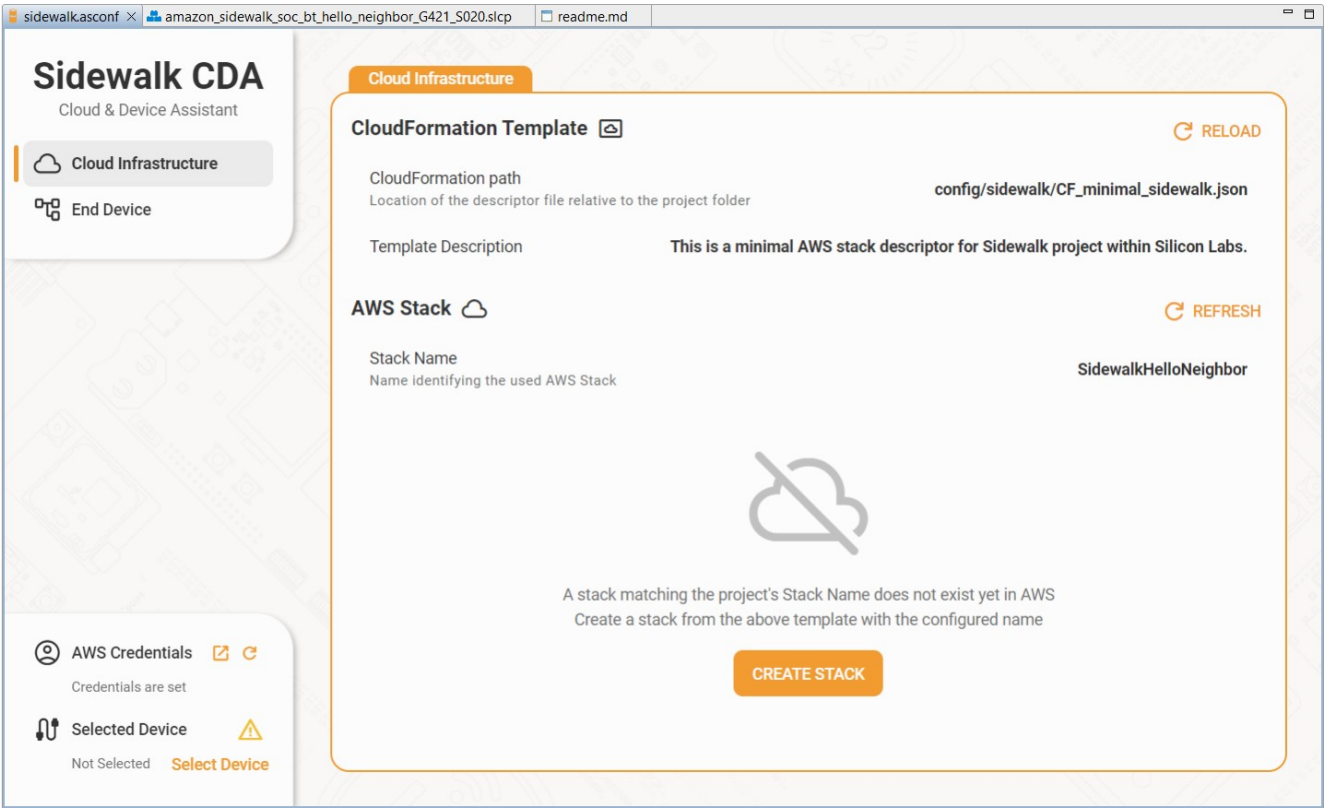
Click **Select Device** to open the Connection Manager dialog and choose from any of the attached devices (the checkbox at top right filters the list to those devices with Sidewalk support).



The Sidewalk Assistant contains two primary pages, the Cloud Infrastructure page and the End Device page.

Sidewalk Assistant: Cloud Infrastructure


Initially, for a new project, the Sidewalk Assistant Cloud Infrastructure page may look like this:



The CloudFormation Template used by this example is described in the top section. The AWS Stack name defined in the project is also visible.

Clicking **Create Stack** triggers the creation of the stack using AWS CloudFormation. The interface displays *Creation in Progress* during the process. You may have to click the **Refresh Data** control multiple times until the results are ready. When stack creation is successfully completed, the AWS Stack section now displays details about the new stack and a **DELETE STACK** control.

Cloud Infrastructure

CloudFormation Template 

RELOAD


CloudFormation path

Location of the descriptor file relative to the project folder

config/sidewalk/CF_minimal_sidewalk.json

Template Description

This is a minimal AWS stack descriptor for Sidewalk project within Silicon Labs.

AWS Stack 

REFRESH

Stack Name

Name identifying the used AWS Stack

SidewalkHelloNeighbor

Status

Current status of the stack

CREATE_COMPLETE

Stack ID

Unique identifier of the stack

arn:aws:cloudformation:us-east-1:894735695240:stack/SidewalkHelloNeighbor/df0df600-bd79-11ed-a870-12d72e0630d7

Creation Time

The time at which this stack was originally created

Wed Mar 08 2023 00:24:25 GMT-0600 (Central Standard Time)

Last Update Time

Indicates the last time the stack was modified

Stack was not modified yet

DELETE STACK

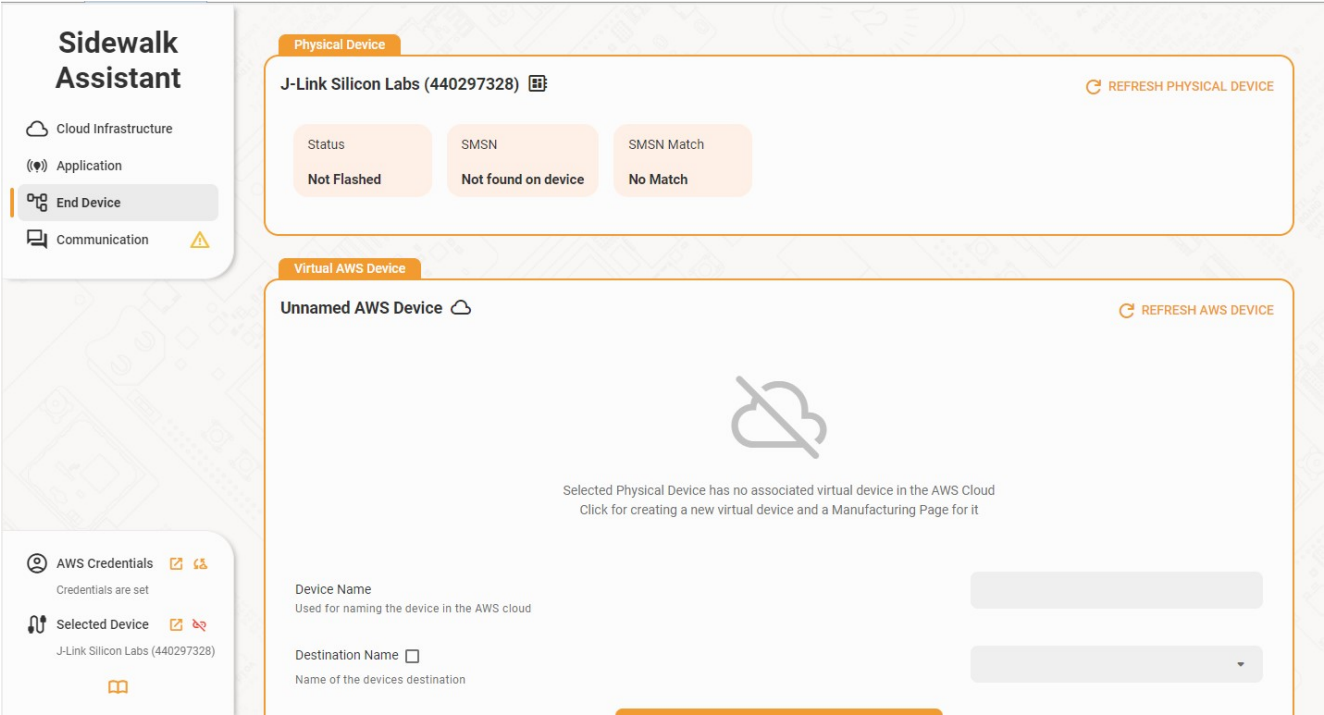
AWS Stack data refreshed

Sidewalk Assistant: End Device

INFO: Instead of using the Sidewalk Assistant, Silicon Labs also provides scripts to provision a device. You can find the documentation to use the script [here](#).

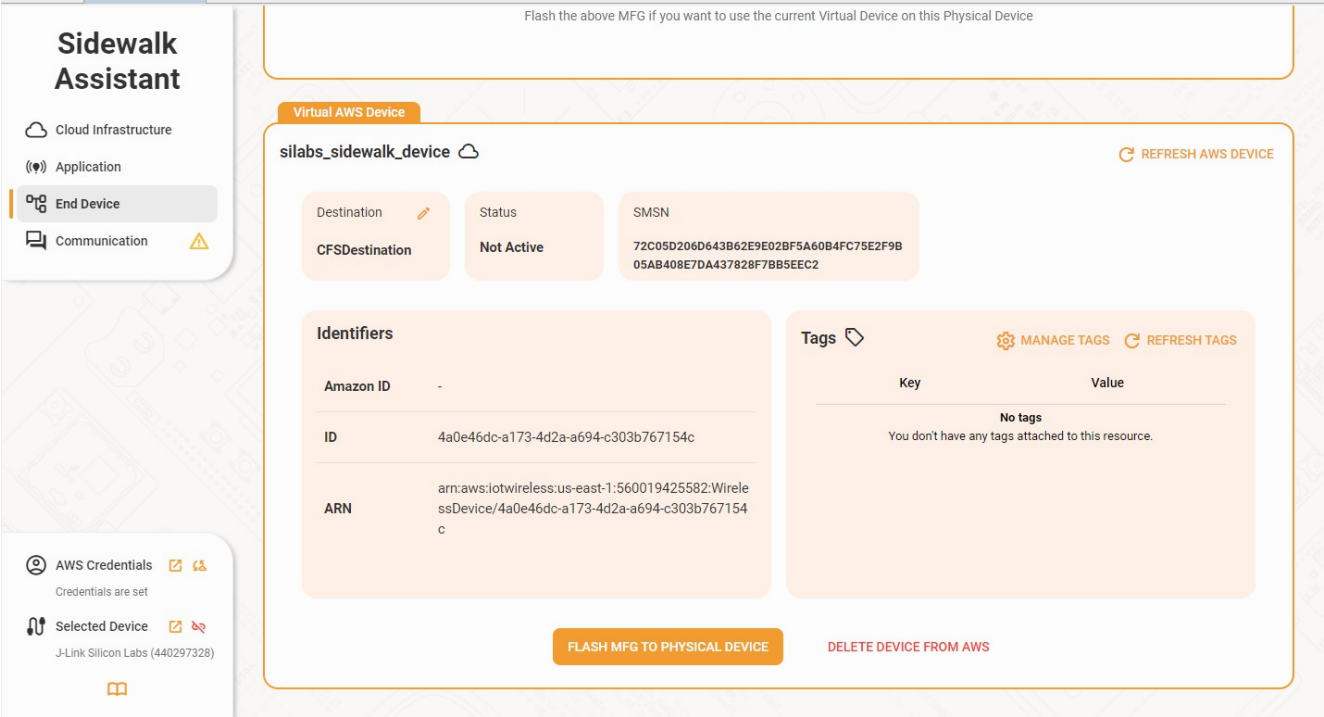
The cloud must be provided with a virtual representation of your device, and a "manufacturing page" must be created from the AWS resources that will equip the endpoint with necessary cryptographic resources to successfully authenticate and encrypt/decrypt traffic to/from your AWS resources in the Sidewalk network.

The manufacturing page is a `.s37` binary file containing the keys and certificates for your device that will be flashed to a specific address. (`.s37` files contain the program bytes and the target address.) To let the Sidewalk Assistant design your virtual device in the cloud and generate a manufacturing page for your physical endpoint, click the **CREATE AWS DEVICE AND MANUFACTURING PAGE** button.



When this first step is complete, the Virtual AWS Device section displays details of the newly-created virtual device.

The GUI also now provides a **FLASH TO DEVICE** control. Click this to flash the manufacturing page to your endpoint. This complements the application image already flashed to the device in a previous step.



If successful, the End Device page should now display details for both the virtual and physical devices.

Physical Device

J-Link Silicon Labs (440297328)

REFRESH PHYSICAL DEVICE

Status	SMSN	SMSN Match
Not Flashed	F59FCC9074F5ABA84164625AB9BEB9BE4 9F0F479AEAB36313211A99501B0D627	No Match

Check the Device Registration and Time Sync

Recall that in [Create and Compile your Sidewalk Application](#), you saw the following log from your device (Terminal 2):

```
02> [00000006] <I> pal kv-storage: kv store opened with 0 object(s)
02> [00000008] <W> pal mfg: no objects found in mfg store
```

Now that you used the Sidewalk Assistant to flash the manufacturing page, if you return to the RTT Viewer (you may need to reset the device and/or reconnect the RTT Viewer), you will see the following log from your device (Terminal 2):

```
02> [00000006] <I> pal kv-storage: kv store opened with 0 object(s)
02> [00000007] <I> pal swi: interrupt init ok
02> [00000008] <I> pal mfg: mfg store opened with 36 object(s)
...
```

When the application is up and running, the logs in terminal 0 are as follows:

```
[00000001] <I> hello neighbor application started
[00000001] <I> sidewalk stack version 1.17.1-13
[00000002] <I> silabs sidewalk extension version 2.2.1
[00000004] <I> sidewalk SMSN: DAF0XXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXEA
[00000005] <I> sidewalk ID:
[00000009] <I> mfg page valid
[00000009] <I> platform initialized
[00000011] <I> main task created
> [00000012] <I> BLE link supported
[00000012] <I> FSK link supported
[00000013] <I> CSS link supported
[00000013] <I> Secure Vault is disabled
[00000055] <I> sidewalk initialized, link mask: 1
[00000056] <I> sidewalk connection policy set, policy: 2
[00000057] <I> sidewalk multi-link policy set, policy: 4
[00000076] <I> sidewalk status not ready
[00000077] <I> registration status: 1, time sync: 1, link: 0
[00000079] <I> sidewalk started, link mask: 1
[00000080] <I> main task started
```

In this log snippet, we can find several useful information:

- The application name: **hello neighbor**
- The Amazon Sidewalk stack version: **1.17.1-13**
- The Sidewalk extension version: **2.2.1**
- The Amazon Sidewalk device SMSN: **DAF0XXXEA**
- The Amazon Sidewalk ID: **BFFFXXXXXX**
- The supported radio links in Amazon Sidewalk context: **BLE link supported**, **FSK link supported** and **CSS link supported**
- The status line: **registration status: 1, time sync: 1, link: 0**

The status line is very important and is used to know about the device status in the Amazon Sidewalk network:

- **Registration status**

Registration status indicates if the device is registered on Amazon's backend or not.

- 0: the device is registered
- 1: the device is not registered

- **Time sync status**

Time sync status indicates if the device has synchronized its current time with the gateway or not. You need to be time-synchronized in order to send and receive messages.

- 0: time is synchronized
- 1: time is not synchronized

- **Link status**

Link status displays the currently active communication channel.

- 0: No connection
- 1: BLE
- 2: FSK
- 4: CSS

For your device to communicate properly, registration status and time sync must be 0. If the endpoint is in-range of a Sidewalk gateway, this process (registration, if a new device, followed by successful time synchronization) should occur within a few minutes. Continue to [Interacting with the Cloud](#) to see how to fully exercise this example application in both directions across the Sidewalk network.

Interacting with the Cloud

Interacting with the Cloud

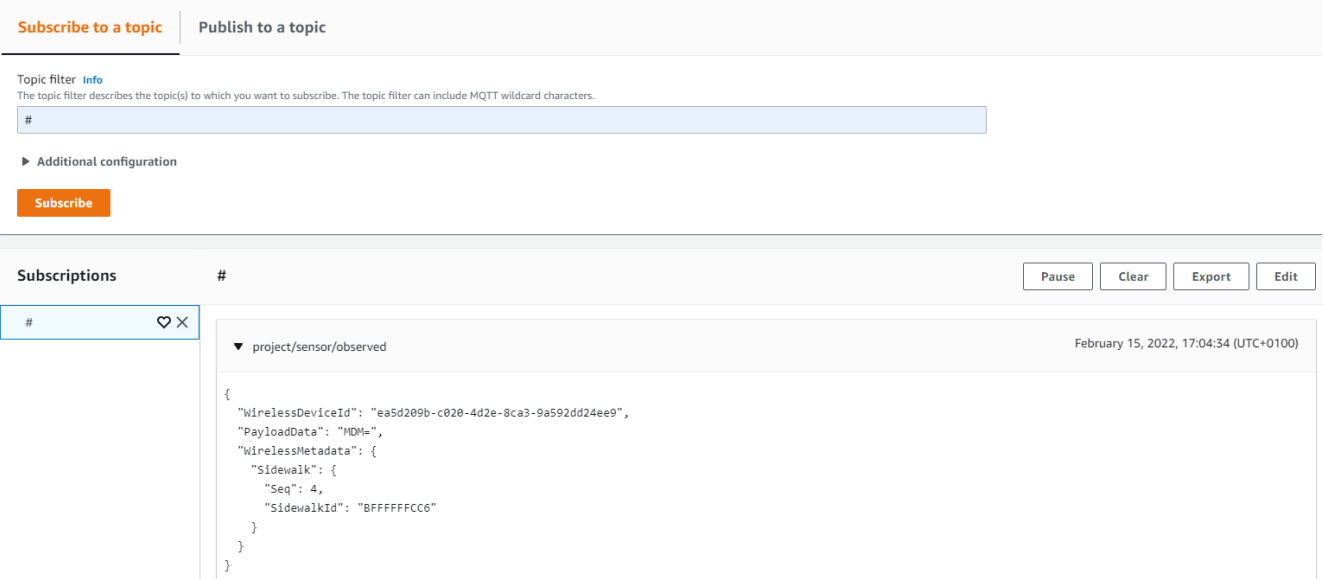
The Sidewalk solution allows an endpoint to natively exchange data with the AWS cloud. Before trying to send and receive message, you should verify that your endpoint achieved time sync. You should see the following line in your device logs.

```
[00482267] <|> registration status: 0, time sync: 0, link: 0
```

Send Data

First, the endpoint sends data to the AWS cloud platform. An AWS Rule reroutes the data to an MQTT topic.

1. Go to the [MQTT test client](#) in AWS.
2. Type “#” in the **Topic filter** field.
3. Click **Subscribe**.
4. Press the main board's button (PB1 for KG100S and PB0 for all other boards). It triggers the Sidewalk endpoint to send a counter value to the AWS IoT Core. For more information on button and CLI action, see the [readme](#) of the **Amazon Sidewalk - SoC Hello Neighbor** sample application.



The screenshot shows the AWS MQTT console interface. At the top, there are tabs for 'Subscribe to a topic' and 'Publish to a topic'. Under 'Subscribe to a topic', the 'Topic filter' field contains '#'. Below this, there is a 'Subscribe' button. The 'Subscriptions' section shows a list of subscriptions, with the first one being '#'. To the right of the subscriptions list, there are buttons for 'Pause', 'Clear', 'Export', and 'Edit'. The message details for the selected subscription are shown in a large text area, displaying a JSON payload: { "WirelessDeviceId": "ea5d209b-c020-4d2e-8ca3-9a592dd24ee9", "PayloadData": "HDM=", "WirelessMetadata": { "Sidewalk": { "Seq": 4, "SidewalkId": "BFFFFFFCC6" } } }. The timestamp of the message is February 15, 2022, 17:04:34 (UTC+0100).

An MQTT message appears in the AWS MQTT console.

The application logs (Terminal 0) are as follows:

```
[01736985] <I> counter update event
[01736985] <I> sending counter update, counter: 0
[01737009] <I> message queued
[01737009] <I> link type: 7ffffff, msg id: 1, msg size: 10, msg type: 2, ack requested: 1, ttl: 120, max retry: 5, additional attr: 0
[01737011] <I> 30 00 00 00 00 00 00 00
[01737012] <I> 00 00
[01737012] <I> ASCII decoded message: 0
[01745316] <I> uplink message sent
[01745317] <I> link type: 2, msg id: 1, msg type: 2
[01749138] <I> downlink message received
[01749139] <I> link type: 2, msg id: 1, msg size: 0, msg type: 3, ack requested: 0, is ack: 1, is duplicate: 0, rssi: -26, snr: 64
```

The message is first queued with parameters. The message is sent when the line **uplink message sent** is displayed and, with default parameters, an acknowledge message is send back by the cloud as show by **downlink message received** and the message being of size 0 (**msg size: 0**).

Receive Data

The Hello Neighbor applications don't support the [MQTT test client](#) publish feature to send message from the cloud to the endpoint. For this step, you will need [AWS CLI configured](#) and associated with your IAM user. Your user should at least have the following policies: *AdministratorAccess* and *AWSIoTWirelessDataAccess*. If your user is *Administrator*, you don't need to do anything. You can check your users in the [IAM Identity Center](#).

You are now ready to receive data on your Sidewalk endpoint. Run the following command in a terminal:

```
aws iotwireless send-data-to-wireless-device --id=[Wireless-Device-ID] --transmit-mode 0 --payload-data="SGVsbG8gIENBWTaWRld2FsayE=" --
wireless-metadata "Sidewalk={Seq=1}"
```

Where:

- **Seq=x**, x should be unique any time you run the command.
- **--id** should be the "Device ID" of the device in the AWS portal IoT Core -> Wireless Connectivity -> Devices -> Sidewalk.
- **--payload-data** is the message to send in binary format (in this case, "Hello Sidewalk!").

You should see "Hello Sidewalk!" on the EFR32 log console.

The application logs (Terminal 0) are as follows:

```
[01957048] <I> downlink message received
[01957049] <I> link type: 2, msg id: 5, msg size: 17, msg type: 2, ack requested: 0, is ack: 0, is duplicate: 0, rssi: -25, snr: 65
[01957051] <I> 48 65 6C 6C 6F 20 20 20
[01957052] <I> 53 69 64 65 77 61 6C 6B
[01957052] <I> 21
[01957052] <I> ASCII decoded message: Hello Sidewalk!
```

Protocol Overview

Amazon Sidewalk Protocol Overview

This section presents some of Amazon Sidewalk's main concepts, such as registration and sub-GHz configurations. Reference Amazon documentation is linked at the bottom of this page.

- [Frustration Free Networking](#): Explanation of device registration to Amazon Sidewalk network.
- [FSK Configuration](#): Short introduction of the protocol using FSK radio layer.
- [CSS Configuration](#): Short introduction of the protocol using CSS radio layer.
- [Multi-link and Auto Connect](#): Short introduction to the multi-link and auto connect features.

Useful links to Amazon documentation:

- [Introduction to Amazon Sidewalk](#)
- [Protocol Specification](#)
- [Multi-link Application Note](#)
- [Sub-GHz Device Profiles Application Note](#)
- [API Reference \(PDF\)](#)

Frustration Free Networking

Frustration Free Networking

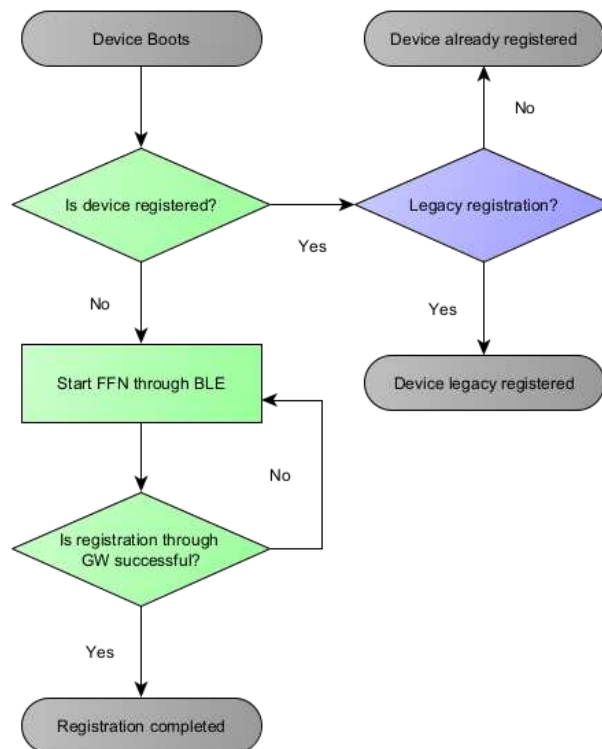
For Sidewalk, the Frustration Free Networking feature allows a device to register to the network without user action. This process is called Touchless Registration. It is an automatic process that occurs between the unregistered Sidewalk endpoint and the Sidewalk gateway over BLE or FSK. This method does not require associating the endpoint with the user's AWS account. To ensure your device registers using Touchless Registration, complete the following:

1. Ensure your gateway is Sidewalk-enabled and in range (the closer, the better).
2. Ensure your endpoint is up and running and open JLinkRTT Viewer for logs.
3. Registration should begin automatically. You should see some log output.
4. Successful registration will be indicated with the log below:

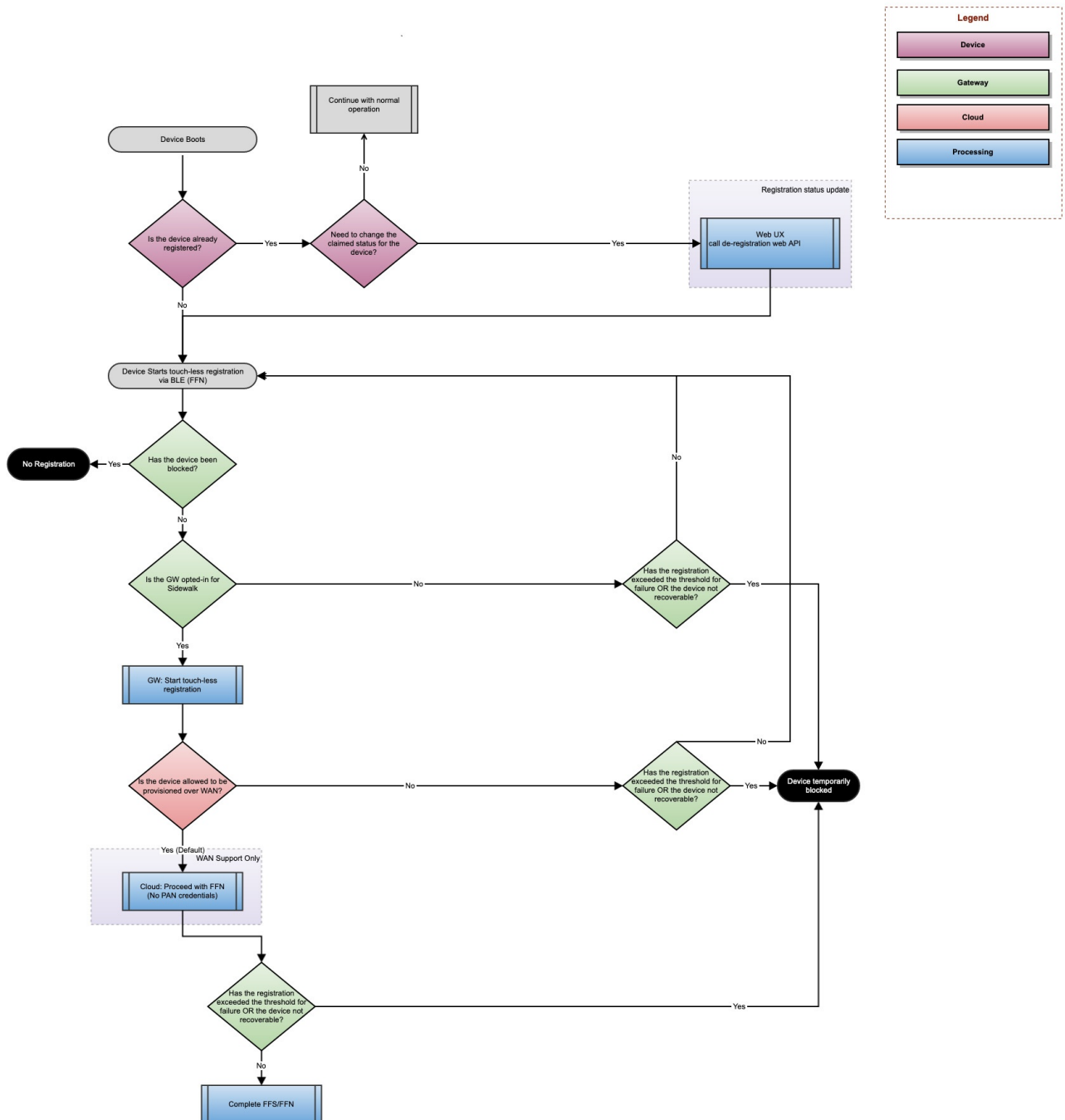
```
<info> app: Registration Status = 0, Time Sync Status = 0 and Link Status = 0
```

Diagram Flow

The following shows the Automatic Touchless Registration simplified flowchart:



If registration is attempted several times in a row and fails each time, the Gateway will block the device for a period of time. If that happens, rebooting the Gateway should reset the list of blocked devices. The following shows a full flowchart including device blocking.



Manual Registration Flow

Manual registration over BLE is an alternative registration mechanism to Automatic Touchless registration. Amazon Sidewalk also supports registration using the [Amazon Sidewalk mobile SDK](#).

Note that this step requires a Mac, Windows or an Ubuntu PC with Bluetooth capability in order to execute the Sidewalk endpoint registration. As an alternative, a Bluetooth USB dongle can be used. Also note that it is good practice to disconnect all your other BLE devices connected to your computer (wireless headset, mouse, keyboard...) during registration.

Create a New Security Profile

1. Go to the [Login with Amazon](#) page.
2. Click **Create a New Security Profile**.

3. Complete the required information: **Security Profile Name**, **Security Profile Description**, and **Consent Privacy Notice URL**. These values can be anything, for example:
 - Security Profile Name: SidewalkSecurityProfile
 - Security Profile Description: SidewalkSecurityProfile
 - Consent Privacy Notice URL: <https://www.silabs.com/>
4. Click **Save**. You should see your new Security profile.
5. Click **Show Client ID and Client Secret**, and note down the **Client ID** and **Client Secret** (or save them as environment variables).
6. Click the gear to the right of your security profile's row.
7. Click **Web Settings**.
8. Click **Edit**, add <http://localhost:8000/> to **Allowed Origins**, and click **Save**.

Obtain an LWA Token

An LWA (Login with Amazon) token is one of the ways to register your Sidewalk endpoint.

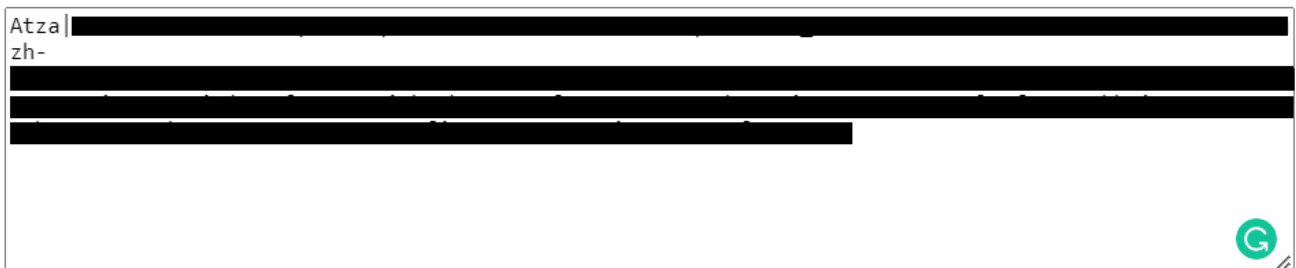
In your Amazon Sidewalk repository clone, go to `/tools/scripts/public/sid_pc_link/apps/device_registration`, and install the module requirements:

```
pip3 install --user -r requirements.txt
```

Execute the following commands in a terminal:

```
python3 main.py -l --client-id <Client ID>
```

A browser window appears and opens a pop-up if allowed. Allow pop-ups on the page if not. The pop-up asks you for Amazon developer credentials. When logged in, the initial window displays an LWA token. The Python script automatically edits the `app_config.json` file with the LWA token. **The standard LWA token is only valid for an hour.**



Sidewalk Endpoint Registration

Open the `app_config.json` file in your Amazon Sidewalk repository folder `/tools/scripts/public/sid_pc_link/apps/device_registration` and edit it as follows.

1. Set the `ENDPOINT_ID` attribute equal to the "smsn" value found in the `certificate_[SIDEWALK_ID].json` file.
2. If you are on Linux, the `BLUETOOTH_ADAPTER` attribute must be set to the Bluetooth dongle that you purchased as part of the setup. The `hcitool` command will read out your Bluetooth interfaces, and will likely be in the form `hciX` (most likely `hci0`, like the default setting in `app_config.json`). If you are on macOS, you do not need to touch this field.
Note: You can leave the rest of the attributes to their default values.
3. Save and close the `app_config.json` file.
4. Run the following command:

```
python3 main.py -r
```


You then see the logs of a series of exchanges on your terminal and on your EFR32 logging interface. After a minute or so, you should see "INFO Device registration succeeded". You have successfully registered your EFR32 device onto the Sidewalk network!

If your device is not detected by the script, it may be because your device connected automatically with the touchless registration using Sidewalk FFN.

If you are using the **Amazon Sidewalk - SoC Bluetooth Hello Neighbor** application, a main board button press is required to connect with the gateway. No button press is needed with the **Amazon Sidewalk - SoC Bluetooth Sub-GHz Hello Neighbor** application.

With the logging console connected, press main board button PB0. You should see the following status message.

```
[I] App - set connection request
[I] BLE State: CONNECTED

[I] App - sidewalk internal event
...
[I] App - sidewalk status changed: 0
[I] App - registration status: 0, time sync status: 0, link status: 1
[I] Delete rx_buffer :: Gateway [f1 76] :: Stream [7] :: Transaction [11]
```

Deregistration

You may want to de-register your endpoint for debug purposes.

⚠ WARNING ⚠: Successful de-registration requires a two-way message exchange between the endpoint, gateway, and cloud. Make sure your device is registered and time-synced with your gateway. You can check with the log output below:

```
<info> app: Registration Status = 0, Time Sync Status = 0 and Link Status = 0
```

To de-register your device run the command below by replacing `YOUR-WIRELESS-DEVICE-ID` with the device ID of the device you want to de-register.

```
aws iotwireless deregister-wireless-device --identifier "YOUR-WIRELESS-DEVICE-ID" --wireless-device-type "Sidewalk"
```

Successful de-registration of the device is indicated by the following log output:

```
<info> app: Registration Status = 1, Time Sync Status = 1 and Link Status = 1
```

Wait a few seconds after the de-registration of your device to see Automatic Touchless registration taking place to register your device.

INFO: When the endpoint is registered but not time-synced and the de-registration API is called, the device is de-registered on the cloud side. The device does not receive the information because it is not time-synced. However, on its request message for time sync, the cloud will issue a message to notify the device it is not registered anymore. Upon receiving this new message, the device will remove its credentials and move to the deregistered state.

FSK Configuration

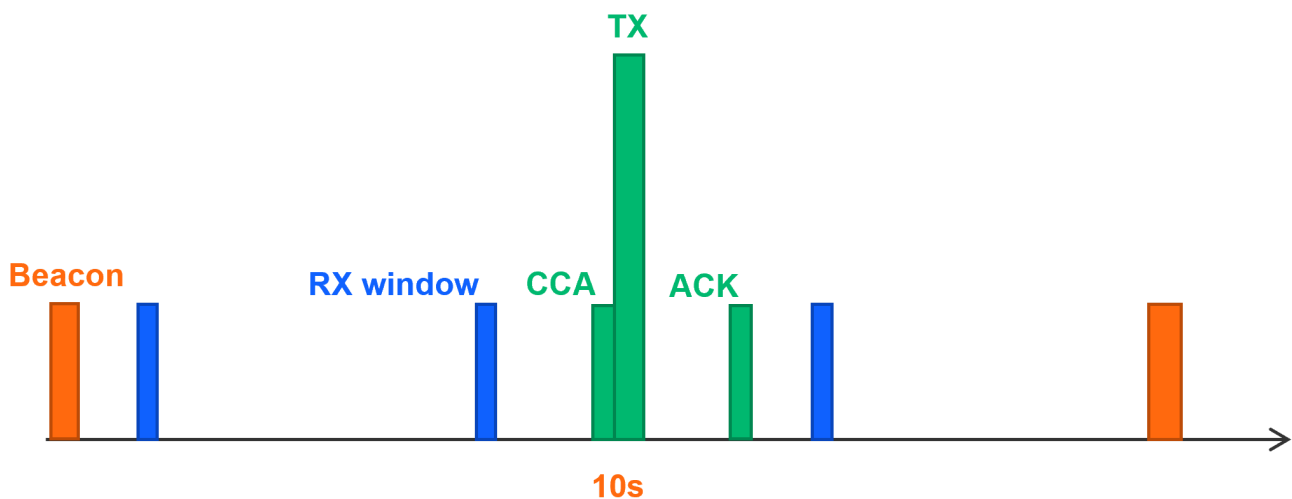
FSK Configuration

FSK means Frequency-Shift Keying. For more information about the FSK radio layer and its power profiles, see the "Sub-GHz Protocol Stack" > "Amazon Sidewalk Endpoint connection modes" > "SubG-FSK connection profiles" section in the [Amazon Sidewalk specification](#).

Typical Behavior

FSK is a synchronous protocol, which means the endpoint keeps the connection with the gateway alive using beacons. Beacons are sent by the gateway every 10 seconds. Messages can be sent and received in between beacons.

By default the connection is kept alive and there are 3 RX opportunities every 10 seconds. Transmissions are preceded by a Clear Channel Assessment (CCA) and followed by an Acknowledgment message (ACK). See the figure below for an example:



When the endpoint boots, it connects to the gateway using discovery; the endpoint listens on every channel to detect the gateway preamble. Upon detection of the preamble, the endpoint retrieves its first beacon and starts communicating with the gateway (touchless registration, configuration negotiation, and time synchronization).

Power Consumption and Energy Modes

FSK Power Profiles

Two connection profiles are available for FSK referred to as "power profiles" 1 and 2. Power profile 1 consists of smaller messages exchanged during synchronization with the gateway. Protocol parameters are chosen by the gateway. Power profile 2 is a full synchronization with parameters chosen by the endpoint and negotiated with the gateway. In both power profiles it is possible to choose the time separation between two RX windows.

In the Hello Neighbor example application, power profile 2 is the default, with an RX window every 3150 ms.

Possible Parameters

The FSK power profile can be modified by the `sid_option` API call using option `SID_OPTION_900MHZ_SET_DEVICE_PROFILE`. Then the structure `sid_device_profile_unicast_params` can be updated with the following values:

- The device profile ID possible values `SID_LINK2_PROFILE_1` and `SID_LINK2_PROFILE_2` contained in `sid_device_profile_id` enum.
- The number of RX windows is always `SID_RX_WINDOW_CNT_INFINITE` for both power profiles.
- The windows separation can be any of the values of enum `sid_link2_rx_window_separation_ms`.
- The type of event for which the device wakes up values are contained in enum `sid_unicast_wakeup_type`.

Finally, while Profile 1 has infinite windows, the application can still turn the protocol off shortly after transmitting a packet to achieve similar operating characteristics of limited number of RX windows. When the protocol is turned off, the wakeups for beacon and data will not be executed. Whenever uplink data needs to be sent, the application can turn the protocol back on, re-establish the Sidewalk connectivity, transmit its data, and turn it back off. Hence, even though the protocol does not automate this process for the application, similar operating characteristics are still achievable.

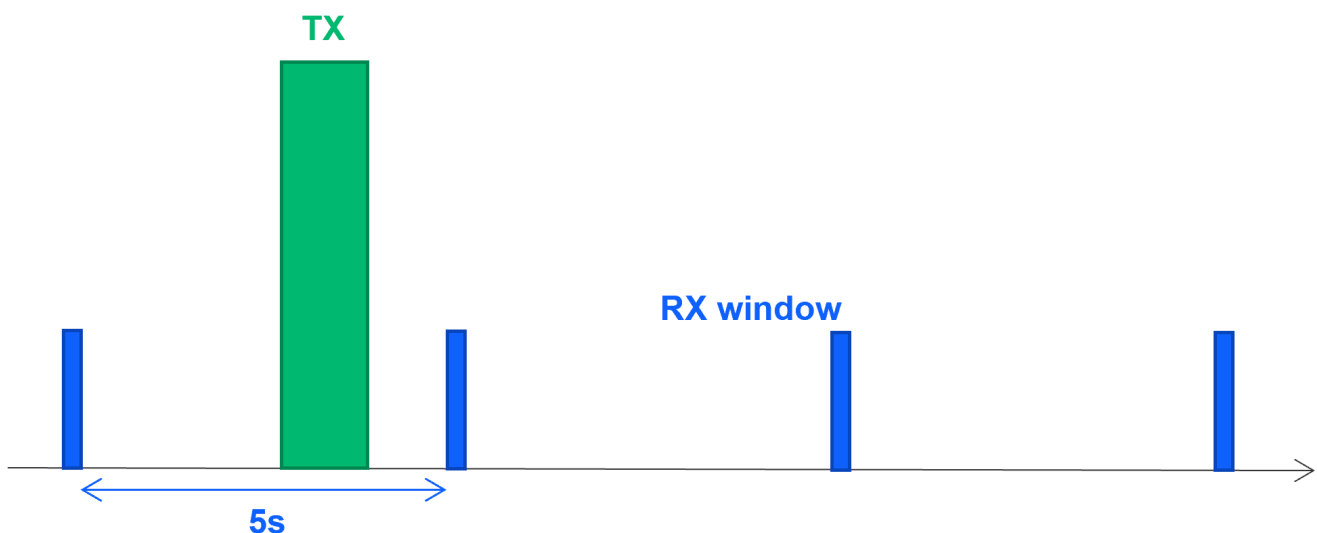
CSS Configuration

CSS Configuration

CSS means Chirp Spread Spectrum. For more information about the CSS radio layer and its power profiles, see the "Sub-GHz Protocol Stack" > "Amazon Sidewalk Endpoint connection modes" > "Asynchronous access mode profiles" section in the [Amazon Sidewalk specification](#).

Typical Behavior

CSS is an asynchronous protocol, which means it does not keep a connection with the gateway. Though transmits are asynchronous, during an active downlink phase listening windows for reception occur every 5 seconds as shown below. These windows either repeat indefinitely or after a transmission for a limited time, depending on the active connection profile (also referred to as "power profiles" A and B).



Power Consumption and Energy Modes

CSS Power Profiles

Power profile A opens a limited number of listening windows after a transmission. Power profile B opens listening windows periodically while also assuring regular transmission to keep the connection alive. In power profile B, the gateway considers the endpoint inactive after 5 minutes without transmission and stops sending messages. To prevent this, the Sidewalk stack implements a keep-alive mechanism for power profile B that creates a power consumption overhead not present in power profile A. There is a trade-off between this additional overhead and lower latency, as in power profile A a device must re-perform time synchronization before transmission if too much time has elapsed since the last transmit. Power profile B is the default connection profile in the Sidewalk stack.

Possible Parameters

The CSS power profile can be modified by the `sid_option` API call using option `SID_OPTION_900MHZ_SET_DEVICE_PROFILE`. Then the structure `sid_device_profile_unicast_params` can be updated with the following values:

- The device profile ID possible values `SID_LINK3_PROFILE_A` and `SID_LINK3_PROFILE_B` contained in `sid_device_profile_id` enum.
- The number of RX windows is either `SID_RX_WINDOW_CNT_INFINITE` for power profile B and any of the other values in enum `sid_rx_window_count` for power profile A.
- The windows separation can be any of the values of enum `sid_link3_rx_window_separation_ms`.
- Values for the type of event for which the device wakes up are contained in enum `sid_unicast_wakeup_type`.

Multi-link and Auto Connect

Multi-link and Auto Connect

Since the release of Sidewalk SDK 1.16, a feature called Multi-link has been introduced. This feature abstracts the process of connection establishment and maintenance for the radio links supported by the Sidewalk stack, making it easier for developers. It offers developers varying degrees of flexibility to control not only the connection behavior of the links but also the transfer of messages over them.

You can find the [dedicated application note on Multi-link](#) on the Amazon website.

The multi-link feature introduces a connection policy framework that affects how connections are established and how messages are sent using the supported radio links. It also impacts the uplink message attributes and the link's connection timeout parameters. Downlink messages from the Sidewalk cloud services always follow the link on which the last uplink was received from the endpoint. Note that the link connection policy configuration does not persist across reboots.

The Multi-link feature is divided into two policies: multi-link and auto connect. Auto connect provides developers with the flexibility to configure various link connection attributes, such as link priority and the maximum number of retry attempts for a link. In contrast, multi-link determines the uplink attributes (like cloud ACK per message, the number of transmission retries for a message, and the time to live for a message) and the link connection attributes by selecting a mode that enforces a set of parameters. Developers are expected to choose a mode rather than individual attributes.

Message Uplink Attributes	Description
ACK	Request Acknowledgment from AWS IoT Wireless managed service
Number of retries	Number of times the stack can retry sending this message
Time To Live (TTL)	Time before the message expires in the queue

Link Connection Control Attributes	Description
Link Priority	Defines the order of priority to attempt connection on a link range from 0 to 2, 0 being the highest
Link Timeout	Maximum Time to Attempt Connection for a Link in seconds

The feature relies on the connected state mode of a link to determine if a link is available to send a message or not. The connected state is defined as follows: The link must be started, registered, and time-synced. Additionally, for BLE, it must be connected to the gateway. FSK is always connected as long as it receives beacons; and CSS is connectionless, so it is always considered connected.

Multi-link

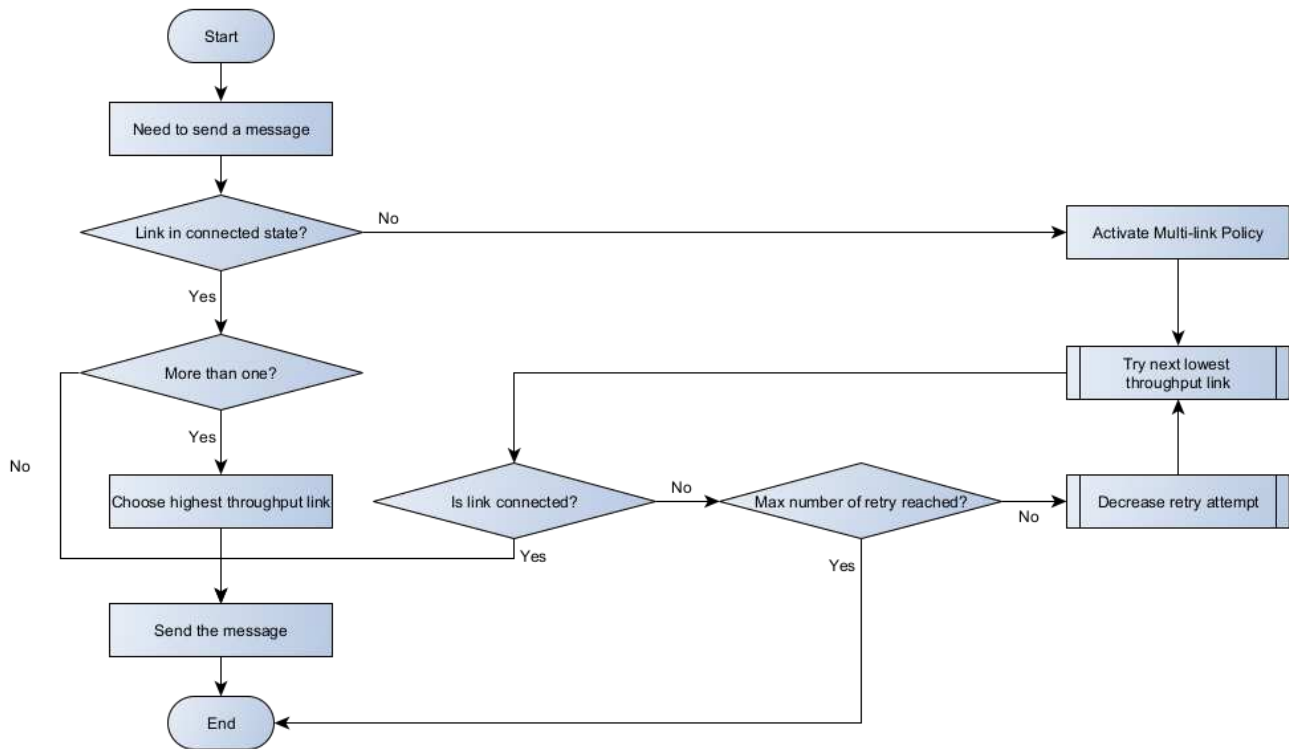
With multi-link, the system determines the message uplink and link connection attributes, so developers do not need to configure these parameters when sending a message. The Sidewalk stack initiates a connection on a link only when a message needs to be transmitted.

The algorithm functions as follows: If the Sidewalk stack has at least one link in a connected state and the message to be sent does not exceed the MTU of that link, the message is transmitted.

- If multiple links are in a connected state, the message transmission attributes of the link with the higher throughput are applied to the enqueued message (e.g., if both BLE and FSK are connected, BLE is used).
- If no link is in a connected state, the message transmission attributes of the link with the lowest throughput are applied to the enqueued message (e.g., if BLE, FSK, and CSS are not connected, CSS is used).
- If the Sidewalk stack has at least one link in a connected state that can send the message, the link auto connection policy algorithm is not triggered, as the connected link can successfully transmit the message.

However, if the Sidewalk stack has no link in a connected state ready to send the message, the multi-link policy will be activated. This policy will then cycle through the links in the priority order set by the policy table (from lowest to highest throughput) multiple times until all the messages queued by the user are sent or expired. Once there are no messages in the endpoint Amazon Sidewalk stack's send queue, the algorithm will stop attempting to establish a connection (time sync) with the Amazon Sidewalk network.

For background connection maintenance, when the stack is configured with multi-link policies for latency or performance, the multi-link algorithm attempts to maintain a connection (uplink active) with the Sidewalk cloud services via the BLE link. This connection maintenance is achieved by updating the advertisement payload. For other policies, the algorithm maintains the default behavior of the link connection policy.

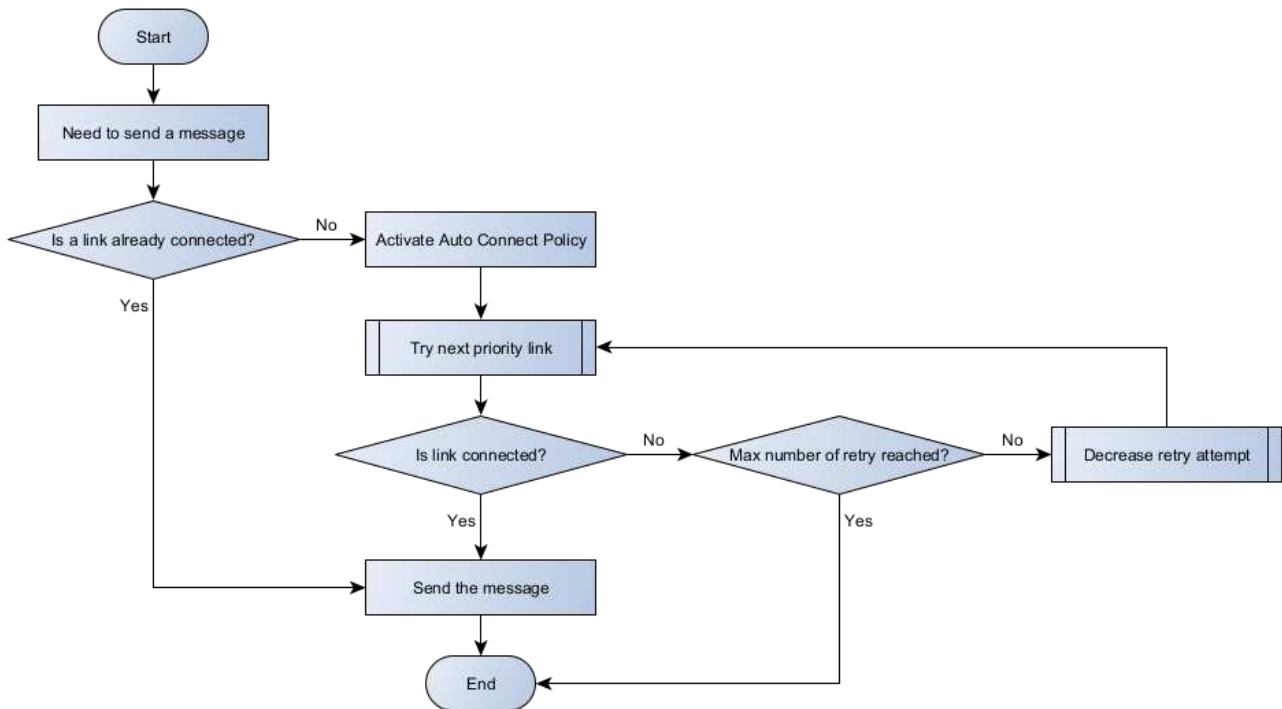


The multi-link mode can be set using the `sid_option` API call using `SID_OPTION_SET_LINK_CONNECTION_POLICY` or `SID_OPTION_GET_LINK_CONNECTION_POLICY`. The values for the Message Uplink Attributes and Link Connection Control Attributes are determined by the chosen policy mode. You can see details of the policy modes in the [dedicated application note on Multi-link](#).

Auto Connect

Auto connect policy offers developers the flexibility to configure the link connection control attributes. The Sidewalk stack initiates a connection on a link only when a message needs to be transmitted, with no background connection maintenance.

The algorithm operates as follows: If the Sidewalk stack has at least one link in a connected state that can send the message, the link auto connection policy algorithm will not be triggered, as the connected link can successfully transmit the message. However, if the Sidewalk stack has no link in a connected state ready to send the message, the auto connect policy will be activated. This policy will then cycle through the links in the priority order set by the developer multiple times until all the messages queued by the user are sent or expired. Once there are no messages in the endpoint Amazon Sidewalk stack's send queue, the algorithm will cease attempting to establish a connection (time sync) with the Amazon Sidewalk network.



The auto connect policy can be set using the `sid_option` API call using `SID_OPTION_SET_LINK_CONNECTION_POLICY` or `SID_OPTION_GET_LINK_CONNECTION_POLICY` and parameters can be chosen using `SID_OPTION_SET_LINK_POLICY_AUTO_CONNECT_PARAMS` or `SID_OPTION_GET_LINK_POLICY_AUTO_CONNECT_PARAMS`. The values for the Link Connection Control Attributes can be configured with Auto Connect. You can see details of the possible values in the [dedicated application note on Multi-link](#).

Overview

Silicon Labs Developer's Guide for Amazon Sidewalk

This developer's guide presents all the concepts needed to develop your own application leveraging Amazon Sidewalk.

- **Stack Structure:** Presentation of Silicon Labs' solution integrating Amazon Sidewalk.
- **Application Development:** Instructions and documentation to allow you to develop your own application including automating device creation, and Secure Element and memory usage.
- Amazon Sidewalk API: Refer to the [Amazon Sidewalk Sid API Developer Guide](#).
- **Testing and Debugging:** Solutions for debugging including logging.
- **Power Consumption Analysis:** Introduction to Amazon Sidewalk power consumption.
- **Performance:** Details on range testing, power consumption, and latency.
- **Multiprotocol with Sidewalk:** Instructions to add standard BLE advertising alongside your Amazon Sidewalk application.
- **Troubleshooting:** Troubleshooting guide with all common issues.

You can find additional more advanced sample applications on the [Silicon Labs Amazon Sidewalk Applications Github repository](#).

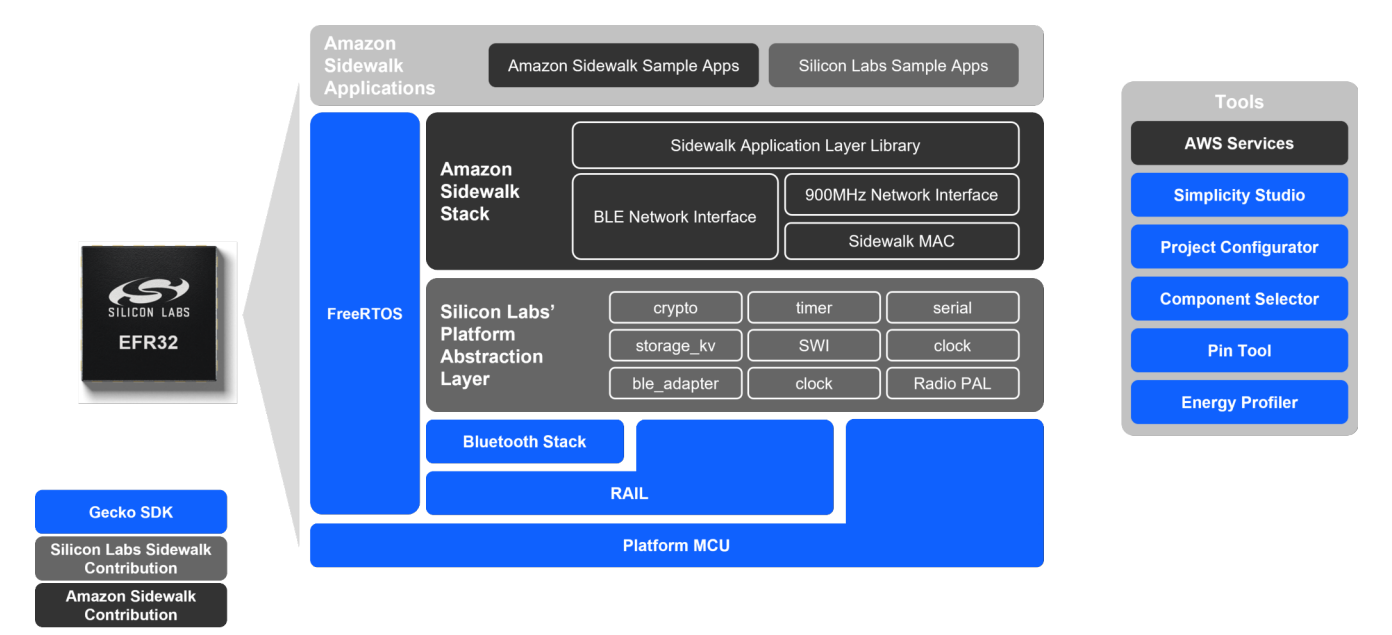
Useful links to Amazon documentation:

- [Amazon Sidewalk General Documentation](#)
- [API Reference \(PDF\)](#)
- [AWS API Reference for Amazon Sidewalk](#)
- [Multi-link Application Note](#)
- [Sub-GHz Device Profiles Application Note](#)
- [Amazon Sidewalk Requirements](#)
- [Sidewalk Qualification](#)
- [Mobile SDK Developer Guide](#)
- [Manufacturing](#)

Stack Structure

Stack Structure

The following figure describes the firmware structure that integrates the Amazon Sidewalk protocol stack and enables Amazon Sidewalk connectivity for the end device. The Platform Abstraction Layer assures communication between Silicon Labs Simplicity SDK and the Amazon Sidewalk protocol stack. The developer creates applications on top of the stack, which Silicon Labs provides as a precompiled library.



- The Amazon Sidewalk stack contains the following blocks:
- Application Layer Library: Contains API call definitions to be used in application development.
 - BLE Network Interface: Contains Amazon Sidewalk-specific configuration for the Bluetooth LE (BLE) link layer.
 - Sub-GHz Network Interface: Contains Amazon Sidewalk-specific configuration for sub-GHz link layer and Semtech drivers.
 - MAC Layer: Amazon Sidewalk implementation of the MAC layer.

Dependencies

- The following software components are used by the stack:
- FreeRTOS: Used as a base to run the stack.
 - PSA crypto: Used for all cryptographic operation.
 - BLE stack: Used to access the BLE stack.
 - SPI driver: Used only in sub-GHz-enabled applications, for communication with a third-party sub-GHz radio module.
 - RAIL: Used to access the radio and high-precision timers.

Library Files

Amazon Sidewalk extension contains three library files: one for the Sidewalk stack, one for the BLE stack and the last one for the Semtech chip driver. The Silicon Labs Platform Abstraction Layer is available in source in the extension.

Software Components

The Amazon Sidewalk extension includes the following software components. Each component can be used in a custom application. Each is described and documented in the following sections.

- Sidewalk
 - Protocol
 - [Amazon Sidewalk](#): the Amazon Sidewalk stack base component
 - [Sidewalk NVM3 Migrator](#): a tool to migrate to the new NVM3 architecture starting Sidewalk 2.0.0
- Utility
 - [Logs](#): the component that allows to control the logs level and output
 - [Sidewalk Application Message Structure](#): an example of lightweight message structure for Sidewalk messages
 - [Sidewalk utils](#): a toolkit with useful functions to retrieve information on the device (extract SMSN from manufacturing page for example)

Amazon Sidewalk

Sidewalk NVM3 Migrator

This component is responsible for automatically handling the changes in the NVM3 structure around the Sidewalk SDK 1.16 release (Sidewalk extension 2.0.0). More precisely, it migrates NVM3 data from the previous unversioned Silabs-specific structure to a new structure which also got a version number, namely 1.0.0.0. This includes migration of manufacturing page (MFG) and Key-Value storage data (the KV storage handles the state of the Sidewalk stack) and other Silabs-specific NVM3 data as well.

The goal is to provide seamless transition to the new structure by simply adding this component, and calling its single API function.

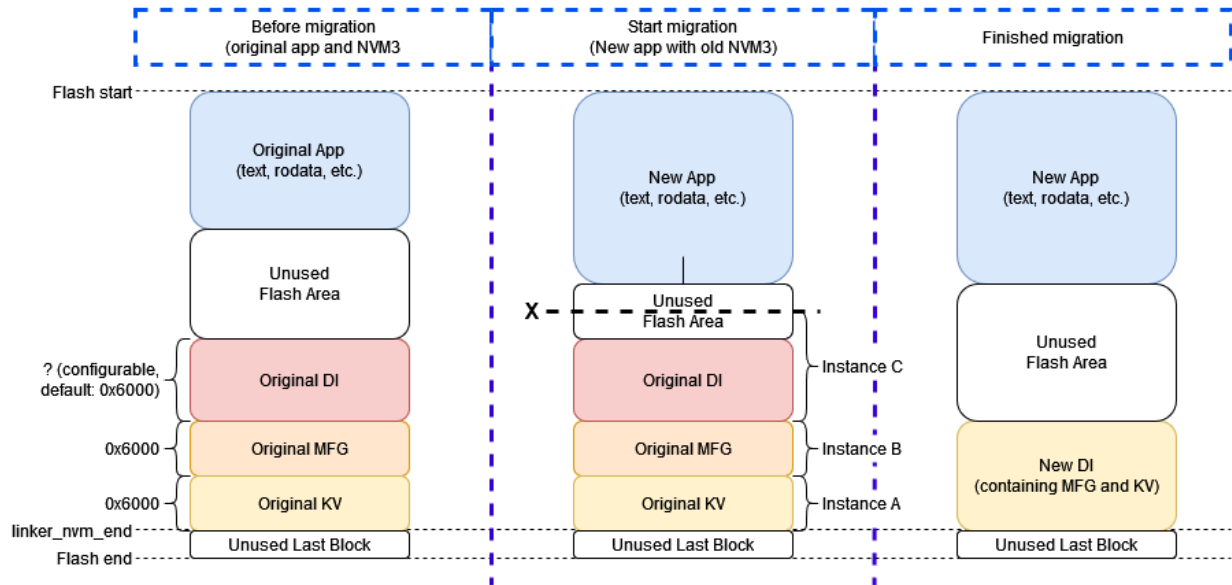
For more information about the latest NVM3 organization, refer to [Non-Volatile Memory Management in Sidewalk Context](#).

You can also refer to the [NVM3 Migrator component readme](#).

The rework of the NVM3 organization has two important advantages:

- Reduce the flash usage to store the same information.
- The address of the reserved space for the default NVM3 instance can be easily changed, thus changing MFG and KV storage address seamlessly.

The following schema shows the main stages of the memory migration.



1. Before migration, we see the assumed original NVM3 structure.
 - NVM3 data is placed at the end of the flash.
 - The MFG and KV instances are 0x6000 large.
 - The size of the original NVM3 Default instance is also 0x6000 by default.
2. When we start migration we expect that the user flashed a new application containing the migrator component, and left the NVM3 data intact.
 - The new application shall not overlap with the original NVM3 data.
3. After the migration we see the new NVM3 structure containing MFG and KV storage data.

The full migration process is shown in the following flow diagram:

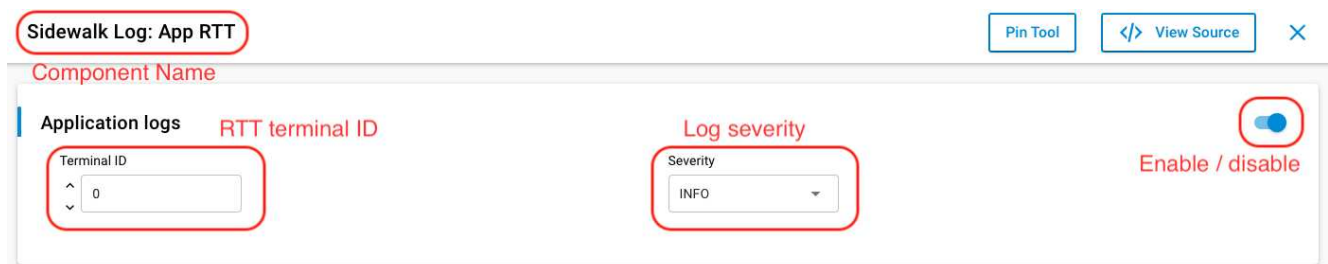
Logs

The purpose of this group of software components is to facilitate the redirection of app, stack, or PAL logs to VCOM or separate RTT terminals. It consists of six components, all dependent on the **Log** component and available in both RTT and UART versions: **Sidewalk Log: App**, **Sidewalk Log: PAL**, and **Sidewalk Log: stack**.

There are three distinct layers (app, stack, and PAL) whose log configurations can be managed independently. This per-layer configuration includes the following three aspects:

- Enabling or disabling logs
- Changing the severity level
- Changing the RTT terminal ID (specific to logs redirected to the RTT interface)

At least one software component (RTT or VCOM) per layer must be selected. To completely disable the logs, the chosen component can be configured accordingly. By default, all applications use the RTT interface to display logs.



To display logs or switch the log output from RTT to UART, refer to [Testing and Debugging documentation](#).

Sidewalk Application Message Structure

The Sidewalk Application Message component is a software module designed to facilitate the transfer of commands and associated data across the Sidewalk network with minimal metadata. It operates on a TLV (tag-length-value) data protocol, which simplifies the process of sending commands with associated data, making it somewhat analogous to the RPC protocol in functionality.

This component's role is to package the data intended for transmission with essential metadata (such as protocol version, command class, command ID, payload length, etc.). The application is then responsible for sending this packaged data as a Sidewalk application message through the Sidewalk network. The message structure aims to be as lightweight as possible and only adds 4 bytes of metadata to the actual message data.

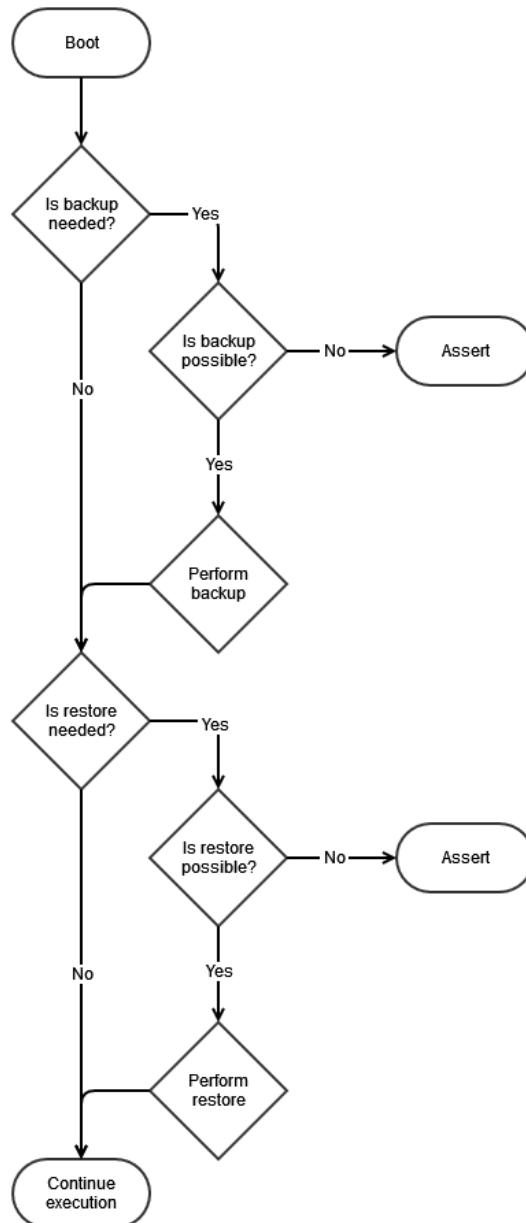
For instance, consider the process of sending a notification to the cloud application when a user presses a button. The application will invoke the necessary functions to encapsulate the button's state and assign the appropriate metadata for the button press event. After the application data is wrapped in the data protocol, it is transmitted over the Sidewalk network by the application.

Credentials Backup/Restore Feature

The out-of-the-box (OOB) application provided with the Pro Kit has a feature that allows users to erase the flash memory without losing the device private keys and other device-specific information. The feature relies on the user data partition that is not affected by mass-erase but it can still be erased. This feature can be used alongside Secure Vault seamlessly.

The manufacturing page needs to persist in order for the application to work in the Amazon Sidewalk network. This data can be divided in two groups, one for device-specific information and the other for information common to all devices created under the same device profile. As the capacity of the user data partition is quite limited, the common information is stored in the application code and only the device-specific information is backed up on the user data partition.

The following flow diagram shows the backup/restore feature workflow.



The Backup feature is used to copy device-specific information from the manufacturing page onto the user data partition. There is no need to copy common information as it is hard-coded in the application. Backup takes place when the application cannot detect any backed up data on the user data partition. This implies that, if the user data partition is erased but not the application, then the application will perform a backup process after each boot.

The Restore feature is used to copy device-specific information from the user data partition and common information from the OOB application code onto the manufacturing page. Restore takes place when the application cannot detect any valid manufacturing page. This implies that, if the device is mass-erased, then the application will perform a restore process after each boot and restore the whole manufacturing page.

Application Development Walkthrough

Application Development Walkthrough

The purpose of this document is to guide you through using and configuring the Amazon Sidewalk solution and implementing an Amazon Sidewalk application. Before going through this document, you should go through the Getting Started to familiarize yourself with the Amazon Sidewalk flow.

Amazon Sidewalk - SoC Empty Sample Application

Create and Compile the Application

You can create and compile the Amazon Sidewalk Empty sample application just like any other Amazon Sidewalk application. This bare application only initializes the Amazon Sidewalk Stack.

Provision your Device using Scripts

To automate device creation for prototyping, scripts are available to create the necessary objects in the cloud and generate the manufacturing page. These scripts are provided in the Silicon Labs extension for Amazon Sidewalk [Github repository](#). To set up a prototype device and register it to the network, the following steps are necessary:

1. Deploy your application in AWS.
2. Compile and flash an application for your embedded device.
3. Create a device profile in AWS.
4. Create a wireless device in AWS.
5. Generate a manufacturing page.
6. Flash the manufacturing page to your embedded device.

You can deploy your AWS application automatically using dedicated tools like Amazon CloudFormation. You can compile and flash your application using Simplicity Studio 5. To create the device profile, wireless device, and manufacturing page you can use [Silicon Labs' prototyping script](#).

Remember to install the requirements listed in the `requirements.txt` file under `amazon_dependencies` with `pip3 install -r requirements.txt`.

The script configuration is handled by a JSON file like the following:

```
{
  "awsAccount": {
    "awsRegion": "us-east-1",
    "awsProfile": "default"
  },
  "commanderPath": "C:\\SiliconLabs\\SimplicityStudio\\v5_4\\developer\\adapter_packs\\commander\\commander.exe",
  "generationRequests": [
    {
      "deviceProfileId": null,
      "deviceName": null,
      "destinationName": "CFSDestination",
      "targetPart": null,
      "quantity": 1
    }
  ]
}
```

An example configuration file is provided at the root of the repository.

The first block `awsAccount` is used to configure the connection to AWS cloud:

- `awsRegion` : The AWS region you would like to add your devices to (Note that only the `us-east-1` region is supported at the moment).
- `awsProfile` : Used to choose the AWS CLI profile linking to the correct credentials. If you do not use profiles, you can leave the default.

The second block defines the toolchain used to create the manufacturing page:

- `commanderPath` : Should link to the the Simplicity Commander executable file.

The third block `generationRequests` controls the target device details:

- `deviceProfileId` : If this is your first time running the script, it should be empty. If not, it contains your Device Profile ID (this field is filled automatically).
- `deviceName` : This is optional, you can choose to give a custom name to your device.
- `destinationName` : Should be the name of the destination used by your AWS cloud application.
- `targetPart` : The OPN of the target part. If empty, manufacturing pages for all supported radio boards will be generated.
- `quantity` : The quantity of devices you want to create (default value is 1).

On the first run, the script creates a prototyping device profile and fills the `deviceProfileId` with the resulting device ID. All subsequent wireless devices will be created using this device profile. Note that during prototyping, a device profile can be linked to a maximum of 1000 wireless devices.

To start the script, execute the following:

```
python3 generate_prototype.py --input <.> --output <output_folder> --config example_config.json
```

The script creates a directory structure as follows in the chosen output folder under `mfg_output` :

```
DeviceProfile_7c51bc6b-0556-2083-6f0d-aeb750c94508
├─ DeviceProfile.json
└─ WirelessDevice_db57b1c9-22ad-c052-07ae-1e1cae9bb384
   └─ SiLabs_MFG.nvm3
      └─ SiLabs_xG21.s37
         └─ SiLabs_xG23.s37
            └─ SiLabs_xG24.s37
               └─ SiLabs_xG28.s37
                  └─ WirelessDevice.json
```

The first directory is named with the device id of your device profile and contains `DeviceProfile.json` , which contains your device profile information. Then a directory is created for every wireless device you created. One wireless device contains manufacturing pages for all supported platforms and a `WirelessDevice.json` file containing your device information (including private keys).

To use the script with command-line arguments instead of a configuration file, simply drop the `--config` parameter.

```
usage: generate_prototype.py [-h] -in INPUT -out OUTPUT [-p AWS_PROFILE] [-n NAME] [-d DST_NAME] [-t TARGET] [-c COMMANDER] [-cfg CONFIG]
```

optional arguments:

```
-h, --help            show this help message and exit
-in INPUT, --input INPUT
                        Path of the input directory
-out OUTPUT, --output OUTPUT
                        Path of the output directory
-p AWS_PROFILE, --aws-profile AWS_PROFILE
                        Name of your AWS profile from .aws/credentials (default: default)
-n NAME, --name NAME  Specified name for the newly created device (default: sidewalk_[user]_device)
-d DST_NAME, --dst-name DST_NAME
                        Destination name used for uplink traffic routing (default: CFSDestination)
-t TARGET, --target TARGET
                        Target part number for which the MFG page generation has to be done (default: all)
-c COMMANDER, --commander COMMANDER
                        Path to commander executable, not needed if already in system PATH
-cfg CONFIG, --config CONFIG
                        Configuration file, if provided other arguments are ignored
```

You can directly flash the generated .s37 file using Simplicity Commander.

Provision your Device with Secure Vault

INFO: This is an optional step, you can decide to try this feature or not.

On Silicon Labs EFR32 Series 2 platforms, you can leverage Secure Vault to store sensitive data (private keys) in a secure place. Amazon Sidewalk can leverage the Secure Vault to store the device private keys. Silicon Labs provides scripts to provision a device directly using Secure Vault for the manufacturing and during prototyping phase: [Provision your prototyping device with Secure Vault](#).

From initializing to Sending a Message Through Sidewalk

Note that in Amazon Sidewalk, you can add the support for an external radio for the FSK and CSS modulation. For instructions on configuring Sidewalk stack to use the external radio transceiver, refer to the [dedicated section](#).

Initialize Sidewalk Protocol Stack

You can find an example of Sidewalk initialization in the `main_thread` function in `app_process.c`.

To initiate a link with Amazon Sidewalk, it is essential to first populate the `sid_config` structure with the necessary configuration details. This structure is defined as follows:

```
struct sid_config {
    uint32_t link_mask;
    struct sid_event_callbacks *callbacks;
    const struct sid_ble_link_config *link_config;
    const struct sid_sub_ghz_links_config *sub_ghz_link_config;
};
```

The `link_mask` in the `sid_config` structure specifies which Sidewalk radio links the device will set up and use. For instance, to configure a single link, you would assign `SID_LINK_TYPE_1` to `link_mask`. For multiple links, you would combine them using the bitwise OR operator, such as `SID_LINK_TYPE_1 | SID_LINK_TYPE_2 | SID_LINK_TYPE_3`. The enumeration `sid_link_type` list possible values that can be combine for the `link_mask`.

```
enum sid_link_type {
    /** Bluetooth Low Energy link */
    SID_LINK_TYPE_1 = 1 << 0,
    /** 900 MHz link for FSK */
    SID_LINK_TYPE_2 = 1 << 1,
    /** 900 MHz link for LORA */
    SID_LINK_TYPE_3 = 1 << 2,
    /** Any Link Type */
    SID_LINK_TYPE_ANY = INT_MAX,
};
```

The `callbacks` member holds the event callbacks for the selected link, which must be stored in static const storage since the Sidewalk stack accesses this member without making a copy.

For BLE link-specific configuration, a pointer to `struct sid_ble_link_config` is passed. Similarly, the `sub_ghz_link_config` is used for configuring the Sub-GHz link.

Once the `sid_config` structure is correctly filled out, the `sid_init` function can be called with this configuration to initiate the link. The function must be called only once for a given link type unless `sid_deinit()` is called to deinitialize it.

```
sid_error_t sid_init(const struct sid_config *config, struct sid_handle **handle);
```

This initialization process is crucial as it sets up the foundational parameters for the Sidewalk stack to operate correctly. It ensures that the device is ready to start communication over the Sidewalk network according to the specified configurations. Remember, proper initialization must precede any attempt to start a link using `sid_start`. This structured approach to initialization and starting links ensures a robust and error-free operation of the Amazon Sidewalk stack.

You can find an example code to initialize Amazon Sidewalk stack in the `main_thread` function of `app_process.c`. Below is also a small example. the `sub_ghz_link_config` can be removed if the application should only support BLE radio.

```
// Set configuration parameters
struct sid_config config =
{
    .link_mask = SID_LINK_TYPE_1 | SID_LINK_TYPE_2, //BLE and FSK
    .callbacks = &event_callbacks,
    .link_config = &ble_config,
    .sub_ghz_link_config = &sub_ghz_link_config,
};

struct sid_handle *sidewalk_handle = NULL;

sid_error_t ret = sid_init(&config, &sidewalk_handle);
if (ret != SID_ERROR_NONE) {
    SL_SID_LOG_APP_ERROR("sidewalk initialization failed, link mask: %x, error: %d", (int) config.link_mask, (int) ret);
    goto error;
}
```

The `ble_config` parameter controls the configuration of the BLE stack, including the name of your device, the advertising and connection parameters and the output power. The `sub_ghz_link_config` field controls the Sub-GHz radio configuration, including maximum output power and registration over Sub-GHz enablement.

The details for the `event_callbacks` structure will be elaborated upon in the subsequent sections.

Start and Stop the Sidewalk Protocol Stack

Once the platform for Amazon Sidewalk is initialized, the Sidewalk stack can be activated using the `sid_start` API. This action is contingent upon the prior initialization of the link. Thanks to the multi-link and auto connect features, multiple links can be initiated simultaneously, but only a single link will be used for uplink and downlink at a given moment. The stack can be started by the `main_thread` function in `app_process.c` of the Amazon Sidewalk Empty application.

```
sid_error_t sid_start(struct sid_handle *handle, uint32_t link_mask);
```

The `sid_start` function is versatile, allowing for the initiation of one or multiple links simultaneously. For a single link, you would set `link_mask` to the specific link type, such as `SID_LINK_TYPE_1`. For multiple links, you would use the bitwise OR operator to combine them, like `SID_LINK_TYPE_1 | SID_LINK_TYPE_3`.

However, it's crucial to remember that you can only start a link type that was previously initialized with `sid_init`.

In summary, `sid_start` is the function that activates the Sidewalk stack, enabling it to perform its intended tasks and ensuring that the device is ready for communication over the Sidewalk network.

Here is an example code to start the Amazon Sidewalk stack:

```
uint32_t link_mask = SID_LINK_TYPE_1 | SID_LINK_TYPE_2; // BLE and FSK

sid_error_t ret = sid_start(sidewalk_handle, link_mask);
if (ret != SID_ERROR_NONE) {
    SL_SID_LOG_APP_ERROR("sidewalk start failed, link mask: %x, error: %d", (int)config.link_mask, (int)ret);
    goto error;
}
```

To halt the operations of the Sidewalk stack, use the `sid_stop` function. When this function is called, the stack will cease to send or receive messages, and all notifications will be suspended. The link status will be updated to reflect a disconnected state, and the time synchronization status will be preserved for future use.

```
sid_error_t sid_stop(struct sid_handle *handle, uint32_t link_mask);
```

The function allows for stopping either a single link or multiple links at once. For example, to stop a single link, set `link_mask` to `SID_LINK_TYPE_1`. To stop multiple links, combine them using the bitwise OR operator, like `SID_LINK_TYPE_1 | SID_LINK_TYPE_3`.

It's important to note that `sid_stop` can only be used to stop links that were previously initialized with `sid_init`. This ensures that the stack is properly configured and that the stop operation is performed on an active link.

In essence, `sid_stop` is a control function that provides the ability to gracefully shut down the Sidewalk stack's operations, ensuring that the device can safely transition to a non-operational state while preserving essential status information for future operations.

Here is an example code to stop the Amazon Sidewalk stack:

```
uint32_t link_mask = SID_LINK_TYPE_1 | SID_LINK_TYPE_2;

sid_error_t ret = sid_stop(sidewalk_handle, link_mask);
if (ret != SID_ERROR_NONE) {
    SL_SID_LOG_APP_ERROR("failed to stop the stack: %d", (int)ret);
    return false;
}
```

Amazon Sidewalk and RTOS

The Amazon Sidewalk stack operates on a real-time OS (FreeRTOS). To manage events and ensure the stack remains operational, you need to implement a loop in the main thread. In the `main_thread` function, there's already a `while(1)` loop that can be utilized for this purpose. Essentially, you need to handle events as they occur. The primary event to keep the stack running is `EVENT_TYPE_SIDEWALK`, and you must process this event to maintain the stack's operation. Additional events can be defined as needed for other purposes. A basic structure for the main loop could be as follows:

At the top of `app_process.c`, initialize a few structures:

```
static QueueHandle_t g_event_queue;
// Sidewalk Events
enum event_type{
    EVENT_TYPE_SIDEWALK = 0,
    EVENT_TYPE_SEND_MESSAGE,
    //Add your custom event types here
    EVENT_TYPE_INVALID
};
#define MSG_QUEUE_LEN (10U)
```

Then, `main_thread` will display as follows:

```
g_event_queue = xQueueCreate(MSG_QUEUE_LEN, sizeof(enum event_type));
app_assert(g_event_queue != NULL, "app: queue creation failed");

while (1) {
    // Add your code
    enum event_type event = EVENT_TYPE_INVALID;
    if (xQueueReceive(g_event_queue, &event, portMAX_DELAY) == pdTRUE) {
        switch (event) {
            case EVENT_TYPE_SIDEWALK:
                sid_process(sidewalk_handle);
                break;
            case EVENT_TYPE_SEND_MESSAGE:
                //call to sid_put_msg here
                break;
            default:
                SL_SID_LOG_APP_ERROR("app: unexpected evt: %d", (int)event);
                break;
        }
    }
}
```

Finally the `on_sidewalk_event` callback should be implemented to add the `EVENT_TYPE_SIDEWALK` to the queue.

```
static void on_sidewalk_event(bool in_isr,
                             void *context)
{
    UNUSED(in_isr);
    UNUSED(context);
    queue_event(g_event_queue, EVENT_TYPE_SIDEWALK);
}
```

The `queue_event` function used in this example is the same as the one implemented in Hello Neighbor:

```
static void queue_event(QueueHandle_t queue,
                       enum event_type event)
{
    // Check if queue_event was called from ISR
    if ((bool)xPortIsInsideInterrupt()) {
        BaseType_t task_woken = pdFALSE;

        xQueueSendFromISR(queue, &event, &task_woken);
        portYIELD_FROM_ISR(task_woken);
    } else {
        xQueueSend(queue, &event, 0);
    }
}
```

This consists of the minimum to have the Sidewalk stack running.

Check Sidewalk Protocol Status

To check the current status of the Amazon Sidewalk stack, use the `on_status_changed` or the `sid_get_status` API. Both return detailed information encapsulated within the `sid_status` structure.

Here is the `on_status_changed` callback trace. You can find it in `app_process.c` file of the Amazon Sidewalk Empty sample application:

```
void (*on_status_changed)(const struct sid_status *status, void *context);
```

Here is the `sid_get_status` function trace:

```
sid_error_t sid_get_status(struct sid_handle *handle, struct sid_status *current_status);
```

Here is the `sid_status` structure definition:

```
struct sid_status {
    /** The current state */
    enum sid_state state;
    /** Details of Sidewalk stack status */
    struct sid_status_detail detail;
};
```

To use the function, pass the handle obtained from `sid_init()` and a pointer to a `sid_status` structure where the current status will be stored. If the function succeeds, the `current_status` will contain the latest status information from the Sidewalk library.

The `sid_state` enumeration and the `sid_status_detail` structure provide a comprehensive overview of the current status and state of the Sidewalk stack. Here's an explanation of their components:

The `sid_state` enumeration describes the operational state of the Sidewalk stack:

- `SID_STATE_READY` : Indicates that the Sidewalk stack is operational and ready to send and receive messages.
- `SID_STATE_NOT_READY` : Used when the Sidewalk stack cannot send or receive messages, such as when the device is not registered, the link is disconnected, or time is not synchronized.
- `SID_STATE_ERROR` : Signifies that the Sidewalk stack has encountered an error. In this case, `sid_get_error()` should be called to obtain a diagnostic error code.
- `SID_STATE_SECURE_CHANNEL_READY` : This state means that the Sidewalk stack can send and receive messages with a secure channel established, but the device is not registered, and time is not synchronized.

The `sid_status_detail` structure contains several fields:

- `registration_status` : This indicates the registration status of the Sidewalk device.
- `time_sync_status` : This indicates whether the Sidewalk device has successfully synchronized its time with the Sidewalk network.
- `link_status_mask` : This is a bitmask used to determine which links are currently active. If the bit corresponding to a link is set, that link is up; otherwise, it is down. For example, to check if `SID_LINK_TYPE_1` is up, the expression `!!(link_status_mask & SID_LINK_TYPE_1)` needs to be true.
- `supported_link_modes` : This array holds the supported modes for each link type, where a link type may support more than one mode simultaneously.

These components are crucial to understand the status and state of the Sidewalk stack, as they provide insights into the device's connectivity, registration, and readiness to function within the Sidewalk network.

Here is an example code to check the Amazon Sidewalk stack status:

```
struct sid_status state = {};

sid_error_t ret = sid_get_status(sidwalk_handle, &state);
if(ret != SID_ERROR_NONE) {
    SL_SID_LOG_APP_ERROR("Sidewalk stack is not initialized: %d", (int)ret);
    return false;
}
```

Here is an example of the `on_status_changed` callback implementation to display the Sidewalk status in the logs:

```
static void on_sidewalk_status_changed(const struct sid_status *status,
                                      void *context)
{
    SL_SID_LOG_APP_INFO("app: REG: %u, TIME: %u, LINK: %lu",
        status->detail.registration_status,
        status->detail.time_sync_status,
        status->detail.link_status_mask);
}
```

Send a Message Through Sidewalk

The `sid_put_msg` function is designed to queue a message for transmission in the Sidewalk stack.

```
sid_error_t sid_put_msg(struct sid_handle *handle, const struct sid_msg *msg, struct sid_msg_desc *msg_desc);
```

Here's an explanation of how it works:

When you call `sid_put_msg`, you need to provide it with three parameters:

- `handle`: This is a pointer to the handle that you received when you called `sid_init()`. It's essentially a reference to the initialized Sidewalk stack.
- `msg`: This is the actual message data that you want to send.
- `msg_desc`: This is a message descriptor that the function will fill out. It serves as an identifier for the message you're sending.

In summary, `sid_put_msg` is a function that queues messages for transmission over the Sidewalk network. Here is an example code to send a message over Amazon Sidewalk:

```
static void send_sidewalk_message(struct sid_handle *app_context)
{
    char message_buff[15] = "Hello Sidewalk!";

    struct sid_msg msg = {
        .data = (void *)message_buff,
        .size = sizeof(message_buff)
    };
    struct sid_status status = {};

    sid_error_t ret = sid_get_status(app_context, &status);
    if (ret != SID_ERROR_NONE) {
        SL_SID_LOG_APP_ERROR("Sidewalk stack is not initialized: %d", (int)ret);
        return;
    }

    if (status.state == SID_STATE_READY || status.state == SID_STATE_SECURE_CHANNEL_READY) {
        SL_SID_LOG_APP_INFO("sending message through Sidewalk");

        struct sid_msg_desc desc = {
            .type = SID_MSG_TYPE_NOTIFY,
            .link_type = SID_LINK_TYPE_ANY,
        };

        sid_error_t ret = sid_put_msg(app_context, &msg, &desc);
        if (ret != SID_ERROR_NONE) {
            SL_SID_LOG_APP_ERROR("queueing data failed: %d", (int)ret);
        } else {
            SL_SID_LOG_APP_INFO("queued data msg id: %u", desc.id);
        }
    } else {
        SL_SID_LOG_APP_ERROR("sidewalk is not ready yet");
    }
}
```

You can call the `send_sidewalk_message` function from the previously implemented main loop. Just add the function call to the `EVENT_TYPE_SEND_MESSAGE` event type and set up a trigger to queue this event (such as a CLI command, button press, or timer).

Silicon Labs also provides a software component that implements a lightweight message format scheme. You can find more information on this component in the [dedicated section](#).

Receive a Message Through Sidewalk

When a message is received from the Sidewalk network, the `on_msg_received` callback is called. You can find its definition in `app_process.c` of the Amazon Sidewalk Empty application.

```
void (*on_msg_received)(const struct sid_msg_desc *msg_desc, const struct sid_msg *msg, void *context);
```

When a message is received, the Sidewalk stack will invoke this callback, passing in the message descriptor and payload. We can then use these parameters to process the message accordingly. It is crucial to handle the received message within the callback efficiently to ensure the application responds correctly to incoming data from the Sidewalk network.

Here is an example of the `on_msg_received` callback implementation:


```
static void on_sidewalk_msg_received(const struct sid_msg_desc *msg_desc,
                                   const struct sid_msg *msg,
                                   void *context)
{
    UNUSED(context);
    SL_SID_LOG_APP_INFO("downlink message received");
    SL_SID_LOG_APP_INFO("msg (type: %d, id: %u, size: %u)", (int) msg_desc->type, msg_desc->id, msg->size);
    if (msg->size != 0) {
        SL_SID_LOG_APP_INFO("received message: %.*s", msg->size, (char *) msg->data);
    }
}
```

Amazon Sidewalk Callbacks

The Amazon Sidewalk stack operates on an event-driven architecture, which enables applications to respond to events as they occur in real-time. This architecture relies heavily on the use of callbacks to handle various events. Ensuring that these callbacks are implemented correctly is crucial for the reliable functioning of devices that utilize the Sidewalk network. As seen above during Sidewalk stack initialization, the list of callbacks is defined within the `struct sid_event_callbacks` as follows:

```
struct sid_event_callbacks {
    void *context;
    void (*on_event)(bool in_isr, void *context);
    void (*on_msg_received)(const struct sid_msg_desc *msg_desc, const struct sid_msg *msg, void *context);
    void (*on_msg_sent)(const struct sid_msg_desc *msg_desc, void *context);
    void (*on_send_error)(sid_error_t error, const struct sid_msg_desc *msg_desc, void *context);
    void (*on_status_changed)(const struct sid_status *status, void *context);
    void (*on_factory_reset)(void *context);
    void (*on_control_event_notify)(const struct sid_control_event_data *data, void *context);
};
```

Each callback serves a specific purpose within the Sidewalk stack's lifecycle, from handling events and message reception to dealing with errors and status changes. The `.context` is a pointer to the application context, which provides a reference that can be used within the callbacks to access application-specific data. For example, the callback `on_status_changed` can be used to update the status of the stack in the `context` user-defined structure. The callbacks prefixed with `on_` represent the functions that will be called when the corresponding event occurs, ensuring that the application can handle these events appropriately.

The `event_callbacks` structure is already present in the Amazon Sidewalk Empty example with all function callbacks already present but not implemented. The callbacks are defined as follows:

- `on_event` : is called for any generic Sidewalk event not already handled by another callback.
- `on_msg_received` : is triggered when a message is received by Sidewalk.
- `on_msg_sent` : is triggered when a message is effectively sent and corresponding ACK (from the MAC layer) is received.
- `on_send_error` : is triggered for any error during message sending including if message was sent but ACK (from the MAC layer) was not received.
- `on_status_changed` : when the Sidewalk status changes, for example when the stack changes from unregistered to time sync registered state (`SIDEWALK_NOT_READY > SIDEWALK_READY`).
- `on_factory_reset` : is triggered when a factory reset command was issued. A factory reset unregisters a Sidewalk device.

Amazon Sidewalk Physical Layer and Configuration

Switch Between BLE, FSK, and CSS

When Multi-link or Auto connect is enabled, the system automatically switches between links based on certain criteria, such as a timeout during time synchronization attempts. For more details on these features, refer to the dedicated section on the [Multi-link feature](#).

If you prefer to manually switch between radio links, you can stop the Sidewalk stack using `sid_stop` and restart it with `sid_start`, specifying a different `sid_link_type link_mask` parameter.

Additionally, it is possible to run another protocol alongside Amazon Sidewalk. The **Amazon Sidewalk - SoC Dynamic Multiprotocol Light** application demonstrates Client BLE working alongside Sidewalk BLE and Client BLE working alongside Sidewalk FSK. You can also implement double advertising in Amazon Sidewalk or perform an OTA using the standard BLE stack. For implementation details, refer to our [Multiprotocol documentation](#).

Change Radio Configuration

In the Amazon Sidewalk Empty sample application, the radio is pre-configured with selected parameters. You can modify these parameters by editing the parameter of the `sid_platform_init` function call in `app_init.c`.

```
sid_error_t sid_platform_init ( const void * platform_init_parameters );
```

The BLE configuration is managed in the `component/ble_subghz/radio/ble/app_ble_config.c` file of the Sidewalk extension, with the main configuration structure being `sid_ble_config_t ble_cfg`. The sub-GHz configuration is divided between different sub-GHz solutions: one for the native radio for Silicon Labs parts that support FSK modulation, and one for the external radio when using an external Semtech radio transceiver. The sub-GHz parameters can be found in the `sidewalk-<extension8version>/component/ble_subghz/radio/subghz/` folder, specifically in `rail/app_subghz_config.c` for the native radio, with `radio_efr32xgxx_device_config_t radio_efr32xgxx_cfg` as the base structure, and in the `semtech/app_subghz_config.c` folder for the Semtech radio, with `sid_sub_ghz_links_config sub_ghz_link_config` as the base structure.

Change Amazon Sidewalk Power Profile

With sub-GHz radio, besides adjusting the radio parameters, you can also select a configuration profile, known as a power profile. This profile alters the protocol behavior by modifying aspects such as the number of listening windows or their periodicity.

For FSK, refer to our documentation on [FSK Configuration](#)

For CSS, refer to our documentation on [CSS Configuration](#)

Modifying the power profile or parameters of the radio link affects the performance of the Amazon Sidewalk protocol. For more details, refer to our study on [Amazon Sidewalk performance](#).

Configure an External Module for SubGHz Radio

To support an external radio transceiver like the Semtech SX1262, you need to configure the communication between the EFR32 and the SX1262. This communication is carried out via SPI. The following section provides examples of the pinout connections between the EFR32 and the SX1262 and how to configure them.

By default, the Semtech SX1262 driver is added to the Sidewalk Empty application for hardware that supports it. To know which radio board supports the Semtech transceiver, refer to the [Hardware Prerequisite page](#).

If you wish to add support for the Semtech LR1110 radio transceiver, refer to the Semtech documentation [here](#).

Configure Transceiver GPIOs

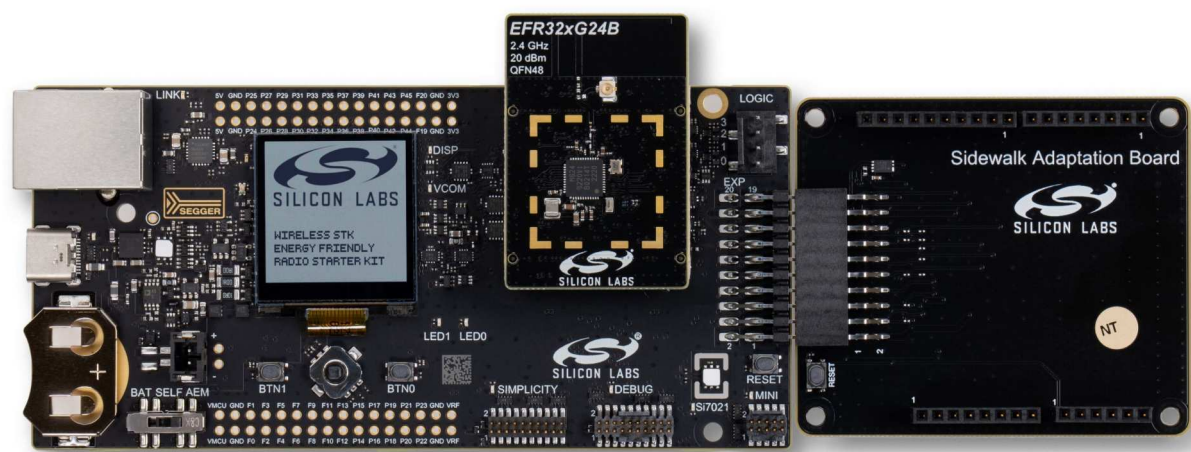
These instructions explain how to configure the EFR32 to communicate with the Semtech SX1262 for sub-GHz protocol support. When designing custom hardware, the same procedure can be followed, but the pinout can be adjusted as needed. It's important to consider that each pin has different availability depending on the sleep level. If you want the EFR to wake up on a radio event, choose the pinout accordingly.

Some Silicon Labs devices, like those in the EFR32xG28 family, support the FSK sub-GHz Sidewalk protocol natively (see the Hardware section in [Getting Started: Prerequisites](#) for more info). Other devices require an external transceiver for Sidewalk FSK support, which also provides support for the other sub-GHz Sidewalk protocol: CSS. For development purposes, this transceiver is conveniently available on the [Semtech SX1262MB2CAS LoRa shield](#). The Semtech shield can be connected to a main board using either an adapter board (recommended) or 10 female-to-female jumper wires (if adapter board is not available). Using this third-party sub-GHz radio module enables sub-GHz Sidewalk communication

(CSS and FSK) for EFR32xG21, EFR32xG24 and EFR32xG26 radio boards. Note that the Semtech shield is not necessary with the KG100S, as a Semtech transceiver is already integrated in that module.

INFO: For superior signal integrity, ease of use, and a more robust development platform, Silicon Labs recommends the Sidewalk Adaptation Board (BRD8042A, included in the [Silicon Labs Pro Kit for Amazon Sidewalk](#)) instead of jumpered wire connections with the radio boards.

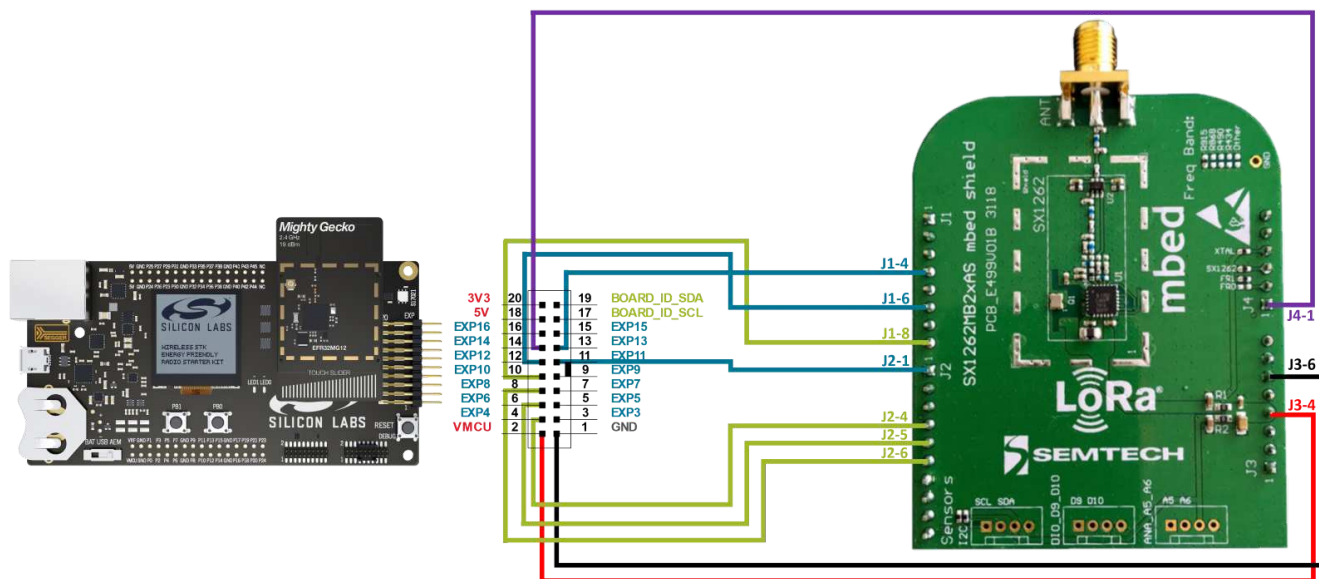
When using the Sidewalk Adaptation Board, connect it to the main board Expansion Header as shown below, and mount the Semtech shield to the female pin headers on the Adaptation Board.



When using jumper wires, connect the Semtech shield to the main board Expansion Header with the following scheme:

EFR32 Mainboard Exp. Pin	Semtech Shield Pin	Function
EXP_HEADER 1	J3-6	GND
EXP_HEADER 4	J2-4	SPI MOSI
EXP_HEADER 6	J2-5	SPI MISO
EXP_HEADER 8	J2-6	SPI SCK
EXP_HEADER 10	J1-8	SPI NSS
EXP_HEADER 11	J2-1	ANT_SW
EXP_HEADER 12	J1-6	DIO1
EXP_HEADER 13	J1-4	BUSY
EXP_HEADER 14	J4-1	SX NRESET
EXP_HEADER 2	J3-4	VMCU

The information in the table is also represented in the following image.



If you wish to customize the wiring between your sub-GHz chip and the EFR32, you can implement such changes in your application with the Pin Tool (in Simplicity Studio).

⚠ WARNING ⚠: In KG100S applications Silicon Labs recommends not using the SPI peripheral, because the multi-chip module (MCM) design already leverages it for communication between the EFR32 and the Semtech radio transceiver. Sharing the SPI bus with additional devices or peripherals can negatively impact time-critical radio control signals and lead to message failure in sub-GHz protocols.

For the SX1262, you can edit the pinout to define the corresponding GPIO for the busy pin, the antenna, DIO, reset, and chip select. Here is an example for the EFR32xG24 configuration in `config/app_gpio_config.h`:

```
// BUSY on PA07
// Used to indicate the status of internal state machine
#define SL_BUSY_PIN          7
#define SL_BUSY_PORT        gpioPortA

// ANT_SW on PA06
// External antenna switch to control antenna switch to RECEIVE or
// TRANSMIT.
#define SL_ANTSW_PIN         6
#define SL_ANTSW_PORT        gpioPortA

// DIO1 on PA08
// IRQ line from sx126x chip
// See sx126x datasheet for IRQs list.
#define SL_DIO_PIN           8
#define SL_DIO_PORT          gpioPortA

// SX NRESET on PA09
// Factory reset pin. Will be followed by standard calibration procedure
// and previous context will be lost.
#define SL_NRESET_PIN        9
#define SL_NRESET_PORT        gpioPortA

#define SL_SX_CS_PIN          SL_SPIDRV_EXP_CS_PIN
#define SL_SX_CS_PORT          SL_SPIDRV_EXP_CS_PORT
```

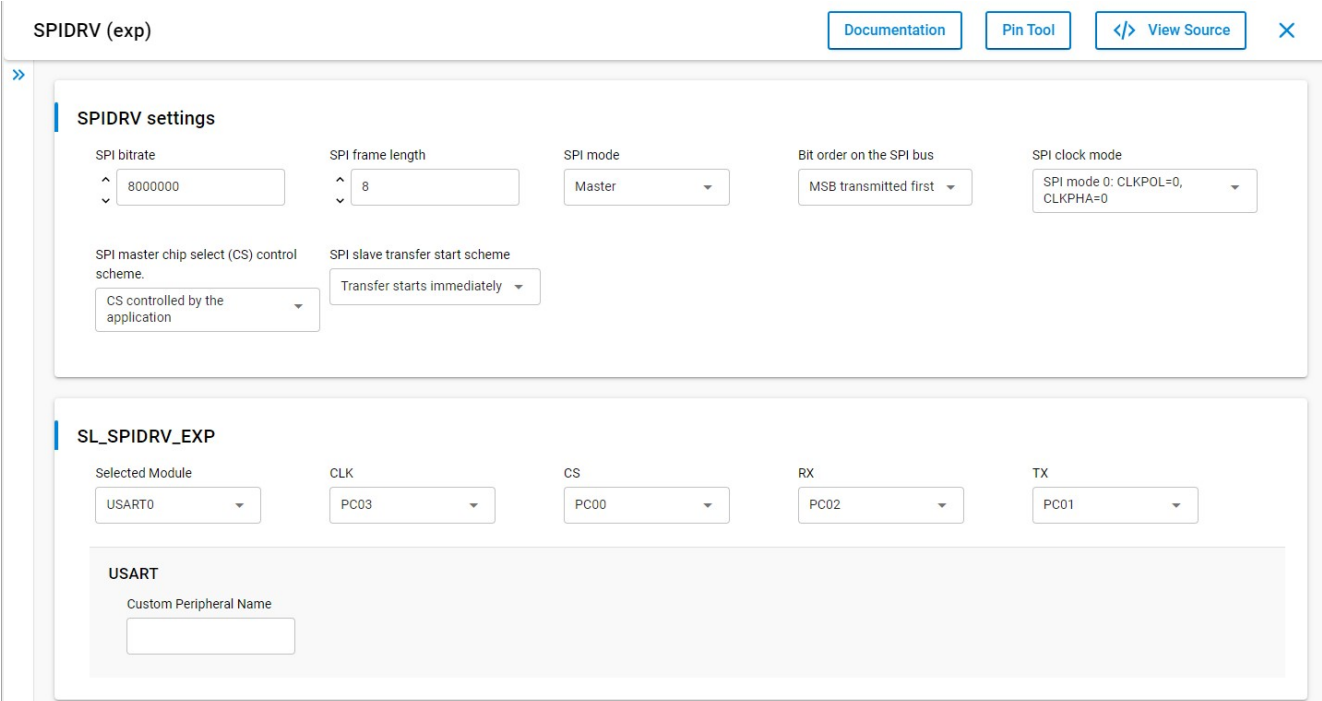
Once you have defined the pinout, you should also update the gpio lookup table in `autogen/app_gpio_config.c` accordingly.

On supported platforms, you should have the following symbols to your build: `SL_FSK_SUPPORTED`, `SL_CSS_SUPPORTED` and `SL_RADIO_EXTERNAL`. If you wish to remove the sub-GHz support from your example, remove the three symbols instead as follows:

1. In the **Project Explorer** panel, right-click on your project and open **Properties**.
2. In the left panel, navigate to **C/C++ General > Path and Symbols**.
3. In the central panel, navigate to **Symbols** and select **GNU C**.
4. In the list of symbols create `SL_FSK_SUPPORTED`, `SL_CSS_SUPPORTED` and `SL_RADIO_EXTERNAL` with value 1 to add the external transceiver support. Or delete them if you wish to remove sub-GHz support.
5. Click **Apply and Close**.

SPI Configuration

Once the SX1262 GPIOs are mapped, configure the SPI bus to communicate with the SX1262. To do so, open the SLCP file of your project and go to software components. Search for the **SPIDRV** component, it should already be installed, and click **Configure**. Under **SL_SPIDRV_EXP**, choose the USART module you want to use for the corresponding mapping. For example, the EFR32xG24 SPI driver configuration is as follows:



SPIDRV (exp)

Documentation Pin Tool </> View Source X

SPIDRV settings

SPI bitrate: 8000000
 SPI frame length: 8
 SPI mode: Master
 Bit order on the SPI bus: MSB transmitted first
 SPI clock mode: SPI mode 0: CLKPOL=0, CLKPHA=0

SPI master chip select (CS) control scheme: CS controlled by the application
 SPI slave transfer start scheme: Transfer starts immediately

SL_SPIDRV_EXP

Selected Module: USART0
 CLK: PC03
 CS: PC00
 RX: PC02
 TX: PC01

USART

Custom Peripheral Name:

Use the pinctool to help you choose available pins in your custom hardware.

Non-Volatile Memory Management in Sidewalk Context

Amazon Sidewalk example applications use non-volatile memory to store the application code, registration information, and manufacturing data. This section describes how this memory is used for each type of data stored.

⚠ WARNING ⚠: Starting Sidewalk extension 2.x.x, a change was made to the NVM3 usage in Sidewalk. This change is not backward compatible, you should be careful to use the manufacturing page generation that corresponds to the Sidewalk version that you are using.

In addition to the application code, one other NVM3 instance is used in an Amazon Sidewalk project: Default. The default instance is also used by the Simplicity SDK Suite for BLE stack status and cryptographic storage. This is where the wrapped

private keys are stored when using Secure Vault. For more information, see *section 2.1* of [AN1135: Using Third Generation Non-Volatile Memory \(NVM3\) Data Storage](#).

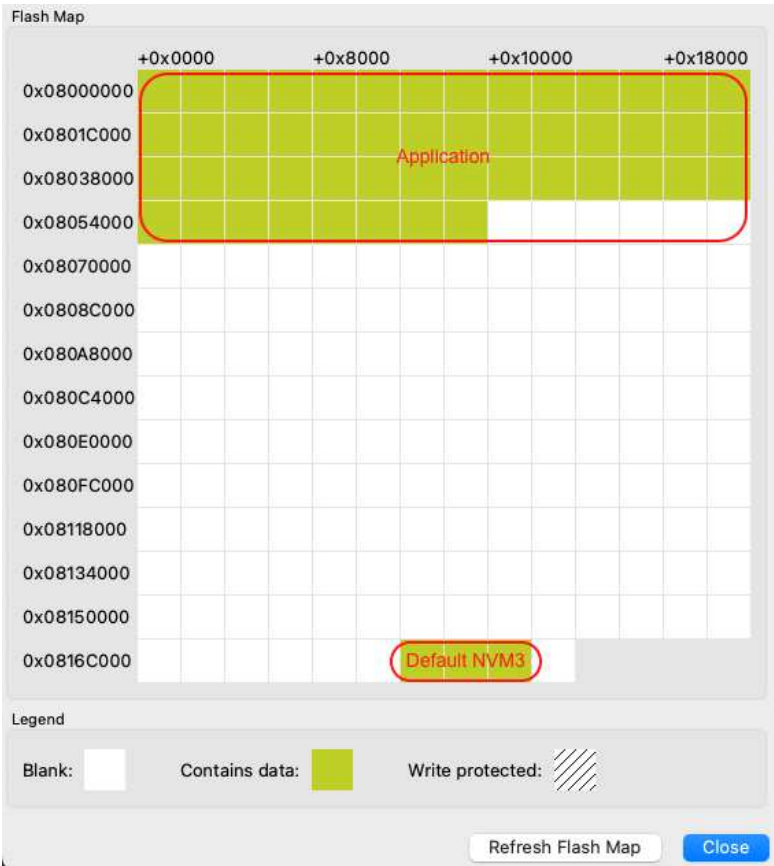
The default instance consists of the following logical partitions and subpartitions:

Partition	Subpartition	Key Range	Comments
Simplicity SDK	BLE stack metadata	0x40000 - 0x4FFFF	Used internally by Simplicity SDK for BLE stack
	PSA crypto metadata	0x83100 - 0x870FF	Used internally by Simplicity SDK for PSA crypto
	PSA crypto wrapped keys	0x83100 - 0x870FF	When using Secure Vault, Sidewalk ED25519 and P256R1 device private keys are wrapped in this subpartition
Sidewalk	Application specific	0xA0000 - 0xA1FFF	Sidewalk applications can store data on this partition
	KV-storage	0xA2000 - 0xA8FFF	Reserved for sidewalk stack usage
	Manufacturing data	0xA9000 - 0xAFFFF	Reserved for sidewalk stack usage

The manufacturing (MFG) partition is used to store the manufacturing page generated for every device. The manufacturing page contains various information such as device public keys, signatures, and Sidewalk Manufacturing Serial Number (SMSN). This instance is read-only from application code.

The Key-value (KV) storage partition contains information about the Amazon Sidewalk stack. The way the key-value pairs are stored on NVM3 is a bit different from the way the manufacturing information is stored. Objects are stored under groups, so there is actually one NVM3 object (so-called group) to store different information. The size of this instance depends on the needs of the stack (depending on the link layer). The values stored in the instance are not part of the public Amazon Sidewalk APIs. This instance is accessible in read-write from application code.

You can see an example of the memory layout in the following picture:



For more information on Silicon Labs NVM driver, see the dedicated documentation [NVM3 - NVM Data Manager](#). For background on NVM3, see the [Platform Resources](#) section.

Memory Map for EFR32xG21 Series

Region	Base Address	End Address	Size	Description
Application	0x00000000	-	-	Application code - size depends on application
Default NVM3 Instance	0x000EC000	0x000F2000	0x6000	Used for BLE stack. Wrapped keys are stored here when using Secure Vault.

Those addresses are given as an example for an EFR32xG21 with a flash size of 1024 kB. Base addresses can change to adapt to smaller flash sizes. You can use Simplicity Commander to display the memory layout of your application:

1. With your kit selected, go to **Device Info** panel.
2. Select **Flash Map** in **Main Flash** panel.

Memory Map for EFR32xG24 Series

Region	Base Address	End Address	Size	Description
Application	0x08000000	-	-	Application code - size depends on application
Default NVM3 Instance	0x0816C000	0x08172000	0x6000	Used for BLE stack. Wrapped keys are stored here when using Secure Vault.

Those addresses are given as an example for an EFR32xG24 with a flash size of 1536 kB. The base addresses can change to adapt to smaller flash sizes or other radio boards. You can use Simplicity Commander to display the memory layout of your application:

1. With your kit selected, go to **Device Info** panel.
2. Select **Flash Map** in **Main Flash** panel.

Handle Logs Level and Preferred Outputs

A specific set of components were created to handle the Sidewalk logs. You can find more information about this components in the [Platform Abstraction Layer documentation](#).

To display logs or switch the log output from RTT to UART, refer to [Testing and Debugging documentation](#).

Amazon Sidewalk Multi-link and Auto Connect Feature

Since the release of Sidewalk SDK 1.16, a feature called Multi-link has been introduced. This feature abstracts the process of connection establishment and maintenance for the radio links supported by the Sidewalk stack, making it easier for developers. It offers developers varying degrees of flexibility to control not only the connection behavior of the links but also the transfer of messages over them.

You can find the dedicated page on Multi-link in our documentation [here](#).

Additional Documentation

- [Power Consumption Analysis](#): For more information on power consumption and optimization
- [Sidewalk Performances](#): For more information on Sidewalk performances and how to configure Sidewalk for your use case
- [Amazon Sidewalk API](#): For more details on Amazon Sidewalk API
- [AWS API Reference for Amazon Sidewalk](#): For AWS Sidewalk Cloud Documentation
- [Sidewalk and Bluetooth FUOTA](#): For more information on FUOTA solutions in Amazon Sidewalk context
- [Silicon Labs Amazon Sidewalk Applications Github repository](#): Additional more advanced sample applications for Amazon Sidewalk

What's Next?

-

[Qualifying a Product](#)

- [Amazon Sidewalk - SoC Qualification sample application](#) that contains all the necessary CLI commands to pass the qualification with Amazon
- [Manufacturing a Product](#)

Amazon Sidewalk API

Amazon Sidewalk API

Refer to the [Amazon Sidewalk Sid API Developer Guide](#).

Testing and Debugging

Testing and Debugging

RTT Logs

The UART interface is not always available to report traces from Sidewalk example applications. Instead, reporting leverages the J-Link RTT interface. To set up the communication between your PC and the EFR32, follow these instructions.

1. Install the [J-Link RTT Viewer](#).
2. Open the J-Link RTT Viewer.
3. In the **Configuration** panel, **Connection to J-Link** section, select **USB**.
4. In the **Specify Target Device** list, select the connected part (for example EFR32BG21BxxxF1024 or EFR32MG24AxxxF1536).
5. In the **Target Interface & Speed** panel, select **SWD** and **4000 kHz**.
6. In the **RTT Control Block** panel, select **Auto Detection**.
7. Click **OK**.

INFO: For the KG100S module, select EFR32BG21BxxxF1024 as the target device.

A terminal opens and the Sidewalk application traces are output as shown below (Bluetooth application).

```
00> [00000001] <I> hello neighbor application started
00> [00000001] <I> sidewalk stack version 1.17.1-13
00> [00000002] <I> silabs sidewalk extension version 2.2.1
02> [00000006] <I> pal kv-storage: kv store opened with 0 object(s)
02> [00000007] <I> pal swi: interrupt init ok
02> [00000008] <W> pal mfg: no objects found in mfg store
```

WARNING: J-Link RTT and Simplicity Studio use the same channel to communicate with the board. If you do not see logs in J-Link RTT, try closing and re-opening Simplicity Studio to reset the connection.

By default, in Amazon Sidewalk sample application, the RTT logs are splitted accross 3 terminals:

- Terminal 0: Outputs the application logs. They are easy to understand and give basic information on the application status.
- Terminal 1: Outputs the Sidewalk stack logs. They are more difficult to understand and require a deep knowledge of the Sidewalk stack. Those logs are in the Amazon Sidewalk stack, the sources are not available.
- Terminal 2: Outputs the Sidewalk Platform Abstraction Layer (PAL) logs. They require some knowledge of Sidewalk and can easily be found in the code.

Switch Logs from RTT to UART

The initial step is to verify that VCOM is active. This can be confirmed by ensuring that SL_BOARD_ENABLE_VCOM is set to 1 in the config/sl_board_control_config.h file. If it is not, you should adjust it accordingly.

To configure the log output from RTT to UART, follow these steps:

1. Navigate to **Software Components** and search for *board control*.

2. In the Board Control component, check if **Enable Virtual COM UART** is enabled.
3. Navigate to **Software Components** and search for *iostream*.
4. Install the **IO Stream: USART** component (or **IO Stream: EUSART** for xG28 and xG24).
5. Access the configuration settings of the newly created VCOM instance called **vcom**.
6. If available, disable **Enable High frequency mode** and **Restrict the energy mode** to allow the reception options.
7. In `config/sl_cli_config_inst.h` change the value of `SL_CLLINST_IOSTREAM_HANDLE` to `sl_iostream_vcom_handle`.
8. Navigate back to **Software Components** and search for *sidewalk*.
9. Uninstall the **Sidewalk Log: App RTT** component and install **Sidewalk Log: App VCOM** component in its place.
10. You can also completely remove RTT by replacing **Sidewalk Log: PAL RTT** and **Sidewalk Log: stack RTT** by their VCOM equivalent. Then, remove the **IO Stream: RTT** component.

Cloud Application Debugging

To debug your cloud application, several AWS objects can be use to monitor the events in your account. See [Amazon CloudWatch](#) for more information.

Power Consumption Analysis

Power Consumption Analysis

This section provides a comprehensive analysis of power consumption for the Amazon Sidewalk protocol, encompassing the three physical layers: BLE, FSK, and CSS. This section details the testing procedures and methodologies applied to each radio layer. Every step of the process is thoroughly explained to ensure the ability to replicate the measurements. Finally, this section culminates in presenting the results for each radio layer, highlighting the average power consumption for fundamental events such as transmission (TX), reception (RX), registration, time synchronization, and idle power, among others.

For more information on Energy modes for Silicon Labs platforms, see the dedicated documentation on [Power Manager](#).

Testing Scenario

In this section are example use cases for each physical layer: BLE (Bluetooth Low Energy), FSK (Frequency-Shift Keying), and CSS (Chirp Spread Spectrum). In each scenario, for each uplink with payload of 19 bytes sent to the cloud, the cloud answered with an ACK message (0 bytes payload). Default parameters for each radio layer are provided. Tests were run on those default configurations to extract Amazon Sidewalk performance out of the box.

Hardware

To perform the necessary tests, various platforms were selected for each physical layer (PHY).

Reference platforms include:

- For BLE: the [EFR32xG24 radio board](#) along with our mainboard
- For FSK: the [EFR32xG28 radio board](#) along with our mainboard
- For CSS: a combination of the [EFR32xG24 radio board](#) and the Semtech SX1262 radio module

Creating a Power Optimized Application

This study requires a fundamental application that is optimized for power efficiency while still running the Amazon Sidewalk stack. This application was derived from the Hello Neighbor sample application, with only minor modifications implemented. By default, the Hello Neighbor application operates in EM2 energy mode when idle. The initial modification involved switching the log output from RTT to UART.

Create a Hello Neighbor Application

Follow public documentation to create the Hello Neighbor application [here](#).

Switch Log Output and CLI to UART

The initial step was to verify that VCOM is active. This was confirmed by ensuring that `SL_BOARD_ENABLE_VCOM` is set to 1 in the `config/sl_board_control_config.h` file. If it is not, you should adjust it accordingly.

To configure the log output from RTT to UART, follow these steps:

1. Navigate to **Software Components** and search for *iostream*.
2. Install the **IO Stream: USART** component (or **IO Stream: EUSART** for xG28 and xG24).
3. Access the configuration settings of the newly created VCOM instance.
4. Deactivate **Enable High frequency mode** and **Restrict the energy mode** to allow the reception options. Then, modify the Baud rate to 9600.
5. In `config/sl_cli_config_inst.h`, change the value of `SL_CLI_INST_IOSTREAM_HANDLE` to `sl_iostream_vcom_handle`.

6. Navigate back to **Software Components** and search for *sidewalk*.
7. Uninstall the **Sidewalk Log: App RTT** component and install **Sidewalk Log: App VCOM** component in its place.

⚠ WARNING ⚠: Regarding the log output alteration from RTT to UART when using xG24 with the Semtech module, it is necessary to employ a UART bridge. This requires remapping the UART pins to available GPIOs where the UART bridge is connected. Since the Semtech connection occupies the UART pins, the WSTK's VCOM support cannot be utilized in this setup.

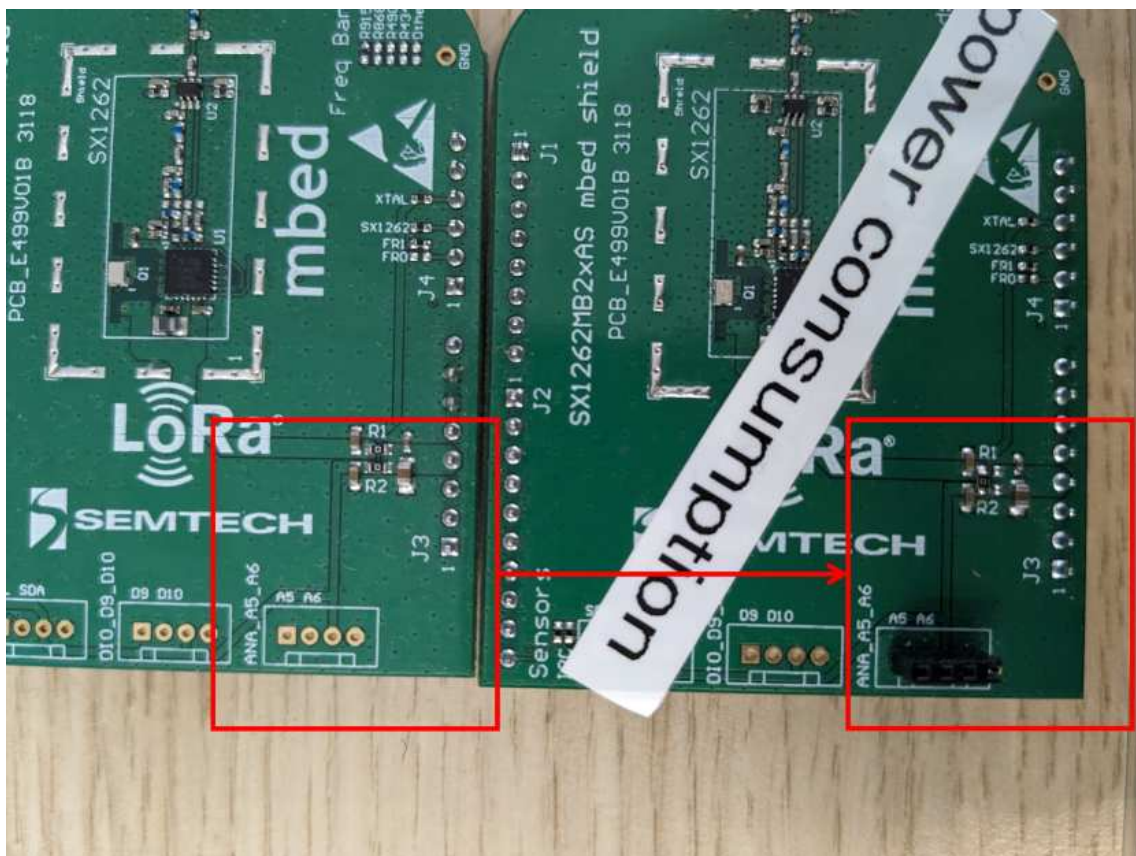
① INFO ①: For the xG24 configuration, it is preset to accommodate the Semtech module. However, if your usage is limited to BLE, you can omit the Semtech-specific parameters, thus leveraging the VCOM capability of the mainboard.

Power Consumption Setup

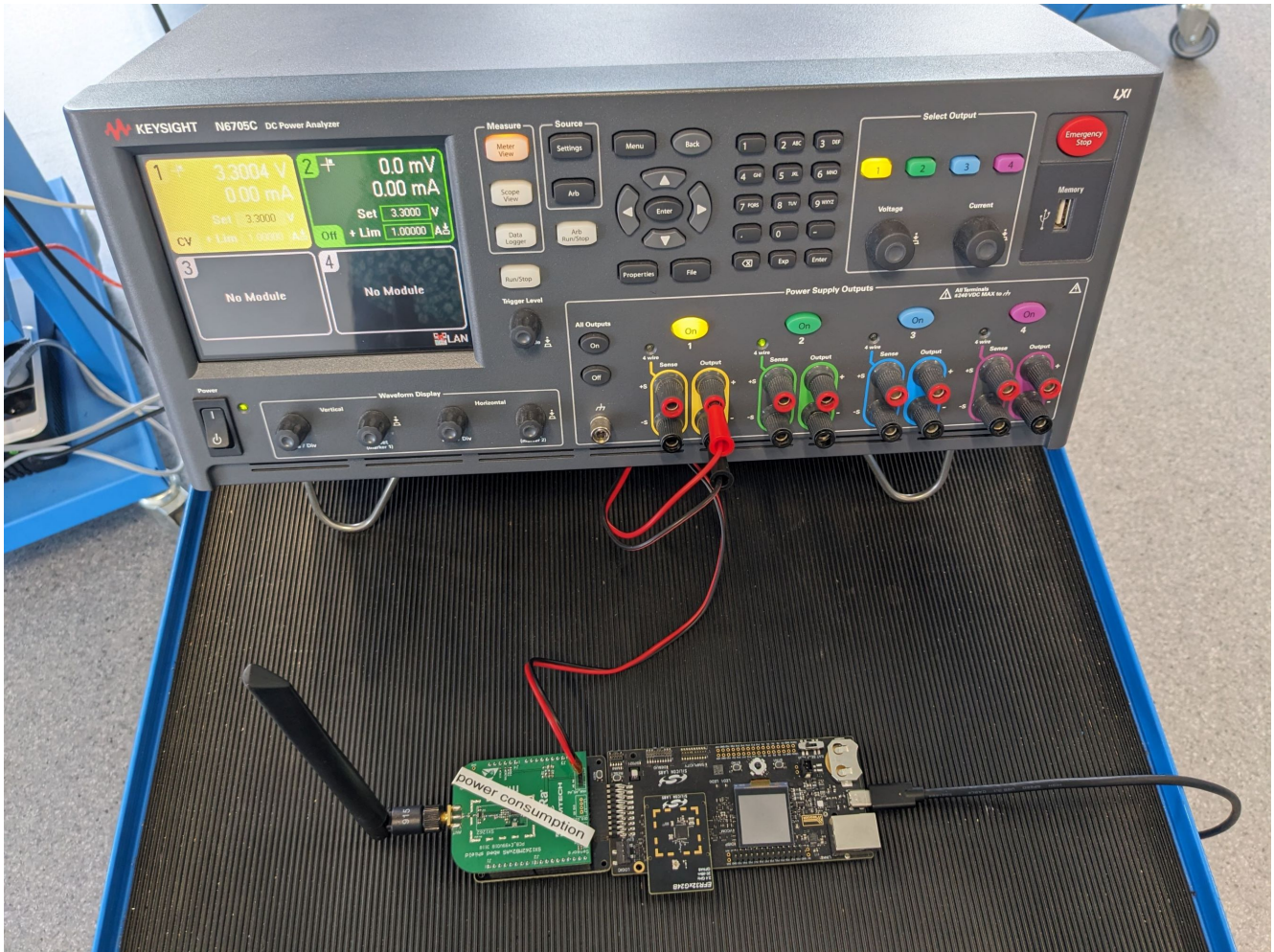
For each of the three physical layers, the Simplicity Studio Energy Profiler was used to assess power consumption. The power usage of the Semtech SX1262 was evaluated independently with a DC/DC power Analyzer from Keysight. To conduct the power analysis on the Semtech radio module, a minor hardware alteration was required to facilitate an external power source, as the module derives power from the DC/DC power Analyzer's supply.

Semtech SX1262 Radio Module Modification

A 0 ohm resistor needed to be removed and another shifted to redirect power supply from the mainboard to the DC/DC Power Analyzer, as shown in the following picture.



The final setup with the DC/DC Power Analyzer looks like the picture below.

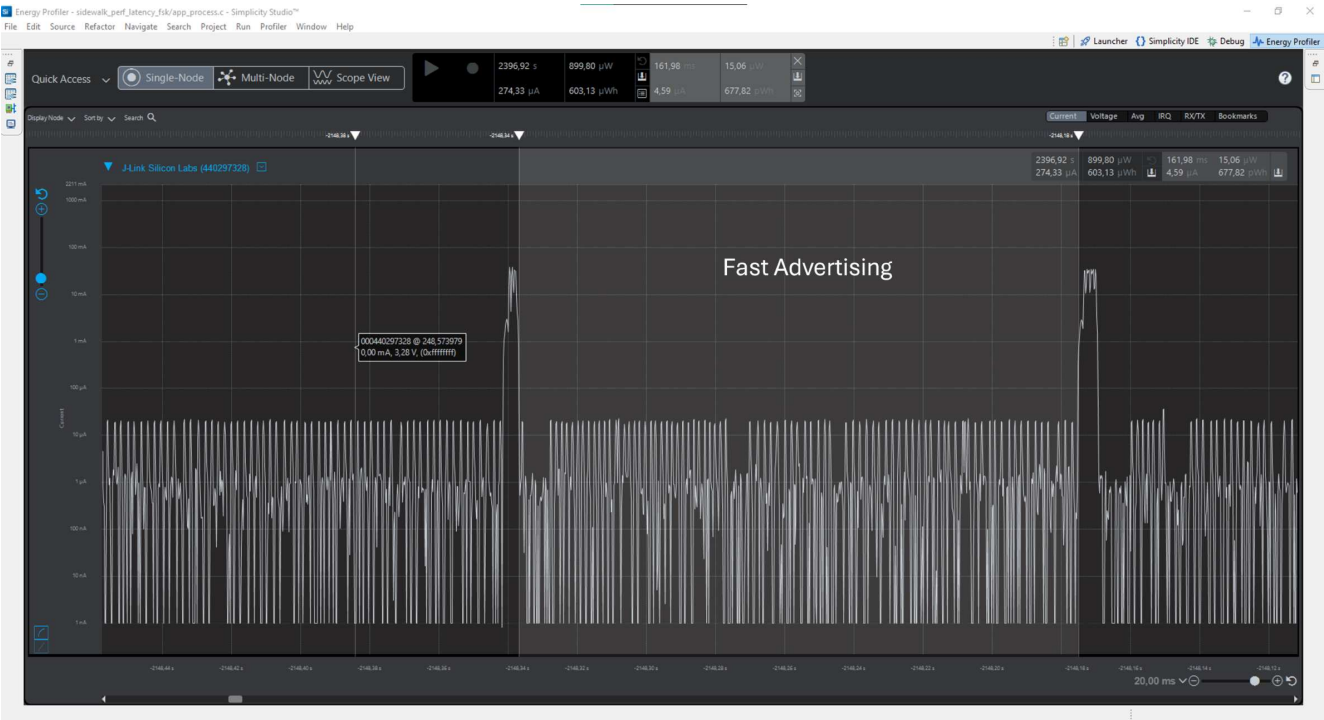


Once all the necessary tools are in place, review each physical layer power consumption.

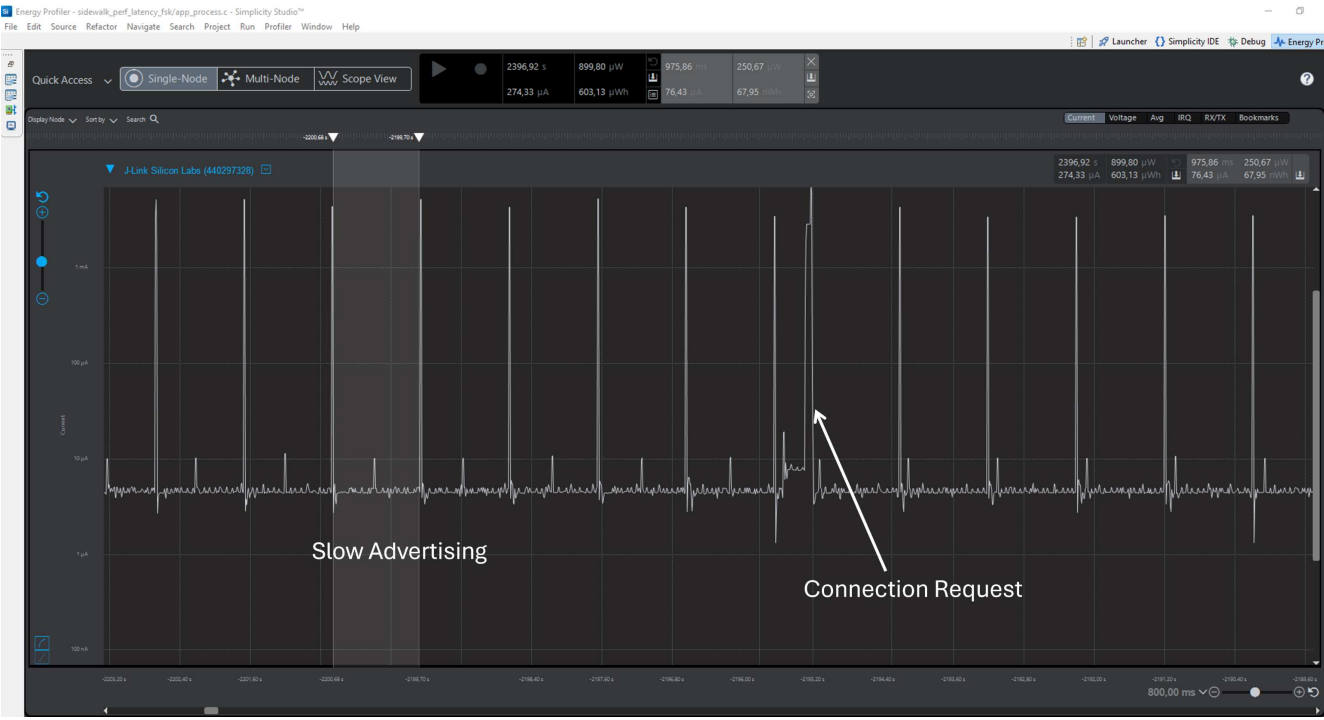
BLE

Per Amazon's guidelines, the process begins with a Fast Advertising phase that lasts for 30 seconds, emitting an advertising beacon every 160 milliseconds. This phase is succeeded by Slow Advertising, during which the beacon is broadcasted every second. Subsequently, the BLE endpoint and the gateway can establish a connection to exchange messages. This connected state is maintained for a minimum of 30 seconds, with a default connection interval of 30 milliseconds.

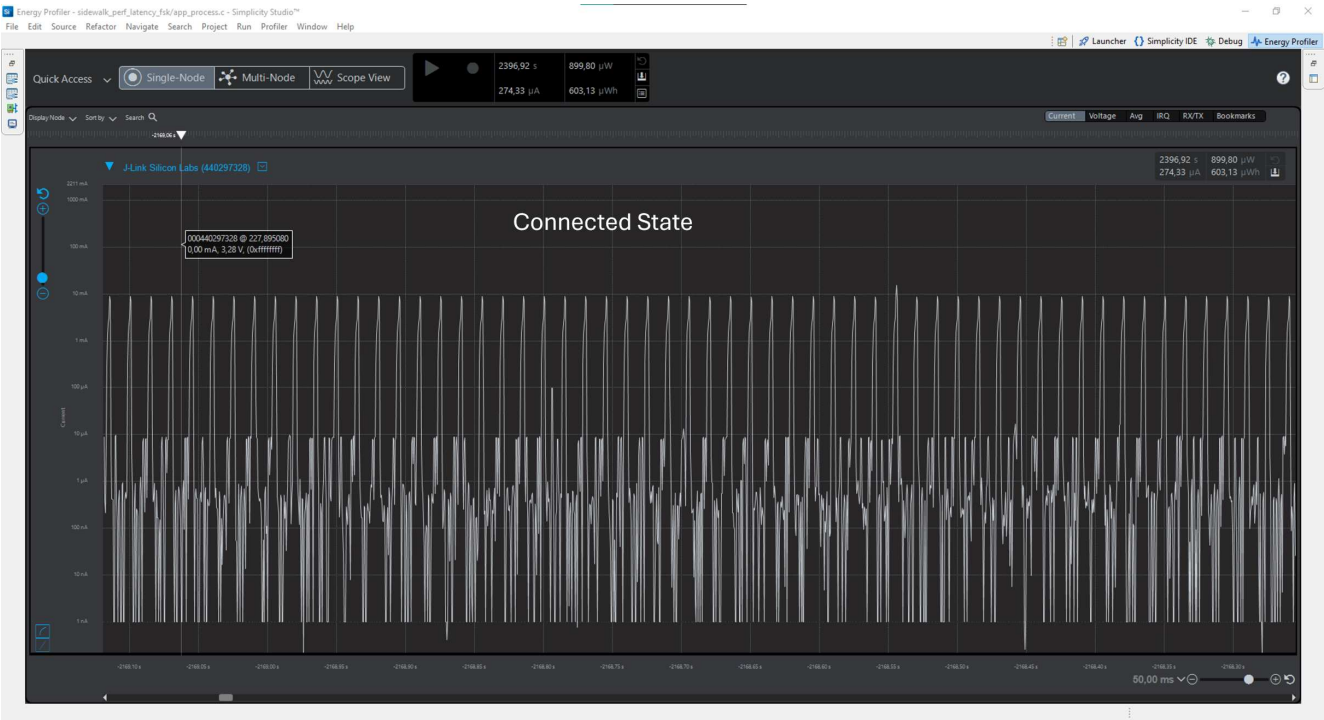
The following picture shows the endpoint power consumption graph during fast advertising.



The following picture shows the endpoint power consumption graph during slow advertising.



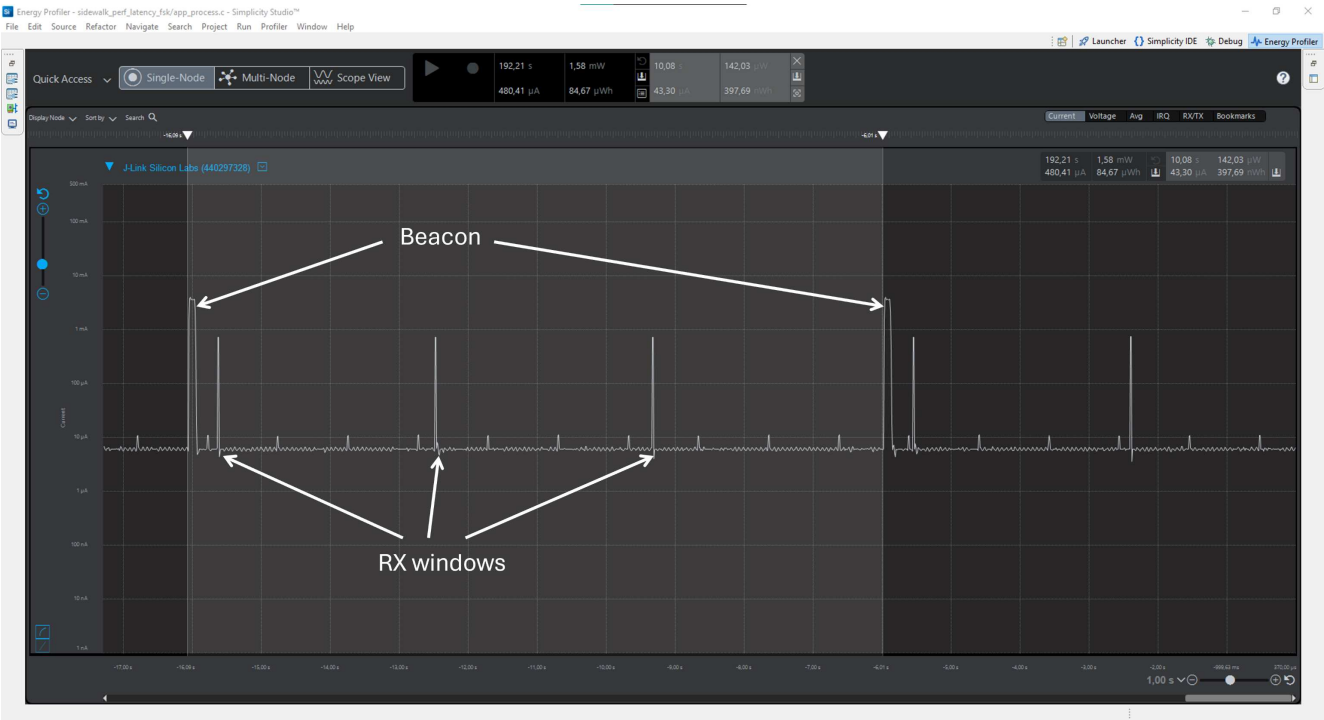
The following picture shows the endpoint power consumption graph while connected to the GW.



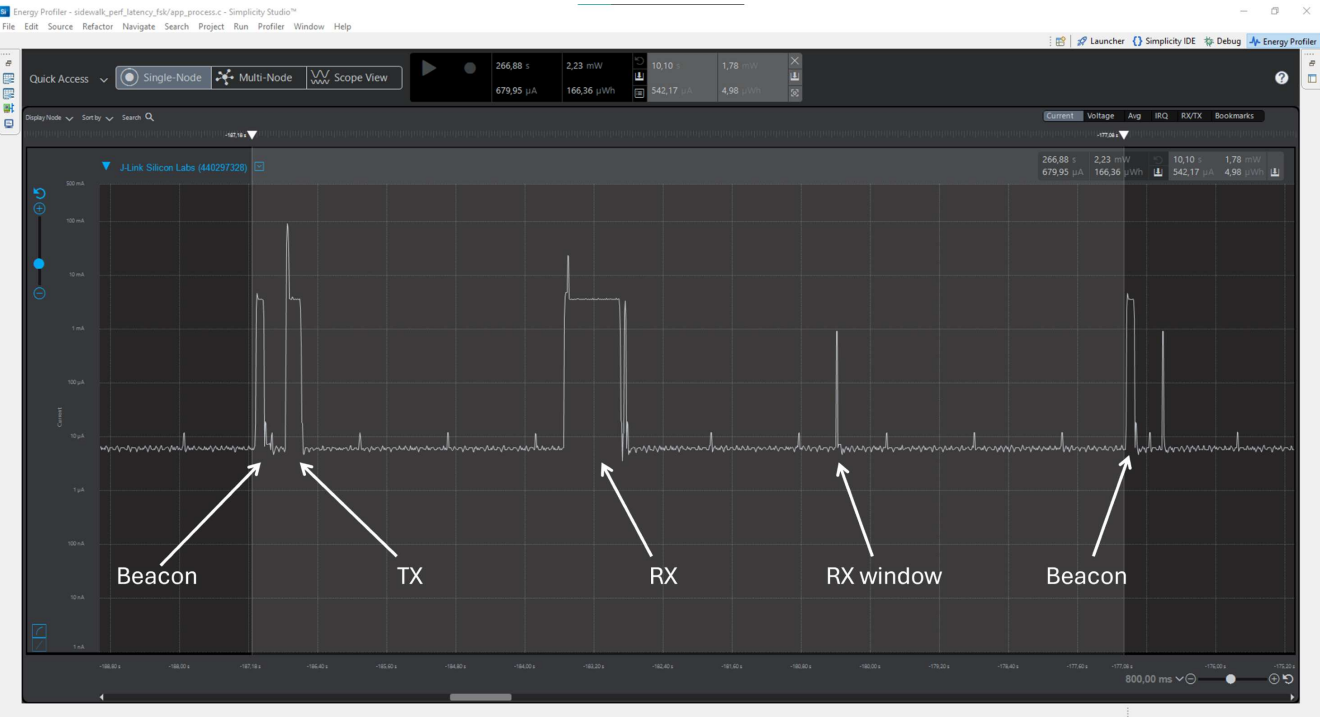
FSK

As per Amazon's standards for FSK, the connection with the gateway is sustained by sending beacons at 10-second intervals. The default configuration permits three opportunities for listening in the time between each beacon. Transmissions may occur at any point between beacons when the device is not in a listening window.

The following picture shows the endpoint power consumption graph between 2 beacons.



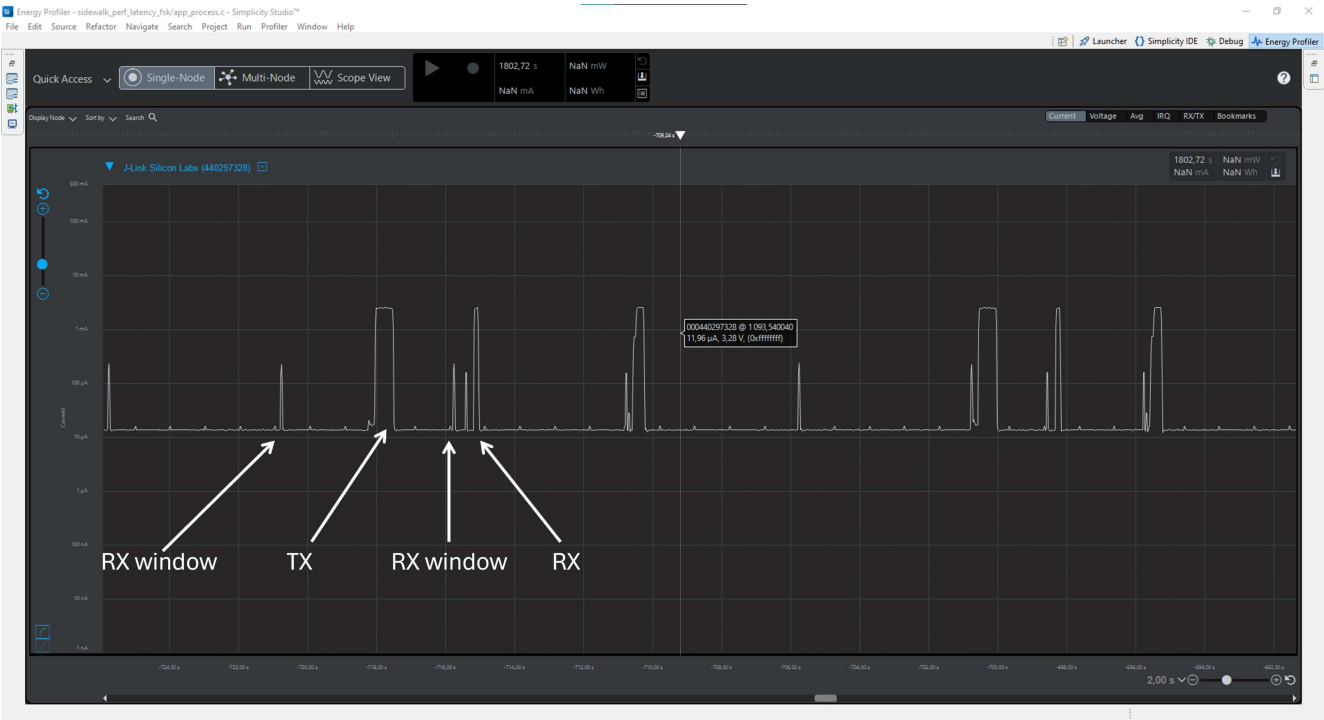
The following picture shows the endpoint power consumption graph between 2 beacons with a transmission and a reception.



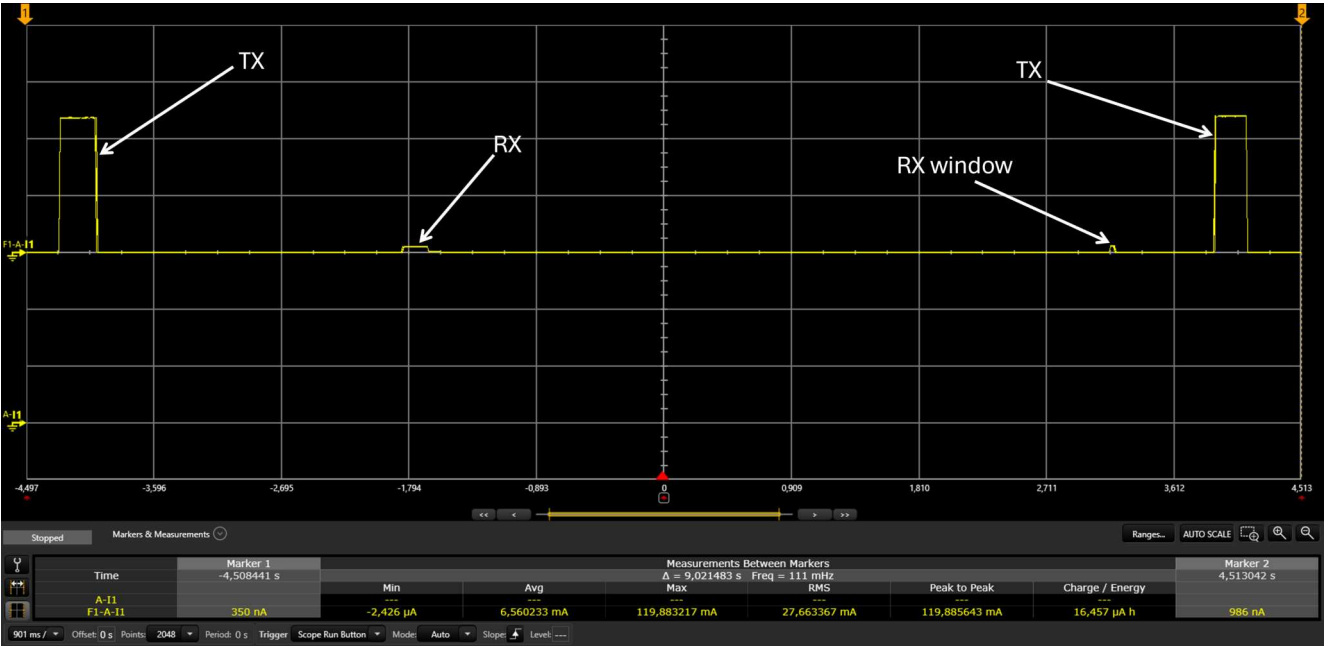
CSS

CSS operates on an asynchronous protocol, aligning its timing with the gateway as needed and executing transmissions when necessary. Listening windows are set to activate every 5 seconds consistently, or they can be triggered after a transmission, depending on the selected power profile. The default power profile utilized in the subsequent graphs is profile B.

The following picture shows the endpoint power consumption of the EFR32xG24 radio board during a transmission and a reception.



The following picture shows the endpoint power consumption of the SX1262 radio module during a transmission and a reception.



Power Consumption Results

The results of the average power consumption study on fundamental events are provided in the following tables. These include:

- Registration
- Time Synchronization
- BLE Advertising

- BLE Connected State
- TX
 - RX
 - Idle State

BLE

BLE Physical Layer	Details
Radio Board	xG24
Physical Layer	BLE
Output Power	20 dBm
TX Payload Size	19 bytes
RX Payload Size	0 byte
Fast Advertising Interval	160 ms
Slow Advertising Interval	1000 ms
Connected State Minimal Duration	30 s

Fundamental Events Average Power Consumption	Results
Registration	2.75 mA
Time Synchronization	1.9 mA
Fast Advertisement	464.67 µA
Slow Advertisement	98.31 µA
Connected State with 1 TX Event	743.11 µA
TX 19 bytes	3.51 mA
RX 0 byte	3.39 mA
30 seconds Connected State without TX	713.79 µA
Idle State	4.82 µA

FSK

FSK Physical Layer	Details
Radio Board	xG28
Radio Layer	FSK
Output Power	20 dBm
TX Payload Size	19 bytes
RX Payload Size	0 byte
Number of RX Window in-between Beacons	3

Fundamental Events Average Power Consumption	Results
Registration	2.66 mA
Time Synchronization	1.03 mA
Beacon Reception	2.44 mA
Idle State	6.12 µA
TX 19 bytes including CCA and ACK	14.21 mA
RX Window	534.11 µA
RX 0 byte	3.84 mA

Fundamental Events Average Power Consumption	Results
10 Seconds Loop Average	43.23 μ A

CSS

CSS Physical Layer	Details
Radio Board	xG24 + SX1262
Radio Layer	CSS
Output Power	20 dBm
TX Payload Size	19 bytes
RX Payload Size	0 byte

Fundamental Events Average Power Consumption	EFR32	SX1262	Global Power Consumption
Time Synchronization	400.88 μ A	10.11 mA	10.51 mA
Idle State	13.7 μ A	547 nA	14.25 μ A
TX 19 bytes	1.67 mA	107.94 mA	109.61 mA
RX Window	55.07 μ A	1.94 mA	1.99 mA
RX 0 bytes	733.47 μ A	3.9 mA	4.63 mA
20 RX windows after TX	15.52 μ A	35.686 μ A	51.2 μ A

Going Further

For each physical layer, several parameters can be tweaked to influence the protocol behavior, thus impacting average power consumption. Below are the most commonly thought after parameters and where to change them.

BLE

The BLE configuration can be seen in the file `sidewalk_<version>/component/ble_subghz/radio/ble/app_ble_config.c` of your sample application. Most enumeration related to Sidewalk BLE are defined in `sidewalk_<version>/component/includes/projects/sid/sal/common/public/sid_ifc/sid_ble_cfg/sid_ble_config_ifc.h`. The default device BLE name is "SL_SIDEWALK", MTU size is 247 bytes, and MAC address type is random private resolvable. BLE connection parameters are chosen by the gateway and the connection timeout is 30 seconds.

Advertising is divided into two behaviors:

- Fast advertising: transmit beacons every 160 ms for 30 seconds after boot
- Slow advertising: transmit beacons every 1 s after fast advertising

You can check the following structures in the [API Reference](#):

- `sid_ble_cfg_adv_param_t` : for advertising parameters
- `sid_ble_cfg_conn_param_t` : for the connection parameters
- `sid_ble_cfg_gatt_profile_t` : for the GATT profile parameters
- `sid_ble_config_t` : for more generic configuration parameters

For more information on Silicon Labs BLE stack and power consumption, see the following pages:

- [BLE General Overview](#)
- [Optimizing Current Consumption in Bluetooth Low Energy Devices](#)
- [Current Consumption Variation with TX Power](#)

FSK

The FSK radio configuration can be seen in the file `sidewalk_<version>/component/ble_subghz/radio/subghz/rail/app_subghz_config.c` of your sample application for Silicon Labs

hardware and `sidewalk_<version>/component/ble_subghz/radio/subghz/semtech/app_subghz_config.c` for Semtech hardware. Most Sidewalk FSK enumeration are defined in

`sidewalk_<version>\component\includes\projects\sid\sil\common\public\sid_ifc\sid_900_cfg\sid_900_cfg.h`. While running with FSK, your EFR32 will be either in EM2 or EM0 energy modes depending on radio events. During radio events like beacons, RX, or TX, your EFR32 will run in EM0 energy mode and in EM2 outside of those events.

For more power optimization, check Amazon Sidewalk power profiles for [FSK](#).

CSS

The CSS radio configuration can be seen in the file

`sidewalk_<version>/component/ble_subghz/radio/subghz/rail/app_subghz_config.c` of your sample application for Silicon Labs hardware and `sidewalk_<version>/component/ble_subghz/radio/subghz/semtech/app_subghz_config.c` for Semtech hardware. While running with CSS, your EFR32 will be either in EM2 or EM0 energy modes depending on radio events. During radio events like RX or TX your EFR32 will run in EM0 energy mode and in EM2 outside of those events.

For more power optimization, check Amazon Sidewalk power profiles for [CSS](#).

Performance

Amazon Sidewalk Performance

Amazon Sidewalk is a new ecosystem for creating shared wireless networks connecting IoT devices at homes, and beyond the front door, across the entire neighborhood, and even the city. Amazon Sidewalk provides the following benefits:

- Offers free collaborative network coverage based on existing products already in end-users' homes
- Offers reliable connectivity where it otherwise is not present today
- Complements existing IoT protocols
- Removes the need for a proprietary gateway
- Provides range extension, frustration-free setup, automatic connection, and AWS connectivity

The goal of this section is to provide an analysis of Amazon Sidewalk's performance to give an idea of possible use cases depending on configuration. It discusses the Amazon Sidewalk ecosystem, its configurations, protocols, testing methods, and results on power consumption, network latency, range, and throughput testing. This section aims to provide insights on how to extract the best performance from Amazon Sidewalk.

How to Extract Amazon Sidewalk Best Performances

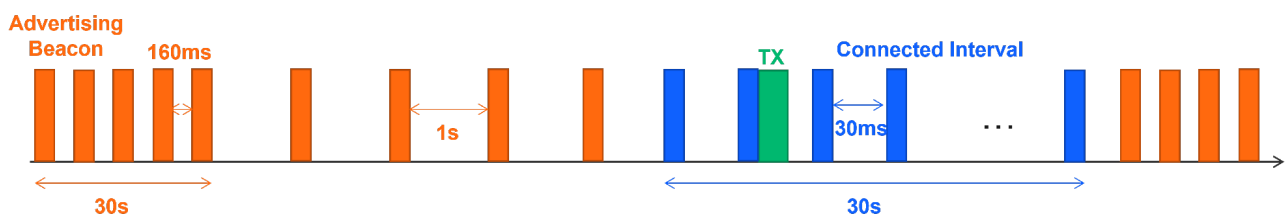
Testing Scenarios

In this section, Silicon Labs considered example use cases for each PHY: BLE (Bluetooth Low Energy), FSK (Frequency-Shift Keying), and CSS (Chirp Spread Spectrum). In each scenario, the setup was to send regular uplinks with payload of 19 bytes to the cloud and wait for an ACK message (0 bytes payload) from the cloud. Default parameters for each radio layer are provided. Tests were run on those default configurations to extract Amazon Sidewalk performance out of the box. For CSS, an alternate configuration was studied to try both power profiles as it changes a lot on the behavior.

Silicon Labs provides the analysis for each link in their default configurations, and also describes the level of configurability for each link. Silicon Labs recommends that developers review the Summary section and explore how to tune the performance of your design based on your application.

BLE Scenario

In accordance with Amazon's specifications, the behavior of BLE is characterized by the use of Bluetooth 5 with a 1 Mbps PHY. The BLE endpoint was established, with the gateway (Amazon Echo 4 for example) assuming the role of BLE Central. Initially, there was a Fast Advertising phase lasting 30 seconds, during which an advertising beacon was emitted every 160ms. This was followed by Slow Advertising, where the beacon was broadcast every 1 second. With no transmission, the connection remained active for 30 seconds. The transmission power was set at 20 dBm, and these settings constituted the default configuration.

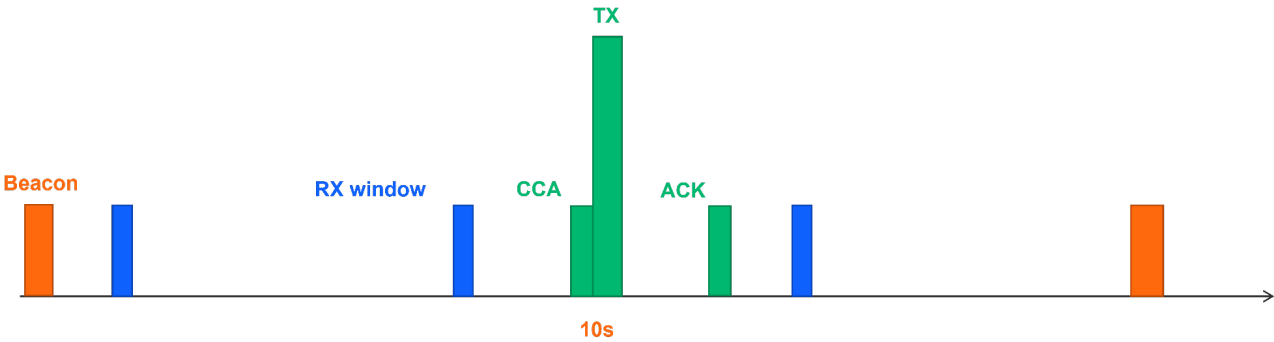


FSK Scenario

According to Amazon's specifications, the FSK behavior includes a 50 Kbps FSK communication speed and a synchronous protocol that keeps the device always connected to the gateway. The connection was maintained through beacons sent

every 10 seconds, with default settings allowing for three listening windows and transmission opportunities between beacons. The transmission power was set at 20 dBm.

FSK allows for two power profiles 1 and 2. The difference between Power profile 1 and Power profile 2 is in the choice of configuration and its implications for device performance. Power profile 1, which is chosen by the gateway, was configured for high power usage, high-throughput, and low-latency. On the other hand, Power profile 2, which is selected by the endpoint, allows the developer to customize the parameters based on the specific needs of their use case. In the context of this study, the same parameters were applied to both power profiles, resulting in no variation in average power consumption and latency outcomes. The default power profile is power profile 1.

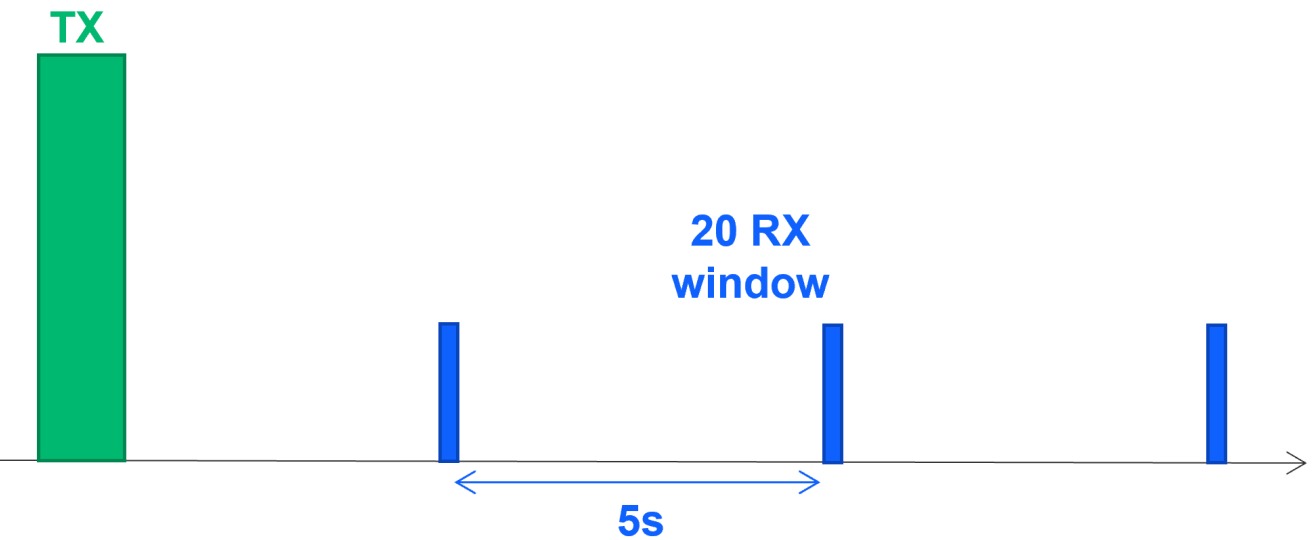


CSS Scenarios

CSS is an asynchronous protocol. It synchronizes time with the gateway when necessary and conducts transmissions as required. The transmission power is set at 20 dBm by default.

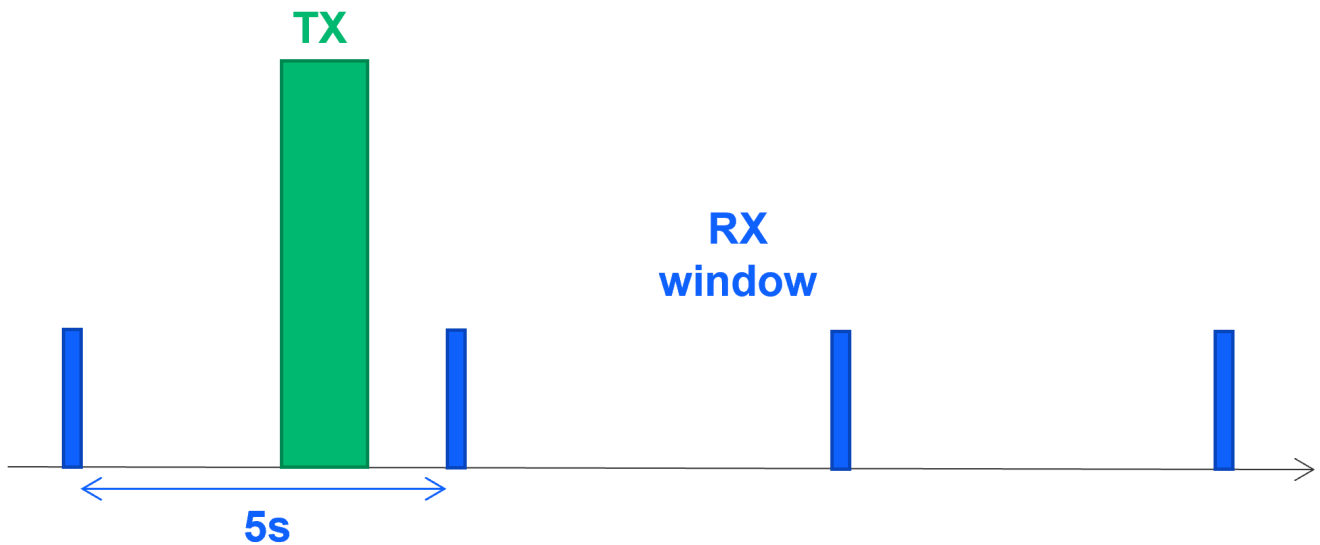
CSS Power Profile A

For the CSS power profile A, listening windows were activated only after a transmission, with up to 20 windows available, one occurring every 5 seconds. A time synchronization was required before any transmission if the last one was more than 5 minutes prior. For the purposes of this study, the default number of RX windows after a transmission was used, which is 20.



CSS Power Profile B

In CSS power profile B, listening windows were scheduled periodically, occurring every 5 seconds indefinitely. Additionally, a keep-alive signal was sent every 5 minutes.



Hardware

To perform the necessary tests, various platforms were selected for each physical layer (PHY).

Reference platforms include:

- For BLE: the [EFR32xG24 radio board](#) on the mainboard
- For FSK: the [EFR32xG28 radio board](#) on the mainboard
- For CSS: a combination of the [EFR32xG24 radio board](#) and the Semtech SX1262 radio module
- For CSS range tests: the [Amazon Sidewalk Test Kit](#)

Power Consumption

Testing Method

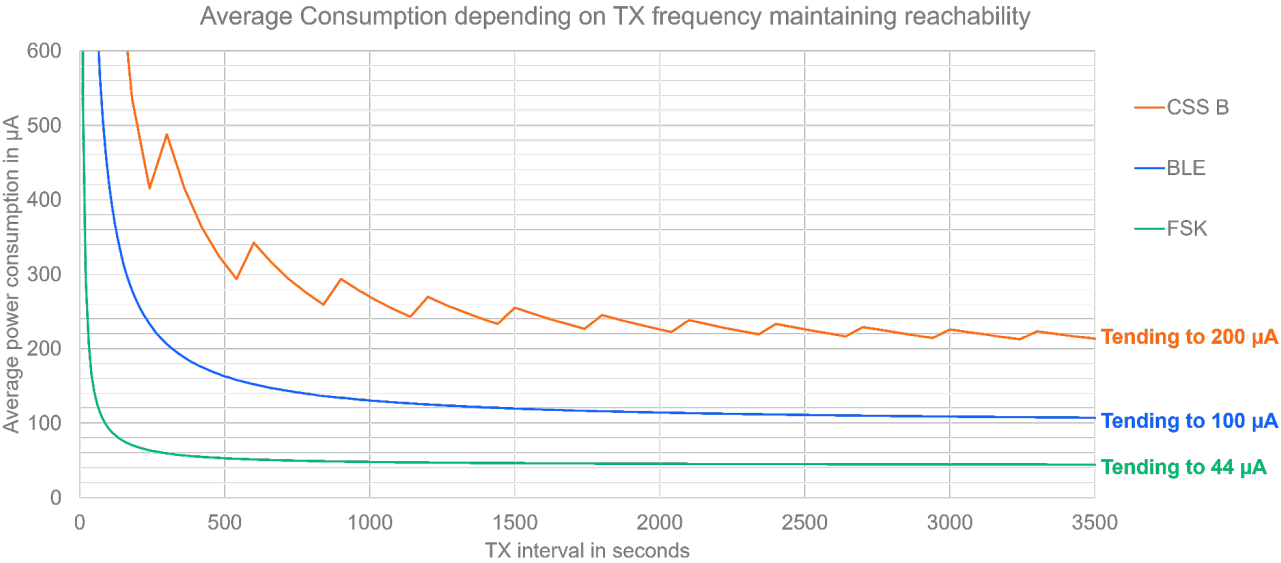
Power consumption was tested using the Simplicity Studio Energy Profiler. The test involved sending 19-byte packets to the cloud and receive an ACK from the cloud. The average power consumption was simulated on TX periodicity. The simulation was based on fundamental events such as TX, RX, and protocol-specific events. Based on those events, a loop was defined.

For more information about the average power consumption of fundamental events, see [Power Consumption Analysis](#).

Results

The graph below illustrates three distinct curves representing the average power consumption of BLE, FSK, and CSS as a function of transmission periodicity. The **green curve** represents **FSK** power consumption, which is notably efficient, stabilizing at around 44µA. The **blue curve** depicts **BLE** power consumption, which levels off at approximately 100µA. This is attributed to the BLE stack's continuous advertising within the Sidewalk context. To significantly reduce power consumption, the stack would need to be halted between transmissions. The **orange curve** corresponds to **CSS** power profile B, showing an average power consumption that approaches 200µA. This curve features slight inclines every 5 minutes, reflecting the keep-alive mechanism of CSS power profile B that transmits every 5 minutes. The power consumption for CSS could be reduced by opting for power profile A, which does not implement this keep-alive feature. In the three scenarios with each PHY, the reachability of the endpoint by the cloud at any given time was maintained.

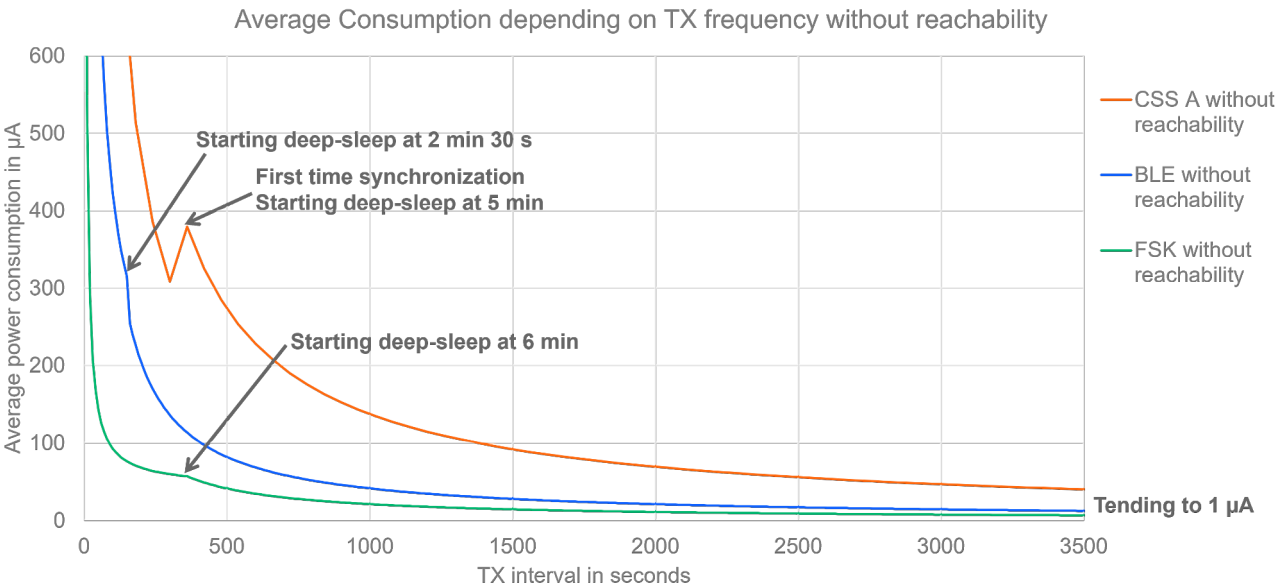
This graphical representation provides a clear comparison of the power efficiency of different protocols and configurations, highlighting the trade-offs between power consumption and communication frequency.



The graph below presents three curves that plot average power consumption against transmission periodicity, with all PHYs tending towards the ultra-low power consumption of EM4, which is 1 μA . This scenario assumes that RX reachability is not maintained and the Sidewalk stack enters sleep mode between messages. The analysis identifies the optimal intervals for each protocol to enter sleep mode to maximize power efficiency.

The **green curve** for **FSK** power consumption quickly approaches the EM4 current, with the optimal sleep interval identified at 6 minutes. The **blue curve** for **BLE** power consumption also rapidly trends towards EM4, with the best time to sleep being at 2 minutes and 30 seconds. The **orange curve** represents **CSS** power profile A, where the ideal point to sleep is at 5 minutes. Notably, the power consumption for CSS power profiles A and B (on the previous graph) remains identical up to the 5-minute mark. Beyond this point, power profile A requires a time synchronization before each transmission due to the absence of the keep-alive mechanism found in power profile B. However, the additional power consumed by this time synchronization is negligible compared to the savings achieved by stopping the stack and entering EM4 sleep mode.

This graph effectively illustrates the power-saving benefits of managing the sleep intervals of communication stacks, particularly in scenarios where maintaining RX reachability is not a priority.

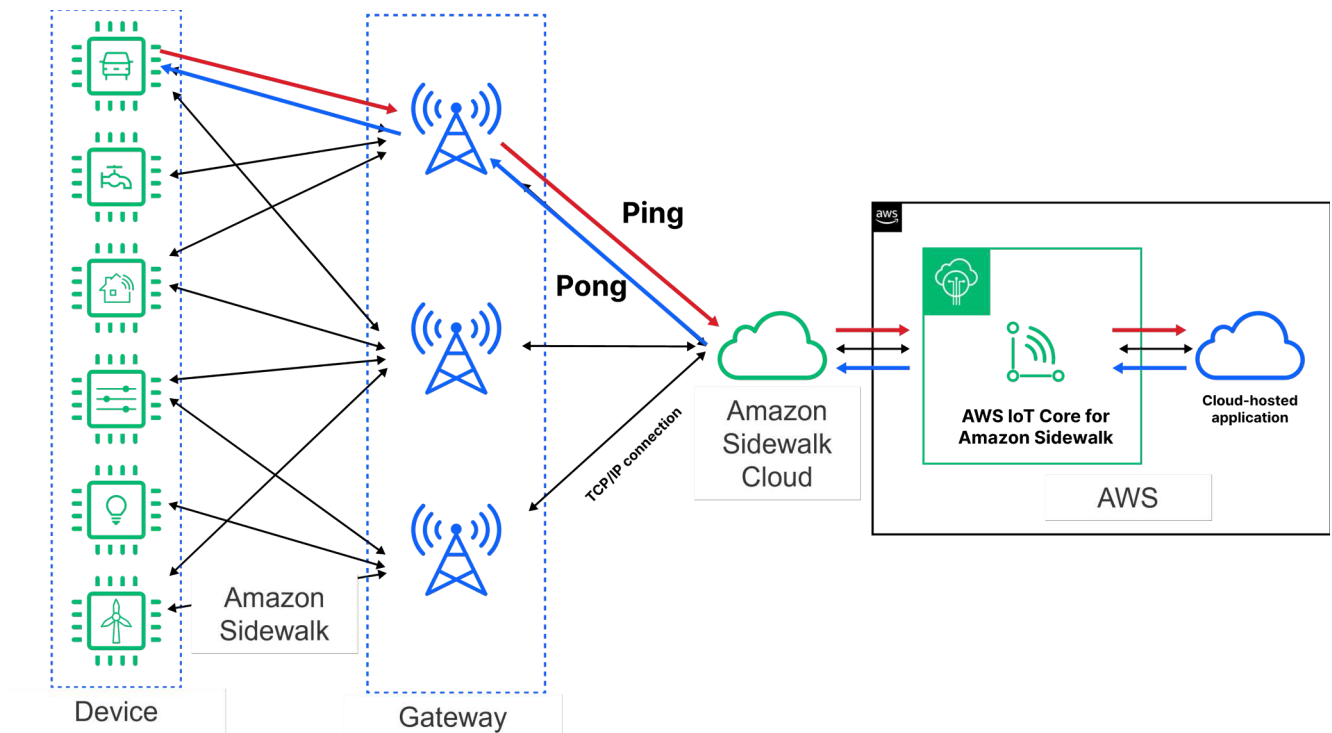


① INFO ①: You can find the **Amazon Sidewalk - SoC EM4 Sleep** sample application to show EM4 sleep in Amazon Sidewalk context in the [Silicon Labs Amazon Sidewalk Applications Github repository](#).

Network Latency

Testing Method

Latency was tested using a custom test application. The test involved sending 19-byte packets to the cloud and receiving an ACK from the cloud. Average latency was computed on each scenario by sending 500 messages. The message traveled from the endpoint to the GW and to the cloud and back.

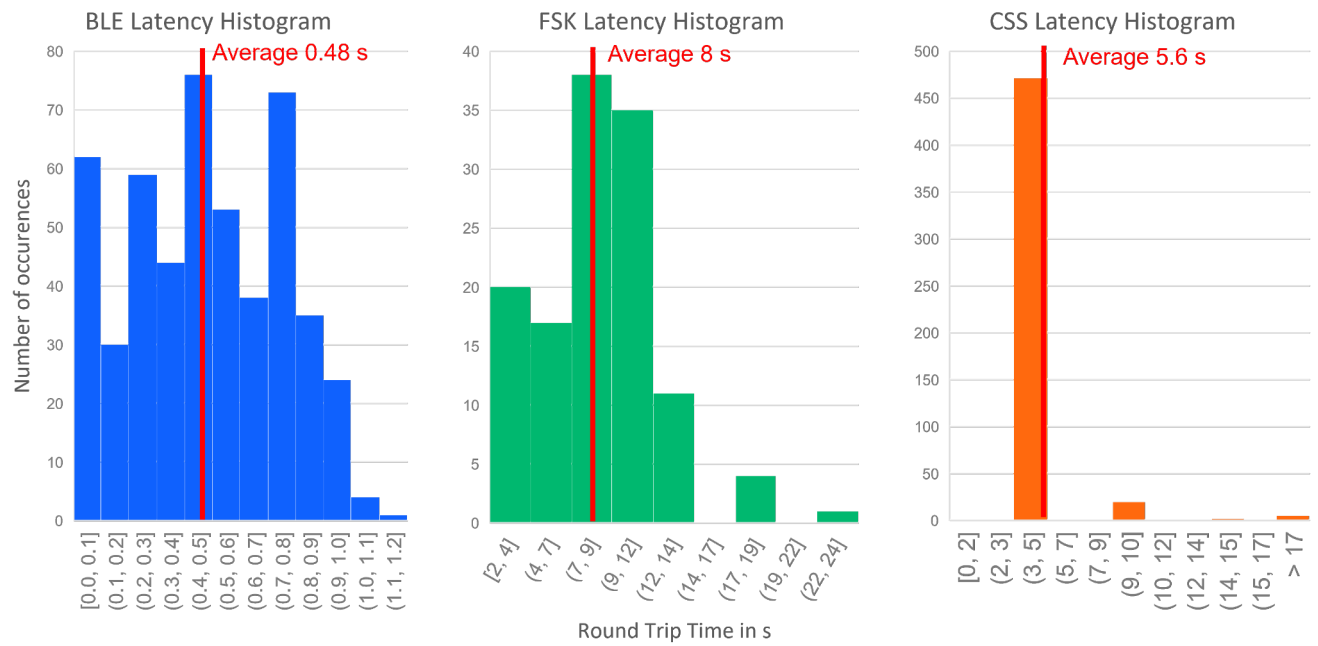


Results

The histograms visually represent the latency testing results for three different physical layers of the Amazon Sidewalk protocol: BLE, FSK, and CSS. Each histogram has latency on the horizontal axis and the number of messages for each latency value on the vertical axis.

- **BLE Latency Histogram:** Shows an average latency of around 500 ms, indicating a relatively quick response time for the BLE protocol.
- **FSK Latency Histogram:** Displays an average latency of approximately 8 seconds. This longer latency is attributed to the protocol's listening windows, which occur every 3 seconds. Consequently, the reception of messages is possible only at these intervals, leading to an average latency that aligns with multiples of the listening window periodicity.
- **CSS Latency Histogram:** Reveals an average latency of 5 seconds. Similar to FSK, this is due to the protocol's RX windows being spaced every 5 seconds, which dictates the frequency of reception opportunities.

The latency results for FSK and CSS highlight the significant impact of the protocols' reception opportunities on performance. Given that Amazon Sidewalk is an uplink-based protocol, while transmissions to the cloud may be swift, the response is limited by the scheduled listening opportunities inherent to the protocol's design.



Range Testing

Testing Methods

Range was tested for both FSK and CSS. For FSK, the test environment was an urban area with the gateway on the third floor. Points were chosen every 50 meters and at each point, messages were sent and the RSSI was retrieved from the answer. The test stopped when messages failed to send. For CSS, the range test was conducted using an Amazon Sidewalk Tracker.

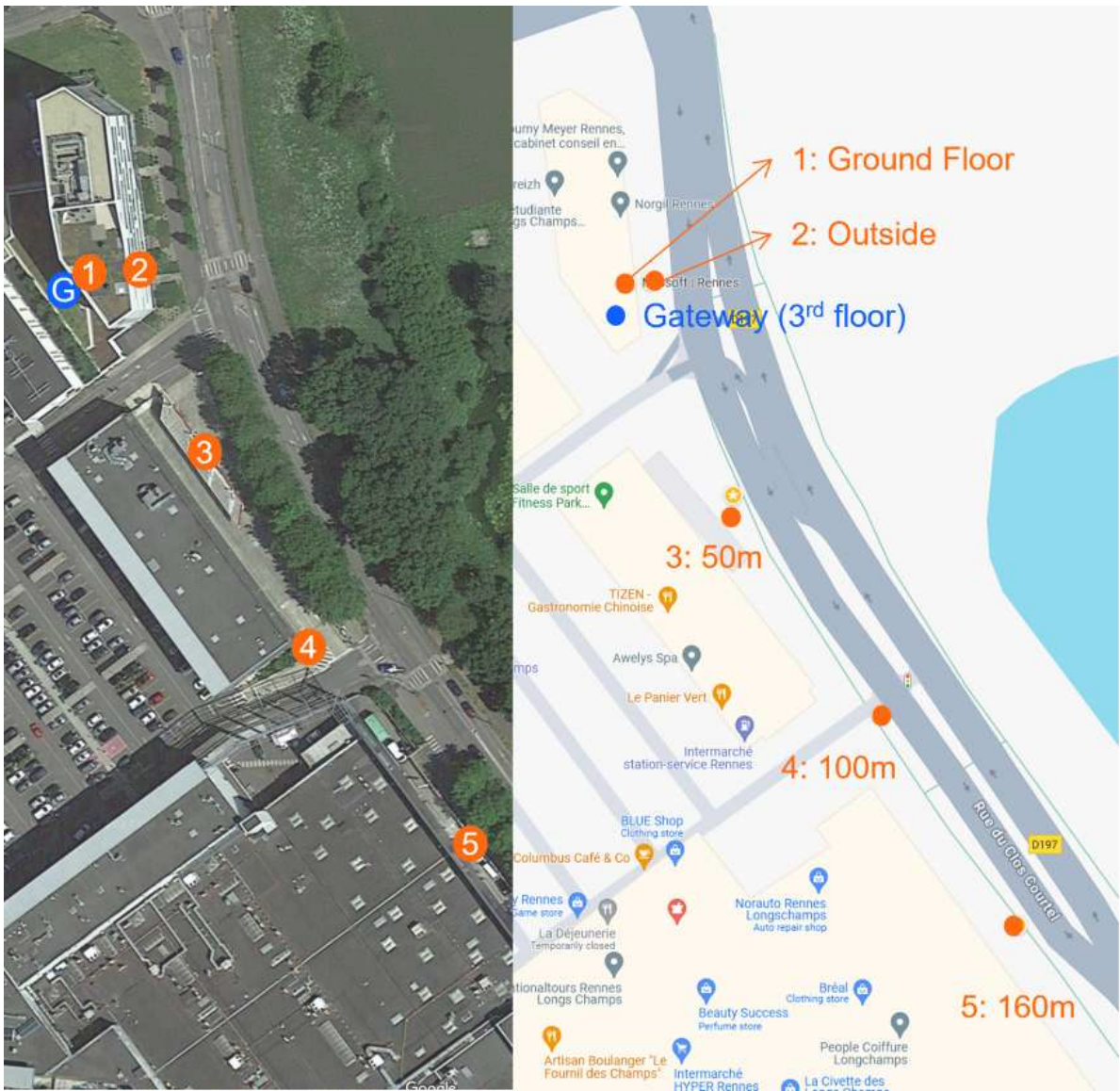
The range for both FSK and CSS was evaluated. In the case of FSK, the testing environment was an urban setting with the gateway positioned on the third floor. Measurement points were established at 50-meter intervals, and at each point, messages were sent and the RSSI was recorded from the response. The testing concluded when message transmission was unsuccessful. For CSS, the range testing was executed using an [Amazon Sidewalk Test Kit](#) and the coverage map was retrieved from Amazon tooling. As we are not using one of Silicon Labs dev kits but the Amazon Test Kit instead, the radio configuration can be different from the Silicon Labs reference platform used throughout this guide. However, radiating power should be pretty close so metrics are still relevant and representative of the CSS capabilities for Amazon Sidewalk.

The tests were conducted outside of the US to ensure only one gateway was in range of the test device.

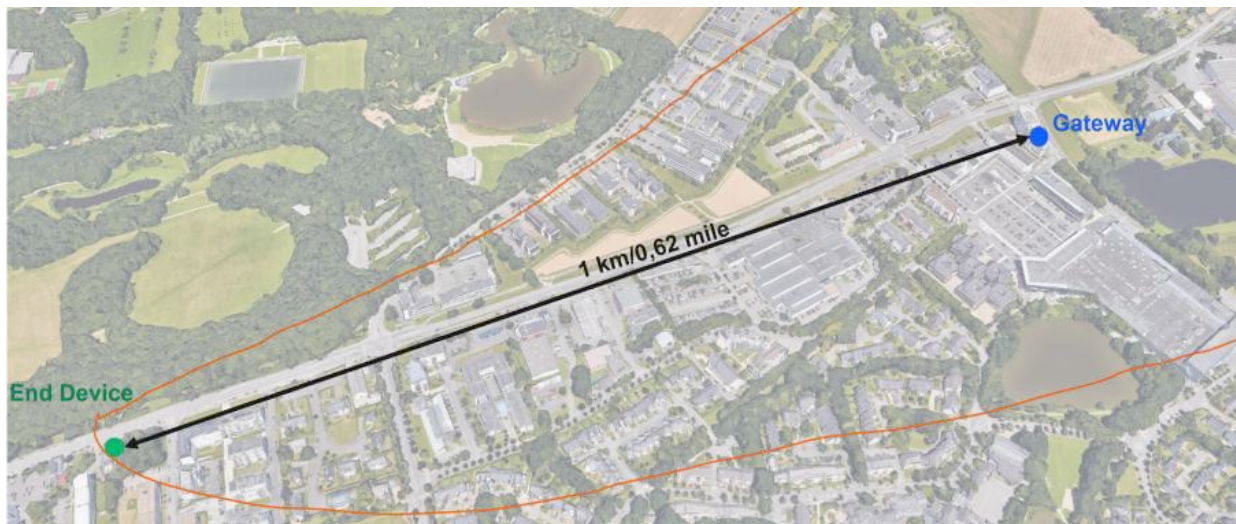
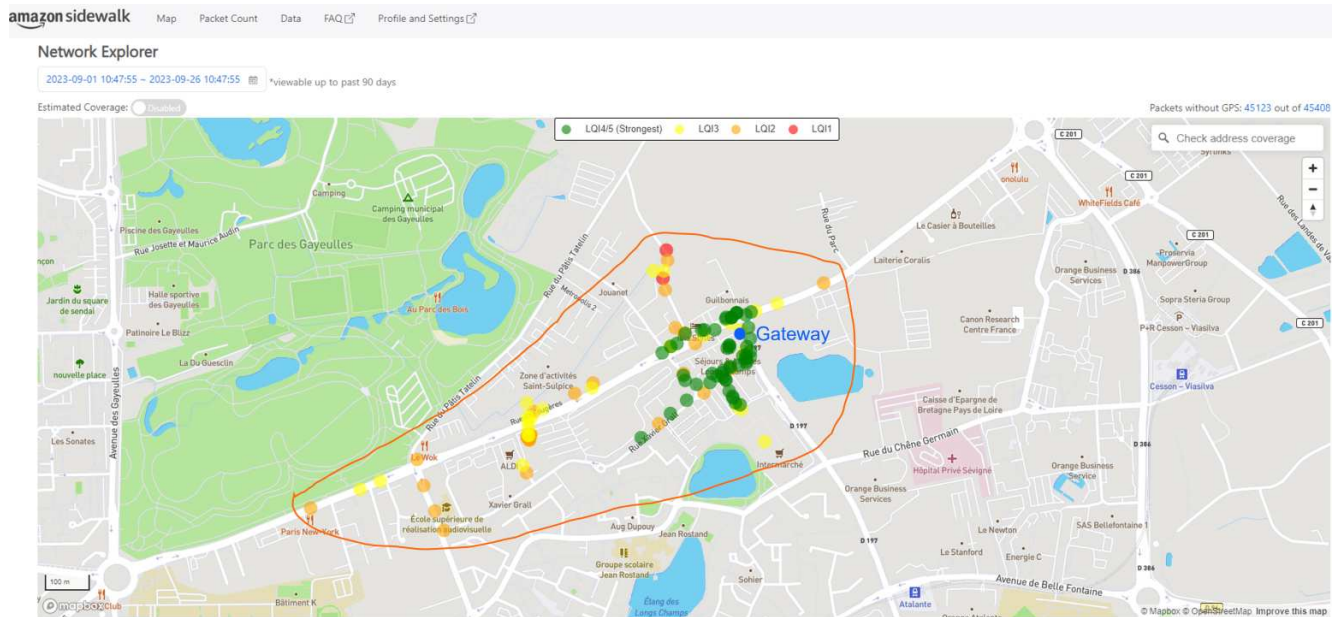


Results

For FSK, the result indicated a maximum range of 160 meters or 524 feet.



For CSS, the result indicated a maximum range of 1 km or 0.62 mile.



Throughput Testing

This is a test to evaluate the throughput of the file transfer mechanism through the Amazon Sidewalk network using the BLE link only.

Testing Methods

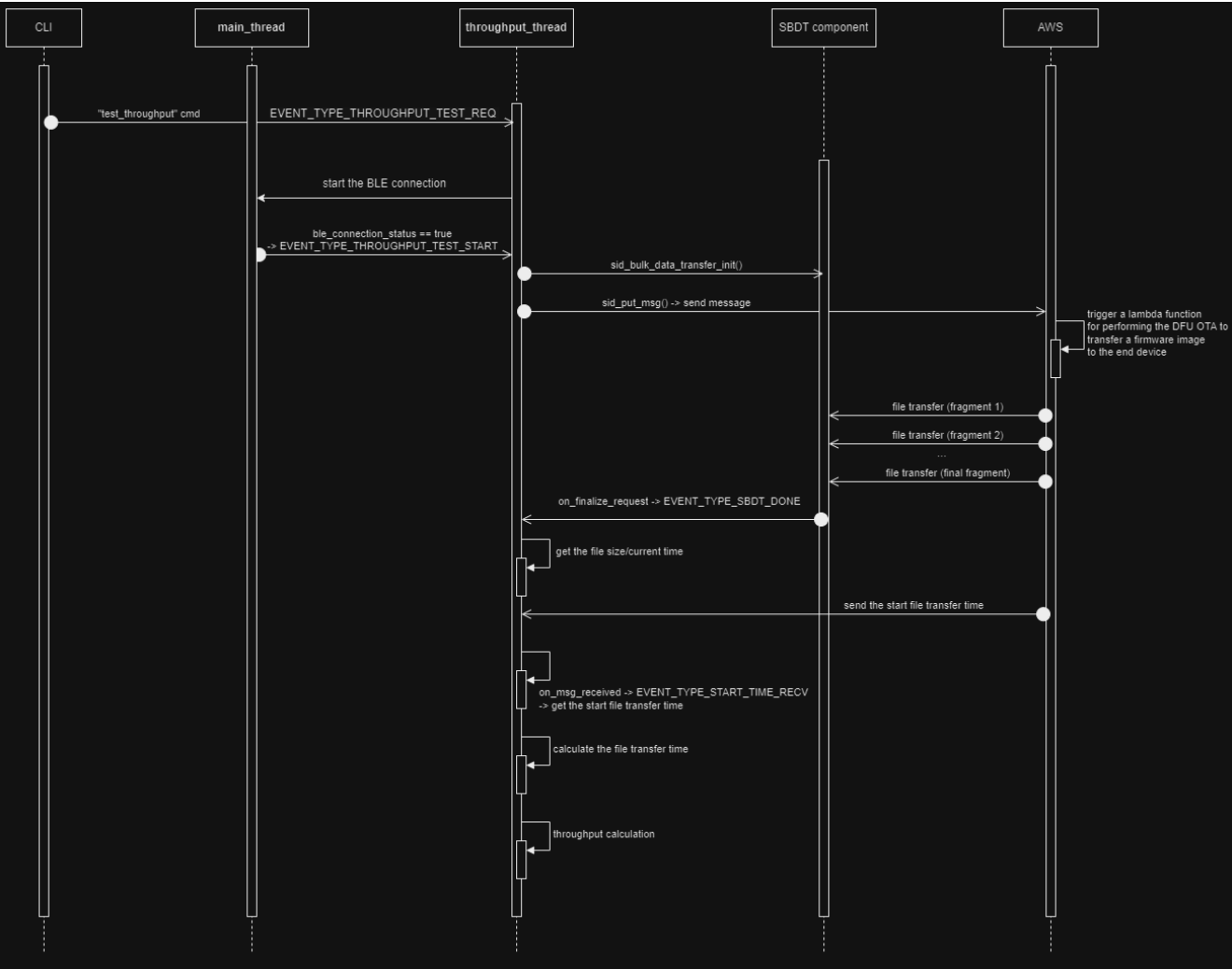
A custom Amazon Sidewalk application has been developed to perform this throughput test. This test uses the Amazon Sidewalk bulk data transfer (SBDT) functionality to transfer firmware files from AWS to the end device, and then calculate the throughput of these transfers over the Sidewalk network.

INFO: The SBDT feature is currently in alpha from Amazon and is not publicly available in AWS yet. Please note that the throughput performances with and without SBDT are comparable.

Once a BLE connection is established between the Gateway and the end device, a random message is sent from the end device to AWS to trigger a lambda function. This lambda function is implemented to use the Firmware update over-the-air

(FUOTA) to send a firmware image to an end device. The transfer file will be divided into smaller chunks of data and then gradually transferred to the device. The file transmission is complete once all the chunks are successfully received on the device side. The file size and the current GPS time will be recorded on the end device. Besides, to estimate the transfer time, the start transfer time is also sent to the device by the lambda function. Finally, after having both the file size and the file transfer time, the throughput value is calculated.

Refer to the sequence diagram below to understand more about the flow of the custom application for the throughput test.



Results

The throughput is tested with two different file sizes and with each file size, it is repeated two times. Indeed, the throughput results through the Amazon Sidewalk network (using the BLE link) are significantly lower than the throughput that can be achieved using pure BLE. It is due to the bandwidth and monthly data usage limitation in the Sidewalk network (read more about it in the **"Appendix" section - question 7** of the **"Amazon Sidewalk Privacy Security"** documentation [here](#)). The throughput results over the Amazon Sidewalk network using the BLE link along with the comparison with the pure BLE are recorded in the table below.

Test	File size (KB)	Throughput (B/s)		Transmission time (s)	
		Amazon Sidewalk	Pure BLE	Amazon Sidewalk	Pure BLE
Test 1	168	113.88	18900	1483	71
Test 2	168	196.61	22700	859	59
Test 3	467	125.63	21500	1483	172
Test 4	467	156.81	20300	859	183

Summary

The performance study on the Amazon Sidewalk protocol concludes with insightful findings for each physical layer. Each layer presents unique advantages and trade-offs, making them suitable for different applications within the Amazon Sidewalk ecosystem. BLE is characterized by its indoor range and offers highly customizable latency and power consumption settings, making it suitable for household appliances. FSK's range is optimal just outside the front door and is extremely power-efficient for battery-powered devices. Notably, the trade-off between latency and power consumption for FSK is influenced by the selected value for the number of RX windows parameter, allowing for customization based on the use case, with outdoor smart lighting being a versatile application. CSS supports long-range devices, with Power Profile B being ideal for long-range applications that require regular downlinks and have fewer concerns for power efficiency, such as industrial sprinklers. Conversely, Power Profile A is better suited for uplink-based use cases that demand high power efficiency, like sensors or location trackers. Each layer's attributes make them uniquely fit for various applications within the Amazon Sidewalk network.

Radio	BLE	FSK	CSS A	CSS B
Range	☀ 30 m	160 m	🌿 1 km	🌿 1 km
Latency	🌿 Good latency	Average latency	☀ Around 5 seconds but only after an uplink	🌿 Around 5 seconds
Power Consumption	Average (Highly configurable)	🌿 Very low	🌿 Low	☀ High
Configuration	🌿 Highly configurable	☀ Few parameters can be changed	☀ Few parameters can be changed	☀ Few parameters can be changed
In a Nutshell	Configuration is a tradeoff between power efficiency and latency Range suggests inside the house applications	Most versatile with good tradeoff between power consumption, latency and range	Specialized for uplink-based communication Uplinks should not be too frequent to keep low power consumption Good for trackers and sensors	For long range and regular downlinks Good for long range command when battery life is less of a concern
Example Use Case	White goods	Outdoor smart lighting	Location tracker	Industrial sprinkler

Going Further

To go further in evaluating Amazon Sidewalk performances, several things can be tried by changing the configuration or experimenting with specific features.

For FSK, a fixed message could be sent periodically with an answer from the cloud, and with a change to the number of RX windows between beacons. This would allow a view of the impact of this parameter on latency and power consumption. The BLE stack is highly configurable and several parameters can be changed like connection interval and peripheral latency. This would be especially interesting for CSS, as TX power consumption is very high. In addition to those existing parameters, Amazon introduced two major features in Sidewalk SDK 1.16 (Sidewalk extension 2.0.0): multi-link and bulk data transfer. The multi-link feature allows switching between available radio layers depending on reception quality. The cost of this radio switch could be evaluated in terms of range, latency, and power consumption. This study does not include throughput testing, but it would be an interesting angle to add. On BLE, it could be measured by leveraging the bulk data transfer feature.

On the gateway side, several configurations and corner-cases could be tried to stress-test the performances of the gateway, especially the impact on the latency depending on the number of devices using and connecting to the gateway.

Multiprotocol with Sidewalk

Multiprotocol with Sidewalk

You can integrate different protocols with Amazon Sidewalk. This page provides a list of tutorials and examples created by Silicon Labs to showcase Amazon Sidewalk alongside other protocols.

Sidewalk Alongside BLE

In the Amazon Sidewalk extension, there is a sample application that demonstrates concurrent operation of Amazon Sidewalk and standard BLE. The sample application is called **Amazon Sidewalk - SoC Dynamic Multiprotocol Light** and demonstrates a light bulb that can be switched via Bluetooth or Amazon Sidewalk (BLE or FSK radio layer).

Sidewalk and BLE Double Advertising

This page describes how to add standard BLE advertising alongside your Amazon Sidewalk application.

INFO: By default, the standard BLE advertising will ONLY be available if your Sidewalk application is also running the BLE link.

Set Up the Project

You can use the Hello Neighbor sample app as a starting point. Make sure your part supports the Bluetooth link. Refer to [Prerequisites](#) to learn which platforms are suitable.

First, modify the number of advertising channels authorized. By default, only one advertising set is allowed. In file `config/sl_bluetooth_advertiser_config.h` set `SL_BT_CONFIG_USER_ADVERTISERS` to 2.

Both BLE attributes (e.g. change the advertising name) can be configured in different places:

BLE Type	Configuration
Sidewalk BLE	In c file : <code>sidewalk_x.x.x/component/ble_subghz/radio/ble/app_ble_config.c</code>
Classic BLE	In the GATT configuration : <code>config/btconf/gatt_configuration.btconf</code>

INFO: If the `gatt_configuration.btconf` is missing, you need to add the `Static GATT Database and Configuration` component to generate this file.

Enable Extra Advertising

To enable the standard BLE advertising based on your GATT configuration, the code snippet below should be added to your application.

```
// The advertising set handle allocated from Bluetooth stack.
static uint8_t advertising_set_handle = 0xff;
void start_standard_ble_advertising()
{
    sl_status_t sc;
    // Create new handle
```



```
app_assert_status(sc); // Generate data for advertising
sc = sl_bt_legacy_advertiser_generate_data(advertising_set_handle,
                                           sl_bt_advertiser_general_discoverable); app_assert_status(sc); // Start advertising
sc = sl_bt_legacy_advertiser_start(advertising_set_handle,
                                   sl_bt_advertiser_connectable_scannable); app_assert_status(sc); app_log_info("Started standard BLE advertising.");}
```

This function should be called **after** the Sidewalk BLE stack initialization to prevent an error. The Sidewalk BLE is initialized with the `sl_bt_start` function. In the SoC Hello Neighbor sample application, it can be added in the `init_and_start_link()` function after the `sl_bt_start` call:

```
#if defined(SL_BLE_SUPPORTED)
// Start advertising over standard ble after Sidewalk BLE is initialized
if (link_type_to_link_mask(SL_SIDEWALK_LINK_BLE) == link_mask) {
    start_standard_ble_advertising();
}
#endif
```

INFO: If `sl_bt_advertiser_create_set` is failing, make sure that [you have enough advertiser sets](#). Silicon Labs recommends only calling this function once after the BLE link have been initialized.

Now that the two BLE channels are initialized, you need to have a way to differentiate the origin of the packet. You can do that by comparing the advertising handle sent in the BT headers. If you use the sample provided above, you can see that the advertising handle is stored in the variable `advertising_set_handle`.

The following snippet shows how to retrieve the handle from an incoming BLE connection and thus know the packet source:

```
// sl_bt_on_event callback
#include "sl_bluetooth.h"

void sl_bt_on_event(sl_bt_msg_t *evt)
{
    sl_status_t sc;
    bd_addr address;
    uint8_t address_type;

    // Handle stack events
    switch (SL_BT_MSG_ID(evt->header)) {
        // ...

        // -----
        // This event indicates that a new connection was opened.
        case sl_bt_evt_connection_opened_id:
            if (evt->data_evt_connection_opened.advertiser == advertising_set_handle) {
                app_log_info("BLE Connection opened from side application");
            } else {
                // If we don't have any info concerning the advertised ID we can assume that it is from sidewalk
                app_log_info("BLE Connection opened from sidewalk");
            }
            break;
        // ...
    }
}
```

Sidewalk and Bluetooth FUOTA

The Bluetooth FUOTA (Firmware Upgrade Over-the-Air) requires a bootloader. Follow this section to install and configure a bootloader. For more information about bootloaders, see [Bootloading](#).

INFO: Silicon Labs highly recommends that you perform the OTA in a [non-sidewalk BLE channel](#).

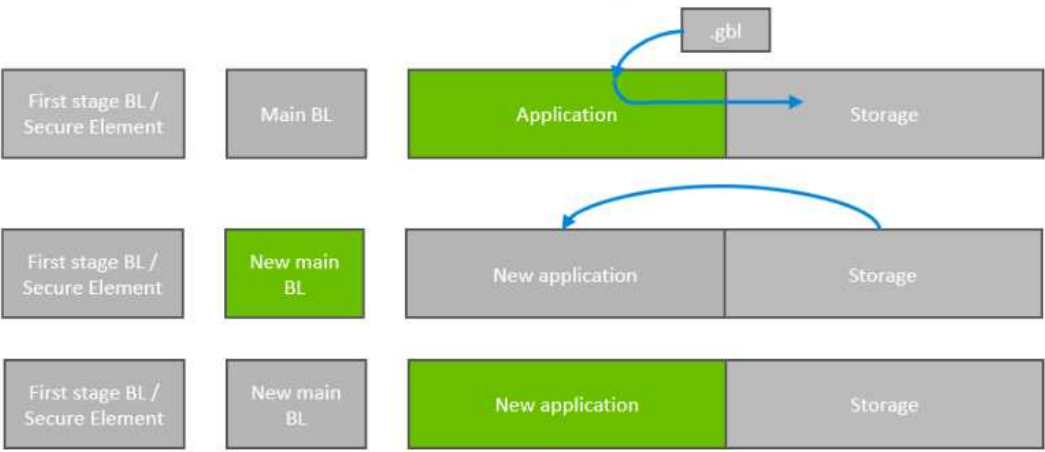
Two methods are available for updating your application. An overview of both OTA methods are available below:

OTA Method	Pros	Cons
Dual Bank	Resilient to transfer failures	Need more storage space in the application flash (we need twice the size of the application)
In place	No backup if OTA fails (retries until success)	Requires less extra space in the Flash memory

Dual-Bank Application Firmware Upgrade

This OTA method downloads the new firmware in a storage space alongside the application. This ensures that the device doesn't enter an incorrect state if something goes wrong during the transfer. You can leverage double advertising to use BLE to download the firmware image.

INFO: If you really want to implement OTA though the Sidewalk BLE channel, we recommend you to use In-Place application to prevent the Sidewalk protocol from interfering with the download of the new application.



The complete schema can be found in section 4.2, *Bootloader Upgrade on Application Bootloaders with Storage*, in [UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher](#).

Enable Dual-Bank Application OTA in your Sidewalk Application

INFO: To install a component in your project, you need to open it with Simplicity Studio. Open the `.slcp` file and select the **SOFTWARE COMPONENT** tab. You can search the component you need and click **Install**.

- 1. Install the component **Application OTA DFU** if not already done.
- 2. Erase the flash memory.

Flash the bootloader.

- Search for the **Bootloader - SoC Internal Storage** project. Select the project with the correct storage size. The EFR32xG24B have a storage size of 1536kB.
- More information on this bootloader can be found in section 7.3.7, *Internal Storage Bootloader*, in [UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher](#).
- You may need to adapt the storage size to your application size. It should not overlap with the NVM3 section. In the Hello Neighbor sample, the storage size needs to be set to **x094000** (instead of **x0B4000**). This value can be configured in the **Common storage** component in the **bootloader project** (not the sidewalk project).

4. Flash the sidewalk application.

5. [Flash the Virtual AWS device](#):

- Open the following file with Simplicity Studio: `config/sidewalk/sidewalk.asconf`.
- Go to **End device**.
- Connect to your device.
- Create a new Virtual AWS device if needed and click **FLASH TO DEVICE**.

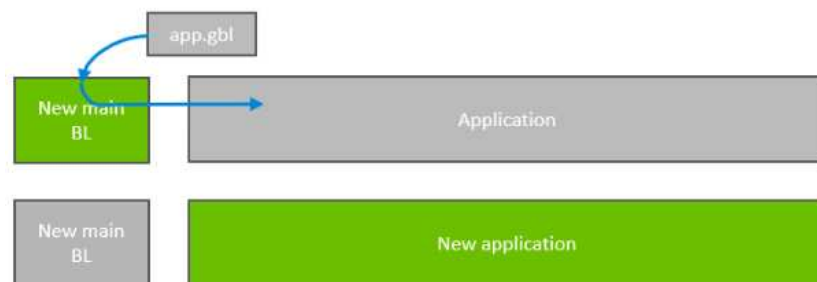
You should then be able to flash your new application (e.g. through the EFR application) by connecting to the BLE channel. Instructions to create a .gbl file containing your new application and flash it through the EFR application can be found in the [bluetooth documentation](#).

In-Place OTA

This OTA method makes your application restart in a special state and download the new version of the application. You should leverage double advertising to use BLE to download the firmware image.

INFO: By default, the Sidewalk BLE MAC address type is set to `SID_BLE_CFG_MAC_ADDRESS_TYPE_RANDOM_PRIVATE_RESOLVABLE`, which is not compatible with this OTA method. This method needs `SID_BLE_CFG_MAC_ADDRESS_TYPE_PUBLIC` MAC address type which can be changed in structure `sid_ble_config_t ble_cfg` of file `component/ble_subghz/radio/ble/app_ble_config.c`.

The new version overwrites the original version. If something goes wrong during the transfer, the application is not functional anymore. If you have space in your Flash memory, consider using the Dual-Bank method.



The complete schema can be found in section 4.1, *Bootloader Upgrade on Bootloaders with Communication Interface (Standalone Bootloaders)*, in [UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher](#).

Enable In Place OTA in your Sidewalk Application

INFO: To install a component in your project, you need to open it with Simplicity Studio. Open the .slcp file and go to **SOFTWARE COMPONENT** tab. You can search the component you need and click **Install**.

1. Install the component **In-Place OTA DFU** if not already done.
2. Erase the flash memory.
3. Flash the bootloader / apploader:
 - Search for the **Bootloader - SoC Bluetooth AppLoader OTA DFU** project.
4. Flash the Sidewalk application.
5. [Flash the Virtual AWS device](#):
 - Open the following file with simplicity studio : config > sidewalk > sidewalk.asconf.
 - Go to **End device**.
 - Connect to your device.
 - Create a new Virtual AWS device if needed and click **FLASH TO DEVICE**.

You should then be able to flash your new application (e.g. through the SiConnect mobile application) by connecting to the BLE channel. Instructions to create a .gbl file containing your new application and flash it to the SiConnect mobile application can be found in the [bluetooth documentation](#).

⚠ WARNING ⚠: If you are using Sidewalk BLE to perform the OTA (which is not recommended), make sure that your BLE address is configured to `SID_BLE_CFG_MAC_ADDRESS_TYPE_PUBLIC` (configuration is done in `sidewalk_x.x.x\component\ble_subghz\radio\ble\app_ble_config.c`). Otherwise, the apploader will not be able to find the device when it restarts in OTA mode.

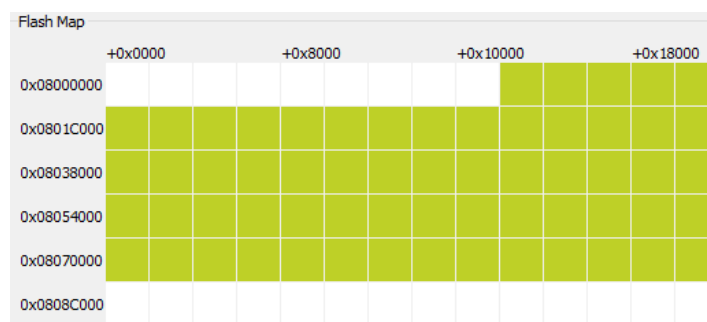
Troubleshoot

My sidewalk application doesn't start

Make sure that you followed the order described above.

If you still have issues:

1. Erase the flash.
2. Flash your sidewalk application.
3. Using Simplicity Commander, you can see the main flash contents. You should see some free space at the beginning of the main flash:



4. Flash the bootloader.
5. Check that the bootloader is present on the flash.

① NOTE ①: In step 3 if your Sidewalk application is flashed at the beginning of the main flash, make sure that the right OTA component is installed: either **In-Place OTA DFU** or **Application OTA DFU** (Double Bank OTA).

The screen shows the reserved space for the **Bootloader - SoC Bluetooth AppLoader OTA DFU** used in the In-Place OTA (**In-Place OTA DFU** component). The reserved space is smaller for the **Bootloader - SoC Internal Storage** used in Double Bank OTA (**Application OTA DFU** component).

Err -8 or no device info found

Make sure that you flash your virtual AWS device info **after** flashing the bootloader and the Sidewalk application.

Troubleshooting

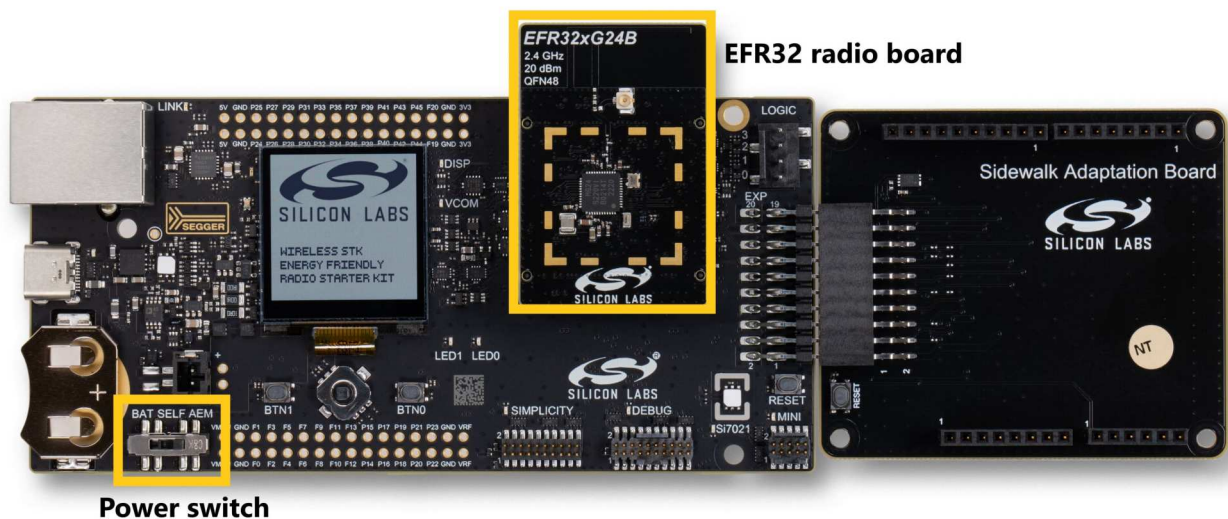
Troubleshooting

J-Link could not connect to the device (EFR32)

When using the Wireless Starter Kit main board (WSTK) along with a radio board, you should be able to connect with J-Link RTT Viewer using the parameters in the [dedicated section](#). Your EFR32 should also be detected by Simplicity Studio 5.

If your EFR32 is not detected or you have difficulties connecting with J-Link RTT Viewer, you can check two points:

- Your WSTK power switch is on **AEM** mode.
- Your radio board is plugged in correctly (see the following)



CloudFormation Errors

Permission Denied

While executing the CloudFormation command to deploy your stack, you have an error showing missing authorization as follows:

<your user> is not authorized to perform: cloudformation:<some action> on resource aws:cloudformation:us-east-1:<some resource> because no identity-based policy allows the cloudformation:<some action> action

It is probably because your IAM user associated to your CLI does not have sufficient permissions. You should ask the person managing your AWS account to help you on this task.

Failed to Create Stack

CloudFormation stack creates objects and every user inside the same IAM policy has access to those objects. The stack only needs to be created once, and can be used for all your devices. If you try to deploy the CloudFormation stack but some objects already exist with the same name, your stack will not deploy with results like the following:

Waiting for changeset to be created.
Waiting for stack create/update to complete

Failed to create/update the stack. Run the following command to fetch the list of events leading up to the failure

```
aws cloudformation describe-stack-events --stack-name NameOfYourStack
```

To get more details on this error, execute this command: `aws cloudformation describe-stack-events --stack-name NameOfYourStack`

This command displays all the actions attempted by CloudFormation. While reading the error messages, if you see an error log about an object that already exists then you should check what objects are already created inside your AWS account.

As an example you can see in the following log that the creation of `CFSRepublishLambdaRole` failed (value of `ResourceStatus` field) because it already exists (value of `ResourceStatusReason` field):

```
{
  "StackId": "arn:aws:cloudformation:us-east-1:78468574544:stack/NameOfYourStack/4a54e4-4583-5435-f548-ad546584dd58",
  "EventId": "CFSRepublishLambdaRole-CREATE_FAILED-2022-09-21T14:18:19.899Z",
  "StackName": "NameOfYourStack",
  "LogicalResourceId": "CFSRepublishLambdaRole",
  "PhysicalResourceId": "",
  "ResourceType": "AWS::IAM::Role",
  "Timestamp": "2022-09-21T14:18:19.899000+00:00",
  "ResourceStatus": "CREATE_FAILED",
  "ResourceStatusReason": "CFSRepublishLambdaRole already exists in stack arn:aws:cloudformation:us-east-1:560019425582:stack/SidewalkStack/a58745f-5458-5742-125e-65a8b45a25",
  "ResourceProperties": "{\"MaxSessionDuration\": \"3600\", \"RoleName\": \"CFSRepublishLambdaRole\", \"Description\": \"Allows IoT to call AWS services on your behalf.\", \"Policies\": [{\"PolicyName\": \"CFSRepublishPolicy\", \"PolicyDocument\": {\"Statement\": [{\"Action\": [\"iot:*\", \"sqs:*\", \"iotwireless:*\", \"logs:*\", \"Resource\": [\"*\"]\", \"Effect\": \"Allow\"}] }}, {\"AssumeRolePolicyDocument\": {\"Version\": \"2012-10-17\", \"Statement\": [{\"Action\": [\"sts:AssumeRole\"], \"Effect\": \"Allow\", \"Principal\": {\"Service\": [\"lambda.amazonaws.com\"]}, \"Sid\": \"\"}]}\"}
}
```

You should delete the duplicated resources or rename them.

Registration Errors

Failing During Device Registration

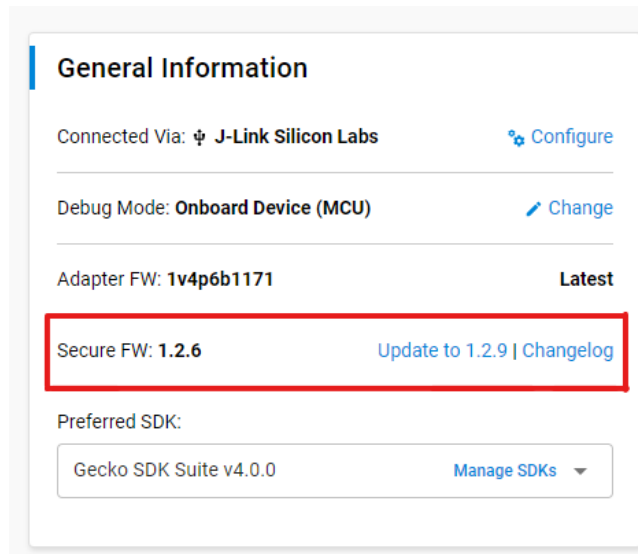
Once your device is detected by the registration script, it starts to exchange messages in order to register. During this message exchange, if you see the following errors on your endpoint logs, then your Secure Element Firmware is probably outdated.

```
<error> GET_DEVICE_ECDH_SIG: 3
<error> MASK NOT FOUND
<info> data_send send -8
<error> Dispatcher: Message handler returned unexpected result --8
<error> MASK NOT FOUND
```

The Amazon Sidewalk examples require that the EFR32 uses a minimal version of Secure Element firmware depending on the radio board MCUs. See the table below to check your version:

Radio Board MCUs	SE minimal version
MG21	1.2.9
MG24	2.1.7
KG100S	1.2.9

You can verify this version in the Simplicity Studio 5 **Launcher** perspective. If the EFR32 is using an older firmware version, update it. If you do not see **Update to 1.2.9** (as shown below), disconnect and re-connect the board until it appears.



Time Synchronization

Once your device is registered, time synchronization between the gateway and the endpoint should take place within a few minutes. If the device does not attempt to synchronize with the gateway, check that your Amazon Echo device is connected to the Internet and localized in the US, then try rebooting your gateway and your endpoint.

Amazon Sidewalk - SoC Hello Neighbor Errors

When trying out the Hello Neighbor application with FSK radio layer, you may see logs about failing Disco module that resemble the following logs (the device is fully registered and attempts to discover the gateway):


```

<info> [00132619] App - sidewalk status changed: 1
<info> [00132619] App - registration status: 0, time sync status: 1, link status: 0[00132631] <info> RTC compensation is not supported.
[00132632] <info> swi thread created
[00132632] <info> Set region 1, country: US, num cfg: 2
[00132673] <info> P2P chnl loaded: 7
[00132673] <info> PAN ID loaded:
[00132673] <info> B9 80 96 21 00
[00132675] <info> Def mcast retries: 1
[00132675] <info> Pairing state loaded: 0
[00132675] <info> Config version 4
[00132675] <info> init_chnl_seed, chsid:9, seed:FFFFFFA610000
[00132676] <info> rnet_mac_disco_init state = 0, req_auth = 0, cur_ev = 00000000
[00132676] <info> min ch symbols ms 151 bcn time ms 167
[00132677] <info> Starting GWD Process in: PASSIVE_SYNC/FFS_MODE
[00132677] <info> rnet_mac_disco_start_sampling state = 1, req_auth = 0, cur_ev = 00000000 f = 00052AC1 r = 0
[00132678] <info> Disco sampling start SUCCESS! state = 2, req_auth = 0, cur_ev = 00000000
[00132679] <info> 900MHz MAC Init: PRB:1 BCN:1 HDR:1, HDR_LORA:0 LDR:0
[00132680] <info> rnet_sec: Loading Setting (key=0xa) not on flash!
[00132682] <error> rnet_sec: File content for 2 mode cannot be trusted!
[00132683] <info> rnet_sec: Loading Setting (key=0x9) not on flash!
[00132685] <error> rnet_sec: File content for 1 mode cannot be trusted!
[00132686] <info> Ring-Net initialized
[00132686] <info> Disco stop sampling success. Time elapsed = 7 ms
[00132686] <info> [MET] B:0 N:0
[00132687] <info> rnet_mac_disco_init state = 1, req_auth = 0, cur_ev = 00000000
[00132687] <info> Starting GWD Process in: PASSIVE_SYNC/FFS_MODE
[00132688] <info> rnet_mac_disco_start_sampling state = 1, req_auth = 0, cur_ev = 00000000 f = 00052AC1 r = 0
[00132688] <info> Disco sampling start SUCCESS! state = 2, req_auth = 0, cur_ev = 00000000
[00132689] <info> Disco active. LSC=0, symb=7490, rx_to_ms=151
[00132690] <info> [MET] B:0 N:0
[00132857] <warning> Disco submodule ended prematurely! ch = 0
[00133025] <warning> Disco submodule ended prematurely! ch = 1
[00133193] <warning> Disco submodule ended prematurely! ch = 2
[00133362] <warning> Disco submodule ended prematurely! ch = 3
[00133530] <warning> Disco submodule ended prematurely! ch = 4

```

This error is generally linked to a missing or incorrectly plugged-in Semtech board.

For more troubleshooting advice, check out the [FAQ on Silicon Labs community forum](#).

PAL API Reference

Overview

PAL API Reference

The Sidewalk Platform Abstraction Layer (PAL) API provides a comprehensive set of interfaces and utilities for developing applications that leverage the Amazon Sidewalk network. This documentation covers various modules and components of the SDK, detailing their functionalities, key features, and usage guidelines. The API is organized into several groups, each focusing on specific aspects of the SDK, such as BLE adaptation, security, storage, and peripheral interfaces. By following this reference, developers can effectively integrate and utilize the Sidewalk SDK in their projects.

Interfaces

Interfaces

Interfaces

The Interfaces module provides a set of standardized interfaces that facilitate communication and interaction between different components of the Sidewalk SDK. These interfaces define the methods and structures required for various functionalities, ensuring consistency and interoperability across the SDK. By customizing interfaces, developers can better fit their needs according to their use case or the specifics of their custom hardware.

Modules

[SAL](#)

SAL

SAL

Sidewalk Abstraction Layer

The Sidewalk Abstraction Layer (SAL) provides interfaces, such as cryptography, storage, timers, and peripheral interfaces.

Modules

[Common Interface](#)

[Critical Region Interface](#)

[Logging Interface](#)

[Peripheral Interfaces](#)

[Radio Interfaces](#)

[SWI Interface](#)

[Security and Crypto](#)

[Storage Interface](#)

[Timer Interfaces](#)

Common Interface

Common Interface

Provides common and generic platform interface code not specific to one PAL module.

Modules

[Type definitions](#)

Functions

- sid_error_t

sid_pal_common_init

(const platform_specific_init_parameters_t *platform_init_parameters)

Implements a platform generic initialization function.
- sid_error_t

sid_pal_common_deinit

(void)

Implements a platform generic deinitialization function.

Function Documentation

sid_pal_common_init

```
sid_error_t sid_pal_common_init (const platform_specific_init_parameters_t * platform_init_parameters)
```

Implements a platform generic initialization function.

Parameters

Type	Direction	Argument Name	Description
const platform_specific_init_parameters_t *	[in]	platform_init_parameters	pointer to platform specific parameters.

This function is only implemented on platforms that require additional vendor specific initialization routines. It provides a generic entry point to the platform implementation of the sid_pal components and is intended to be called at start of day.

sid_pal_common_deinit

```
sid_error_t sid_pal_common_deinit (void )
```

Implements a platform generic deinitialization function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is only implemented on platforms that require additional vendor specific deinitialization routines. It provides a generic entry point to the platform implementation of the sid_pal components.

Type definitions

Type definitions

Modules

[platform_parameters_t](#)

platform_parameters_t

Platform parameters structure.

This structure defines the platform parameters that are used for initializing the SID PAL (Platform Abstraction Layer).

Public Attributes

- sid_pal_mfg_store
_region_t

mfg_store_region
- platform_specific_i
nit_parameters_t

platform_init_parameters

Public Attribute Documentation

mfg_store_region

```
sid_pal_mfg_store_region_t platform_parameters_t::mfg_store_region
```

Manufacturing store region.

platform_init_parameters

```
platform_specific_init_parameters_t platform_parameters_t::platform_init_parameters
```

Platform specific initialization parameters.

Critical Region Interface

Critical Region Interface

Provides API's used by components of the Sidewalk SDK to solve concurrency issues.

Functions

void [sid_pal_enter_critical_region\(\)](#)

Code executed between calls of function declared below will be protected from interruption from different thread or ISR context.

void [sid_pal_exit_critical_region\(\)](#)

Implements enabling of all hardware and software interrupts that were previously disabled by enter function Code executed between calls of function declared below will be protected from interruption from different thread or ISR context.

Function Documentation

sid_pal_enter_critical_region

```
void sid_pal_enter_critical_region ()
```

Code executed between calls of function declared below will be protected from interruption from different thread or ISR context.

Note

- Implementation is actually disabled IRQ's, so remember, amount of code and execution time of code between calls of enter and exit functions should be as minimal as possible

Implements disabling of all hardware and software interrupts

sid_pal_exit_critical_region

```
void sid_pal_exit_critical_region ()
```

Implements enabling of all hardware and software interrupts that were previously disabled by enter function Code executed between calls of function declared below will be protected from interruption from different thread or ISR context.

Note

- Implementation is actually disabled IRQ's, so remember, amount of code and execution time of code between calls of enter and exit functions should be as minimal as possible

Logging Interface

Logging Interface

Logging Interface

Interface for log for sidewalk SDK. For more details, please see [Logging Documentation](#)

Modules

- [sid_pal_log_buffer](#)
- [Type definitions](#)

Functions

void	sid_pal_log (sid_pal_log_severity_t severity, uint32_t num_args, const char *fmt,...) Printf style logging function.
sid_pal_log_severity_t	sid_log_control_get_current_log_level (void) The function returns current logging level.
void	sid_pal_log_flush (void) Flush log function.
char const *	sid_pal_log_push_str (char *string) String log pushing operation - in the case of JLink RTT logs since these are deferred, they require that a special function be used in order to ensure that strings are copied correctly in order for pushing then at a later time.
bool	sid_pal_log_get_log_buffer (struct sid_pal_log_buffer *const log_buffer) Allows for retrieval of log buffers from sid_pal_log implementation.
void	sid_pal_hexdump (sid_pal_log_severity_t severity, const void *address, int length) Log raw data bytes.

Macros

#define	SID_PAL_HEXDUMP_MAX (8) Define the maximum number of bytes per single line of sid_pal_hexdump() logging.
#define	SID_PAL_VA_NARG (...) Macro to count the number of arguments in a variadic macro.
#define	SID_PAL_VA_NARG_ (...) Helper macro to count the number of arguments in a variadic macro.
#define	SID_PAL_VA_ARG_N (_1, _2, _3, _4, _5, _6, _7, _8, _9, _10, _11, _12, _13, _14, _15, _16, _17, _18, _19, _20, _21, _22, _23, _24, _25, _26, _27, _28, _29, _30, _31, _32, _33, _34, _35, _36, _37, _38, _39, _40, _41, _42, _43, _44, _45, _46, _47, _48, _49, _50, _51, _52, _53, _54, _55, _56, _57, _58, _59, _60, _61, _62, _63, N, ...) Helper macro to extract the number of arguments.
#define	SID_PAL_RSEQ_N () Macro to define a reverse sequence of numbers from 62 to 0.

#define	SID_PAL_LOG_HIGHEST_SEVERITY (level, fmt_, ...) Logs a message with the highest severity level.
#define	SID_PAL_LOG (level_, fmt_, ...) Logs a message if the severity level is less than or equal to the configured log level.
#define	SID_PAL_HEXDUMP (level_, data_, len_) Dumps a block of data in hexadecimal format.
#define	SID_PAL_LOG_FLUSH () Flushes the log buffer.
#define	SID_PAL_LOG_PUSH_STR (x) Pushes a string to the log.
#define	SID_PAL_LOG_ERROR (fmt_, ...) Logs an error message with the highest severity level.
#define	SID_PAL_LOG_WARNING (fmt_, ...) Logs a warning message if the severity level is greater than or equal to the warning level.
#define	SID_PAL_LOG_INFO (fmt_, ...) Logs an informational message if the severity level is greater than or equal to the info level.
#define	SID_PAL_LOG_DEBUG (fmt_, ...) Logs a debug message if the severity level is greater than or equal to the debug level.
#define	SID_PAL_LOG_TRACE () Logs a trace message with file name, line number, and function name.
#define	SID_HAL_LOG (level, fmt_, ...) Logs a message if the severity level is less than or equal to the configured log level.
#define	SID_HAL_LOG_ERROR (fmt_, ...) Logs an error message with the highest severity level.
#define	SID_HAL_LOG_WARNING (fmt_, ...) Logs a warning message if the severity level is greater than or equal to the warning level.
#define	SID_HAL_LOG_INFO (fmt_, ...) Logs an informational message if the severity level is greater than or equal to the info level.
#define	SID_HAL_LOG_DEBUG (fmt_, ...) Logs a debug message if the severity level is greater than or equal to the debug level.
#define	SID_HAL_LOG_FLUSH SID_PAL_LOG_FLUSH Flushes the log buffer.
#define	SID_HAL_LOG_PUSH_STR (x) Pushes a string to the log.
#define	SID_HAL_LOG_HEXDUMP_ERROR (data_, len_) Dumps a block of data in hexadecimal format with error severity.
#define	SID_HAL_LOG_HEXDUMP_WARNING (data_, len_) Dumps a block of data in hexadecimal format with warning severity.
#define	SID_HAL_LOG_HEXDUMP_INFO (data_, len_) Dumps a block of data in hexadecimal format with info severity.
#define	SID_HAL_LOG_HEXDUMP_DEBUG (data_, len_) Dumps a block of data in hexadecimal format with debug severity.

Function Documentation

sid_pal_log

```
void sid_pal_log (sid_pal_log_severity_t severity, uint32_t num_args, const char * fmt, ... )
```

Printf style logging function.

Parameters

Type	Direction	Argument Name	Description
sid_pal_log_severity_t	[in]	severity	Severity of the log
uint32_t	[in]	num_args	Number of arguments to be logged
const char *	[in]	fmt	Format string to print with variables
...	N/A		

sid_log_control_get_current_log_level

```
sid_pal_log_severity_t sid_log_control_get_current_log_level (void )
```

The function returns current logging level.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Implemented in sid_log_control

sid_pal_log_flush

```
void sid_pal_log_flush (void )
```

Flush log function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The function flushes the log buffers to the output interface

sid_pal_log_push_str

```
char const * sid_pal_log_push_str (char * string)
```

String log pushing operation - in the case of JLink RTT logs since these are deferred, they require that a special function be used in order to ensure that strings are copied correctly in order for pushing then at a later time.

Parameters

Type	Direction	Argument Name	Description
Type	Direction	Argument Name	Description
char *	[in]	string	Pointer to the string that has to be copied.

For platforms that do not use deferred logging, this can remain unimplemented.

sid_pal_log_get_log_buffer

```
bool sid_pal_log_get_log_buffer (struct sid_pal_log_buffer *const log_buffer)
```

Allows for retrieval of log buffers from sid_pal_log implementation.

Parameters

Type	Direction	Argument Name	Description
struct sid_pal_log_buffer *const	[in]	log_buffer	Pointer to log buffer descriptor

OPTIONAL If business logic wants to send logs over an external connection they can use this api to pull out logs from sid_pal_log implementation if implemented. Internally the implementation can use a circular list of buffers to store the logs as they come.

sid_pal_hexdump

```
void sid_pal_hexdump (sid_pal_log_severity_t severity, const void * address, int length)
```

Log raw data bytes.

Parameters

Type	Direction	Argument Name	Description
sid_pal_log_severity_t	[in]	severity	Severity of the log
const void *	[in]	address	Pointer to data to be logged
int	[in]	length	The length of data to be logged (in bytes)

sid_pal_log_buffer

Describes the log buffer that is retrieved from sid_pal_log_ifc implementation.

Public Attributes

uint8_t *	buf	Raw buffer that the sid_pal_log implementation will copy the log string into.
uint8_t	size	Size of the above Raw buffer, sid_pal_log_get_log_buffer will replace the size value with the actual size of the log string that was copied in bytes.
uint8_t	idx	Index of log string.

Public Attribute Documentation

buf

```
uint8_t* sid_pal_log_buffer::buf
```

Raw buffer that the sid_pal_log implementation will copy the log string into.

size

```
uint8_t sid_pal_log_buffer::size
```

Size of the above Raw buffer, [sid_pal_log_get_log_buffer](#) will replace the size value with the actual size of the log string that was copied in bytes.

idx

```
uint8_t sid_pal_log_buffer::idx
```

Index of log string.

Type definitions

Type definitions

Enumerations

```
enum sid_pal_log_severity_t {
    SID_PAL_LOG_SEVERITY_ERROR = 0
    SID_PAL_LOG_SEVERITY_WARNING = 1
    SID_PAL_LOG_SEVERITY_INFO = 2
    SID_PAL_LOG_SEVERITY_DEBUG = 3
}
Log severity levels.
```

Enumeration Documentation

sid_pal_log_severity_t

sid_pal_log_severity_t

Log severity levels.

This enum defines the severity levels for logging messages.

Enumerator	
SID_PAL_LOG_SEVERITY_ERROR	Error severity level
SID_PAL_LOG_SEVERITY_WARNING	Warning severity level
SID_PAL_LOG_SEVERITY_INFO	Info severity level
SID_PAL_LOG_SEVERITY_DEBUG	Debug severity level

Peripheral Interfaces

Peripheral Interfaces

Peripheral Interfaces

The Peripheral Interfaces module provides interfaces for interacting with various hardware peripherals. These interfaces ensure consistent and platform-independent access to peripheral functionalities such as GPIO, temperature sensors, and timers. Developers can create applications that are portable across different hardware platforms while maintaining consistent behavior and performance.

Modules

[GPIO](#)

[Serial Bus Interface](#)

[Serial Client Interface](#)

[Temperature](#)

GPIO

GPIO

The GPIO Interface module provides interfaces for interacting with General-Purpose Input/Output (GPIO) pins within the Sidewalk SDK. These interfaces ensure consistent and platform-independent access to GPIO functionalities, allowing developers to control and monitor digital signals across different hardware platforms.

Modules

[Type definitions](#)

Typedefs

typedef void(*) [sid_pal_gpio_irq_handler_t](#)(uint32_t gpio_number, void *callback_arg)
GPIO Pin IRQ handler.

Functions

sid_error_t	sid_pal_gpio_set_direction (uint32_t gpio_number, sid_pal_gpio_direction_t direction) sid_pal_gpio_set_direction is used to set the direction of the GPIO.
sid_error_t	sid_pal_gpio_read (uint32_t gpio_number, uint8_t *value) sid_pal_gpio_read is used to read data from GPIO pin.
sid_error_t	sid_pal_gpio_write (uint32_t gpio_number, uint8_t value) sid_pal_gpio_write is used to write data to the GPIO.
sid_error_t	sid_pal_gpio_toggle (uint32_t gpio_number) sid_pal_gpio_toggle is used to toggle the GPIO.
sid_error_t	sid_pal_gpio_set_irq (uint32_t gpio_number, sid_pal_gpio_irq_trigger_t irq_trigger, sid_pal_gpio_irq_handler_t gpio_irq_handler, void *callback_arg) sid_pal_gpio_set_irq is used to generate an interrupt based on the configuration and set callback function.
sid_error_t	sid_pal_gpio_irq_enable (uint32_t gpio_number) sid_pal_gpio_irq_enable is used to enable an interrupt.
sid_error_t	sid_pal_gpio_irq_disable (uint32_t gpio_number) sid_pal_gpio_irq_disable is used to disable an interrupt.
sid_error_t	sid_pal_gpio_input_mode (uint32_t gpio_number, sid_pal_gpio_input_t mode) sid_pal_gpio_input_mode is used to configure input mode of GPIO.
sid_error_t	sid_pal_gpio_output_mode (uint32_t gpio_number, sid_pal_gpio_output_t mode) sid_pal_gpio_output_mode is used to configure output mode of GPIO.
sid_error_t	sid_pal_gpio_pull_mode (uint32_t gpio_number, sid_pal_gpio_pull_t pull) sid_pal_gpio_pull_mode is used to configure pull type of GPIO.

Typedef Documentation

sid_pal_gpio_irq_handler_t


```
typedef void(* sid_pal_gpio_irq_handler_t) (uint32_t gpio_number, void *callback_arg) (uint32_t gpio_number, void *callback_arg)
```

GPIO Pin IRQ handler.

Parameters

Type	Direction	Argument Name	Description
	[in]	gpio_number	The logical GPIO number.
	[in]	callback_arg	The argument to be passed to the callback function.

Note

- The callback to be called when the configured transition occurs.

Function Documentation

sid_pal_gpio_set_direction

```
sid_error_t sid_pal_gpio_set_direction (uint32_t gpio_number, sid\_pal\_gpio\_direction\_t direction)
```

sid_pal_gpio_set_direction is used to set the direction of the GPIO.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number.
sid_pal_gpio_direction_t	[in]	direction	Direction of GPIO.

sid_pal_gpio_read

```
sid_error_t sid_pal_gpio_read (uint32_t gpio_number, uint8_t * value)
```

sid_pal_gpio_read is used to read data from GPIO pin.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number.
uint8_t *	[out]	value	Value read from the GPIO.

sid_pal_gpio_write

```
sid_error_t sid_pal_gpio_write (uint32_t gpio_number, uint8_t value)
```

sid_pal_gpio_write is used to write data to the GPIO.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number.
uint8_t	[out]	value	Value to write to GPIO.

sid_pal_gpio_toggle

```
sid_error_t sid_pal_gpio_toggle (uint32_t gpio_number)
```

sid_pal_gpio_toggle is used to toggle the GPIO.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number.

sid_pal_gpio_set_irq

```
sid_error_t sid_pal_gpio_set_irq (uint32_t gpio_number, sid\_pal\_gpio\_irq\_trigger\_t irq_trigger, sid\_pal\_gpio\_irq\_handler\_t gpio_irq_handler, void * callback_arg)
```

sid_pal_gpio_set_irq is used to generate an interrupt based on the configuration and set callback function.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number.
sid_pal_gpio_irq_trigger_t	[in]	irq_trigger	The interrupt config types to generate an interrupt based on the configuration.
sid_pal_gpio_irq_handler_t	[in]	gpio_irq_handler	The callback function to be called on interrupt.
void *	[in]	callback_arg	The argument to be passed to the callback function.

sid_pal_gpio_irq_enable

```
sid_error_t sid_pal_gpio_irq_enable (uint32_t gpio_number)
```

sid_pal_gpio_irq_enable is used to disable an interrupt.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number to enable interrupts.

sid_pal_gpio_irq_disable

```
sid_error_t sid_pal_gpio_irq_disable (uint32_t gpio_number)
```

sid_pal_gpio_irq_disable is used to disable an interrupt.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number to disable interrupts.

sid_pal_gpio_input_mode

```
sid_error_t sid_pal_gpio_input_mode (uint32_t gpio_number, sid_pal_gpio_input_t mode)
```

sid_pal_gpio_input_mode is used to configure input mode of GPIO.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number to operate on.
sid_pal_gpio_input_t	[in]	mode	The input mode to set.

sid_pal_gpio_output_mode

```
sid_error_t sid_pal_gpio_output_mode (uint32_t gpio_number, sid_pal_gpio_output_t mode)
```

sid_pal_gpio_output_mode is used to configure output mode of GPIO.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number to operate on.
sid_pal_gpio_output_t	[in]	mode	The output mode to set.

sid_pal_gpio_pull_mode

```
sid_error_t sid_pal_gpio_pull_mode (uint32_t gpio_number, sid_pal_gpio_pull_t pull)
```

sid_pal_gpio_pull_mode is used to configure pull type of GPIO.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	gpio_number	The logical GPIO number to operate on.
sid_pal_gpio_pull_t	[in]	pull	The pull mode to set.

Type definitions

Type definitions

Enumerations

```
enum sid_pal_gpio_pull_t {
    SID_PAL_GPIO_PULL_NONE
    SID_PAL_GPIO_PULL_UP
    SID_PAL_GPIO_PULL_DOWN
}
enum for configuring GPIO pull

enum sid_pal_gpio_output_t {
    SID_PAL_GPIO_OUTPUT_PUSH_PULL
    SID_PAL_GPIO_OUTPUT_OPEN_DRAIN
}
enum for configuring GPIO output type

enum sid_pal_gpio_input_t {
    SID_PAL_GPIO_INPUT_CONNECT
    SID_PAL_GPIO_INPUT_DISCONNECT
}
enum for configuring GPIO input buffer connect state

enum sid_pal_gpio_direction_t {
    SID_PAL_GPIO_DIRECTION_INPUT
    SID_PAL_GPIO_DIRECTION_OUTPUT
}
enum for configuring direction of GPIO.

enum sid_pal_gpio_irq_trigger_t {
    SID_PAL_GPIO_IRQ_TRIGGER_NONE
    SID_PAL_GPIO_IRQ_TRIGGER_RISING
    SID_PAL_GPIO_IRQ_TRIGGER_FALLING
    SID_PAL_GPIO_IRQ_TRIGGER_EDGE
    SID_PAL_GPIO_IRQ_TRIGGER_LOW
    SID_PAL_GPIO_IRQ_TRIGGER_HIGH
}
GPIO pin interrupt config types.
```

Enumeration Documentation

sid_pal_gpio_pull_t

sid_pal_gpio_pull_t

enum for configuring GPIO pull

Enumerator	
SID_PAL_GPIO_PULL_NONE	Configure GPIO as no pull
SID_PAL_GPIO_PULL_UP	Configure GPIO as pull up

SID_PAL_GPIO_PULL_DOWN	Configure GPIO as pull down
------------------------	-----------------------------

sid_pal_gpio_output_t

sid_pal_gpio_output_t

enum for configuring GPIO output type

Enumerator	
SID_PAL_GPIO_OUTPUT_PUSH_PULL	Configure GPIO as push pull
SID_PAL_GPIO_OUTPUT_OPEN_DRAIN	Configure GPIO as open drain output

sid_pal_gpio_input_t

sid_pal_gpio_input_t

enum for configuring GPIO input buffer connect state

Enumerator	
SID_PAL_GPIO_INPUT_CONNECT	Configure GPIO as input connected
SID_PAL_GPIO_INPUT_DISCONNECT	Configure GPIO as input disconnected

sid_pal_gpio_direction_t

sid_pal_gpio_direction_t

enum for configuring direction of GPIO.

Enumerator	
SID_PAL_GPIO_DIRECTION_INPUT	Configure GPIO as input
SID_PAL_GPIO_DIRECTION_OUTPUT	Configure GPIO as output

sid_pal_gpio_irq_trigger_t

sid_pal_gpio_irq_trigger_t

GPIO pin interrupt config types.

Enumerator	
SID_PAL_GPIO_IRQ_TRIGGER_NONE	Disable interrupt
SID_PAL_GPIO_IRQ_TRIGGER_RISING	Trigger interrupt on rising edge
SID_PAL_GPIO_IRQ_TRIGGER_FALLING	Trigger interrupt on falling edge
SID_PAL_GPIO_IRQ_TRIGGER_EDGE	Trigger interrupt on both edges
SID_PAL_GPIO_IRQ_TRIGGER_LOW	Low level triggered interrupt
SID_PAL_GPIO_IRQ_TRIGGER_HIGH	High level triggered interrupt

Serial Bus Interface

Serial Bus Interface

The Serial Bus Interface module provides interfaces for interacting with serial bus communication protocols within the Sidewalk SDK. These interfaces ensure consistent and platform-independent access to serial bus functionalities, allowing developers to implement communication between different components and peripherals across various hardware platforms.

Modules

[Type definitions](#)

Type definitions

Type definitions

Modules

[sid_pal_serial_bus_client](#)

[sid_pal_serial_bus_iface](#)

[sid_pal_serial_bus_factory](#)

Enumerations

```
enum  sid\_pal\_serial\_bus\_bit\_order {  
    SID_PAL_SERIAL_BUS_BIT_ORDER_MSB_FIRST  
    SID_PAL_SERIAL_BUS_BIT_ORDER_LSB_FIRST  
}  
Describes the bit order of messages exchanged on serial bus interface.  
  
enum  sid\_pal\_serial\_bus\_client\_select {  
    SID_PAL_SERIAL_BUS_CLIENT_DESELECT  
    SID_PAL_SERIAL_BUS_CLIENT_SELECT  
}  
Describes the client selection or deselection.
```

Enumeration Documentation

sid_pal_serial_bus_bit_order

sid_pal_serial_bus_bit_order

Describes the bit order of messages exchanged on serial bus interface.

Enumerator	
SID_PAL_SERIAL_BUS_BIT_ORDER_MSB_FIRST	Most significant bit first.
SID_PAL_SERIAL_BUS_BIT_ORDER_LSB_FIRST	Least significant bit first.

sid_pal_serial_bus_client_select

sid_pal_serial_bus_client_select

Describes the client selection or deselection.

Enumerator	
SID_PAL_SERIAL_BUS_CLIENT_DESELECT	Deselect the serial bus client
SID_PAL_SERIAL_BUS_CLIENT_SELECT	Select the serial bus client

sid_pal_serial_bus_client

Describes the configuration of the serial bus client.

Public Attributes

uint32_t	client_selector
uint32_t	speed_hz
enum sid_pal_serial_bus_bit_order	bit_order
uint8_t	mode
const void *	client_selector_extension
bool(*)	client_selector_cb Callback to select client on serial bus.
void *	client_selector_context

Public Attribute Documentation

client_selector

uint32_t sid_pal_serial_bus_client::client_selector

client id on the serial bus.

speed_hz

uint32_t sid_pal_serial_bus_client::speed_hz

baud rate.

bit_order

enum [sid_pal_serial_bus_bit_order](#) sid_pal_serial_bus_client::bit_order

bit order.

mode

uint8_t sid_pal_serial_bus_client::mode

serial bus mode.

client_selector_extension

```
const void* sid_pal_serial_bus_client::client_selector_extension
```

pointer to the client selector data.

client_selector_cb

```
bool(* sid_pal_serial_bus_client::client_selector_cb) (const struct sid_pal_serial_bus_client *const client, enum  
sid_pal_serial_bus_client_select select, void *context)
```

Callback to select client on serial bus.

If this callback is not set, then [client_selector](#) will be used.

client_selector_context

```
void* sid_pal_serial_bus_client::client_selector_context
```

Context returned back in [client_selector_cb](#)

sid_pal_serial_bus_iface

The set of callbacks the implementation supports.

Public Attributes

sid_error_t(*)	xfer	Callback to transfer messages in full duplex mode.
sid_error_t(*)	xfer_hd	Callback to transfer messages in half duplex mode.
sid_error_t(*)	destroy	Callback to delete the serial bus interface.

Public Attribute Documentation

xfer

```
sid_error_t(* sid_pal_serial_bus_iface::xfer) (const struct sid_pal_serial_bus_iface *iface, const struct sid_pal_serial_bus_client *client, uint8_t *tx, uint8_t *rx, size_t xfer_size)
```

Callback to transfer messages in full duplex mode.

Returns

- sid_error_t error code indicating the result of the operation.

xfer_hd

```
sid_error_t(* sid_pal_serial_bus_iface::xfer_hd) (const struct sid_pal_serial_bus_iface *iface, const struct sid_pal_serial_bus_client *client, uint8_t *tx, uint8_t *rx, size_t tx_size, size_t rx_size)
```

Callback to transfer messages in half duplex mode.

Returns

- sid_error_t error code indicating the result of the operation.

destroy

```
sid_error_t(* sid_pal_serial_bus_iface::destroy) (const struct sid_pal_serial_bus_iface *iface)
```

Callback to delete the serial bus interface.

Returns

- sid_error_t error code indicating the result of the operation.

sid_pal_serial_bus_factory

Factory for creating serial bus client interfaces.

Public Attributes

`sid_error_t(*` [create](#)
Callback to create serial bus client interface.

`const void *` [config](#)

Public Attribute Documentation

create

```
sid_error_t(* sid_pal_serial_bus_factory::create) (const struct sid_pal_serial_bus_iface **iface, const void *config)
```

Callback to create serial bus client interface.

Returns

- `sid_error_t` error code indicating the result of the operation.

config

```
const void* sid_pal_serial_bus_factory::config
```

pointer to client config

Serial Client Interface

Serial Client Interface

The Serial Client Interface module provides interfaces for implementing serial communication clients within the Sidewalk SDK. These interfaces ensure consistent and platform-independent access to serial communication functionalities, allowing developers to create applications that can communicate with various serial devices and peripherals across different hardware platforms.

Modules

[Type definitions](#)

Type definitions

Type definitions

Modules

- [sid_pal_serial_callbacks_t](#)
- [sid_pal_serial_params_t](#)
- [sid_pal_serial_ifc_s](#)
- [sid_pal_serial_client_factory_t](#)

Typedefs

```
typedef const struct sid_pal_serial_ifc_t
sid_pal_serial_ifc_s *
Type definition for a constant pointer to a structure representing the serial client interface.
```

Typedef Documentation

sid_pal_serial_ifc_t

sid_pal_serial_ifc_t

Type definition for a constant pointer to a structure representing the serial client interface.

sid_pal_serial_callbacks_t

Defines types of callbacks.

The one of runtime parameters' elements has such type. It is used to get externally defined callbacks which would be used in interface's methods.

Public Attributes

sid_error_t(*	tx_done_cb	Declaration of function type.
sid_error_t(*	rx_done_cb	Declaration of pointer to function.
sid_error_t(*	new_rx_done_cb	Notifies that some data has been received and can be fetched by method sid_pal_serial_ifc_s::get_frame .

Public Attribute Documentation

tx_done_cb

```
sid_error_t(* sid_pal_serial_callbacks_t::tx_done_cb) (void *user_ctx)
```

Declaration of function type.

The function itself is callback which can be called by interface's methods. This callback should be defined outside a module which defines implementation for this interface. Most likely it would be defined in the module which uses implementation of the interface. Callback is called when transmission by means of implementation is finished.

rx_done_cb

```
sid_error_t(* sid_pal_serial_callbacks_t::rx_done_cb) (void *user_ctx, const uint8_t *buffer_received, size_t buffer_size)
```

Declaration of pointer to function.

The function itself is callback which can be called by interface's methods. This callback should be defined outside a module which defines implementation for this interface. Most likely it would be defined in the module which uses implementation of the interface. Callback is called when implementation has finished reception of data by its means.

new_rx_done_cb

```
sid_error_t(* sid_pal_serial_callbacks_t::new_rx_done_cb) (void *user_ctx)
```

Notifies that some data has been received and can be fetched by method [sid_pal_serial_ifc_s::get_frame](#).

sid_pal_serial_params_t

This structure type defines all configurations/values for specific "instance" which could be initialized/derived in runtime.

Note

- It will be extended when more functionality is added to implementation.

Public Attributes

```
void *    params_ctx

void *    user_ctx

const    callbacks
sid_pal_serial_call
backs_t *
```

```
const struct sid_event_queue
queue
*
```

Public Attribute Documentation

params_ctx

```
void* sid_pal_serial_params_t::params_ctx
```

Runtime parameters for specific implementation. Its type defined in implementation so create function can cast it to that type.

user_ctx

```
void* sid_pal_serial_params_t::user_ctx
```

External parameters to pass in callbacks. The implementation doesn't need to know the type and content of it, since callbacks defined outside it and most likely in the same module as context.

callbacks

```
const sid_pal_serial_callbacks_t* sid_pal_serial_params_t::callbacks
```

Pointer to constant structure with defined callbacks. Callbacks will be used in methods.

queue

```
const struct sid_event_queue* sid_pal_serial_params_t::queue
```

Pointer to the event queue structure.

sid_pal_serial_ifc_s

Declaration of pointers to serial client interface's methods.

Public Attributes

sid_error_t(*)	send	Pointer to implementation-depended method which send data in "buffer_to_send" with size "buffer_size" over serial connection to host.
sid_error_t(*)	get_frame	Fetches frame if one was received.
sid_error_t(*)	process	Pointer to implementation-depended method which executes routine to sustain connection with the host.
sid_error_t(*)	get_mtu	Pointer to implementation-depended method.
void(*)	destroy	Pointer to implementation-depended method which free memory taken by specific instance.

Public Attribute Documentation

send

```
sid_error_t(* sid_pal_serial_ifc_s::send) (const sid_pal_serial_ifc_t *_this, const uint8_t *frame_to_send, size_t frame_size)
```

Pointer to implementation-depended method which send data in "buffer_to_send" with size "buffer_size" over serial connection to host.

Returns

- SID_ERROR_NONE - in case method finished with success. In case of error the error's type depend upon implementation.

get_frame

```
sid_error_t(* sid_pal_serial_ifc_s::get_frame) (const sid_pal_serial_ifc_t *_this, uint8_t **frame_received, size_t *frame_size)
```

Fetches frame if one was received.

It is advisable to copy it instantaneously, since there is no guaranty that frame won't be corrupted or erased after call of this function finished.

Returns

- Any other error possible if function not succeeded to execute properly.

process

```
sid_error_t(* sid_pal_serial_ifc_s::process) (const sid_pal_serial_ifc_t *_this)
```

Pointer to implementation-depended method which executes routine to sustain connection with the host.

get_mtu

```
sid_error_t(* sid_pal_serial_ifc_s::get_mtu) (const sid_pal_serial_ifc_t *_this, uint16_t *mtu)
```

Pointer to implementation-depended method.

It retrieves mtu (maximum transmit unit).

destroy

```
void(* sid_pal_serial_ifc_s::destroy) (const sid_pal_serial_ifc_t *_this)
```

Pointer to implementation-depended method which free memory taken by specific instance.

sid_pal_serial_client_factory_t

Generic type for instance factory.

Public Attributes

- sid_error_t(*

[sid_pal_serial_client_create](#)

General type for constructors of this interface.
- const void *

[config](#)

Public Attribute Documentation

sid_pal_serial_client_create

```
sid_error_t(* sid_pal_serial_client_factory_t::sid_pal_serial_client_create) (sid_pal_serial_ifc_t const **_this, const void *config, sid_pal_serial_params_t const *params)
```

General type for constructors of this interface.

It's advised to use it.

config

```
const void* sid_pal_serial_client_factory_t::config
```

The configuration to be passed in sid_pal_serial_create.

Temperature

Temperature

The Temperature Interface module provides interfaces for interacting with temperature sensors within the Sidewalk SDK. These interfaces ensure consistent and platform-independent access to temperature measurement functionalities, allowing developers to create applications that can accurately monitor and respond to temperature changes across different hardware platforms.

Functions

- sid_error_t

sid_pal_temperature_init(void)

Init temperature detection.
- int16_t

sid_pal_temperature_get(void)

Get temperature.

Function Documentation

sid_pal_temperature_init

sid_error_t sid_pal_temperature_init (void)

Init temperature detection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_temperature_get

int16_t sid_pal_temperature_get (void)

Get temperature.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Radio Interfaces

Radio Interfaces

Radio Interfaces

The Radio Interfaces module provides interfaces for interacting with radio hardware within the Sidewalk SDK. These interfaces ensure consistent and platform-independent access to radio functionalities, such as transmission, reception, and configuration of radio parameters. Developers can create applications that are portable across different hardware platforms while maintaining consistent radio behavior and performance.

The Radio Interfaces module includes support for various radio operations, including:

- **Transmission and Reception:** Interfaces for sending and receiving data over the radio.
- **Configuration:** Interfaces for configuring radio parameters such as data rate, channel, and power settings.
- **Event Handling:** Mechanisms for handling radio events such as transmission completion, reception completion, and errors.
- **State Management:** Interfaces for managing the state of the radio, including transitions between different states such as sleep, standby, and active modes.

By utilizing the Radio Interfaces module, developers can ensure that their applications can effectively communicate over the radio within the Amazon Sidewalk network, regardless of the underlying hardware platform.

Modules

[FSK Interface](#)

[LoRa Interface](#)

[Sub-GHz Interface](#)

FSK Interface

FSK Interface

FSK modulation defines for Sidewalk.

Modules

[Type definitions](#)

Type definitions

Type definitions

Modules

[sid_pal_radio_fsk_cad_params_t](#)

[sid_pal_radio_fsk_modulation_params_t](#)

[sid_pal_radio_fsk_packet_params_t](#)

[sid_pal_radio_fsk_phy_hdr_t](#)

[sid_pal_radio_fsk_pkt_cfg_t](#)

[sid_pal_radio_fsk_rx_packet_status_t](#)

[sid_pal_radio_fsk_phy_settings_t](#)

Enumerations

```
enum    sid\_pal\_radio\_fsk\_header\_type {  
        SID_PAL_RADIO_FSK_SIDEWALK_HEADER = 0  
        SID_PAL_RADIO_FSK_CUSTOM_HEADER = 1  
    }  
    Sidewalk Phy FSK header type.
```

```
enum    radio\_fsk\_fcs\_t {  
        RADIO_FSK_FCS_TYPE_0 = 0  
        RADIO_FSK_FCS_TYPE_1 = 1  
    }  
    Radio FSK FCS enumeration definition.
```

Macros

```
#define    SID_PAL_RADIO_FSK_MOD_SHAPING_OFF 0x00  
    Radio Mod Shaping parameter.
```

```
#define    SID_PAL_RADIO_FSK_MOD_SHAPING_G_BT_03 0x08
```

```
#define    SID_PAL_RADIO_FSK_MOD_SHAPING_G_BT_05 0x09
```

```
#define    SID_PAL_RADIO_FSK_MOD_SHAPING_G_BT_07 0x0A
```

```
#define    SID_PAL_RADIO_FSK_MOD_SHAPING_G_BT_1 0x0B
```

```
#define    SID_PAL_RADIO_FSK_BW_4800 0x1F  
    Bandwidth.
```

```
#define    SID_PAL_RADIO_FSK_BW_5800 0x07
```

```
#define    SID_PAL_RADIO_FSK_BW_7300 0x0F
```

```
#define    SID_PAL_RADIO_FSK_BW_9700 0x1E
```

```
#define SID_PAL_RADIO_FSK_BW_11700 0x16

#define SID_PAL_RADIO_FSK_BW_14600 0x0E

#define SID_PAL_RADIO_FSK_BW_19500 0x1D

#define SID_PAL_RADIO_FSK_BW_23400 0x15

#define SID_PAL_RADIO_FSK_BW_29300 0x0D

#define SID_PAL_RADIO_FSK_BW_39000 0x1C

#define SID_PAL_RADIO_FSK_BW_46900 0x14

#define SID_PAL_RADIO_FSK_BW_58600 0x0C

#define SID_PAL_RADIO_FSK_BW_78200 0x1B

#define SID_PAL_RADIO_FSK_BW_93800 0x13

#define SID_PAL_RADIO_FSK_BW_117300 0x0B

#define SID_PAL_RADIO_FSK_BW_156200 0x1A

#define SID_PAL_RADIO_FSK_BW_187200 0x12

#define SID_PAL_RADIO_FSK_BW_234300 0x0A

#define SID_PAL_RADIO_FSK_BW_312000 0x19

#define SID_PAL_RADIO_FSK_BW_373600 0x11

#define SID_PAL_RADIO_FSK_BW_467000 0x09

#define SID_PAL_RADIO_FSK_BW_100KHZ SID_PAL_RADIO_FSK_BW_93800

#define SID_PAL_RADIO_FSK_BW_117KHZ SID_PAL_RADIO_FSK_BW_117300

#define SID_PAL_RADIO_FSK_BW_125KHZ SID_PAL_RADIO_FSK_BW_156200

#define SID_PAL_RADIO_FSK_BW_150KHZ SID_PAL_RADIO_FSK_BW_156200

#define SID_PAL_RADIO_FSK_BW_250KHZ SID_PAL_RADIO_FSK_BW_312000

#define SID_PAL_RADIO_FSK_BW_500KHZ SID_PAL_RADIO_FSK_BW_467000

#define SID_PAL_RADIO_FSK_PREAMBLE_DETECTOR_OFF 0x00
Radio Preamble detection.

#define SID_PAL_RADIO_FSK_PREAMBLE_DETECTOR_08_BITS 0x04

#define SID_PAL_RADIO_FSK_PREAMBLE_DETECTOR_16_BITS 0x05

#define SID_PAL_RADIO_FSK_PREAMBLE_DETECTOR_24_BITS 0x06

#define SID_PAL_RADIO_FSK_PREAMBLE_DETECTOR_32_BITS 0x07

#define SID_PAL_RADIO_FSK_ADDRESSCOMP_FILT_OFF 0x00
Radio sync word correlators activated.

#define SID_PAL_RADIO_FSK_ADDRESSCOMP_FILT_NODE 0x01
```

```
#define SID_PAL_RADIO_FSK_ADDRESSCOMP_FILT_NODE_BROAD 0x02

#define SID_PAL_RADIO_FSK_RADIO_PACKET_FIXED_LENGTH 0x00
Radio packet length modes.

#define SID_PAL_RADIO_FSK_RADIO_PACKET_VARIABLE_LENGTH 0x01

#define SID_PAL_RADIO_FSK_CRC_OFF 0x01
packet params crc types

#define SID_PAL_RADIO_FSK_CRC_1_BYTES 0x00

#define SID_PAL_RADIO_FSK_CRC_2_BYTES 0x02

#define SID_PAL_RADIO_FSK_CRC_1_BYTES_INV 0x04

#define SID_PAL_RADIO_FSK_CRC_2_BYTES_INV 0x06

#define SID_PAL_RADIO_FSK_CRC_2_BYTES_IBM 0xF1

#define SID_PAL_RADIO_FSK_CRC_2_BYTES_CCIT 0xF2

#define SID_PAL_RADIO_FSK_DC_FREE_OFF 0x00
packet params Radio whitening mode

#define SID_PAL_RADIO_FSK_DC_FREEWHTENING 0x01

#define SID_PAL_RADIO_FSK_WHITENING_SEED 0x01FF

#define SID_PAL_RADIO_FSK_SYNC_WORD_LENGTH 8

#define SECS_TO_MUS (X)
timeout duration in usec.

#define SID_PAL_RADIO_FSK_DEFAULT_TX_TIMEOUT SECS_TO_MUS(5)
set max radio timeout to 5sec

#define SID_PAL_RADIO_FSK_TIMEOUT_DURATION_1_SEC SECS_TO_MUS(1)
1 sec timeout used by diagnostics code

#define SID_MAX_CUSTOM_PHYHDR_SZ 4
```

Enumeration Documentation

sid_pal_radio_fsk_header_type

sid_pal_radio_fsk_header_type

Sidewalk Phy FSK header type.

Enumerator	
SID_PAL_RADIO_FSK_SIDEWALK_HEADER	Sidewalk-specific header
SID_PAL_RADIO_FSK_CUSTOM_HEADER	Custom header

radio_fsk_fcs_t

radio_fsk_fcs_t

Radio FSK FCS enumeration definition.

Enumerator	
RADIO_FSK_FCS_TYPE_0	4-octet FCS
RADIO_FSK_FCS_TYPE_1	2-octet FCS

sid_pal_radio_fsk_cad_params_t

Sidewalk Phy FSK CAD (Channel Activity Detection) parameters.

Public Attributes

int16_t	fsk_ed_rssi_threshold
uint16_t	fsk_ed_duration_mus
uint8_t	fsk_cs_min_prm_det
uint32_t	fsk_cs_duration_us
uint32_t	fsk_cs_lbt_rx_timeout
uint16_t	fsk_cs_lbt_preamble_len

Public Attribute Documentation

fsk_ed_rssi_threshold

int16_t sid_pal_radio_fsk_cad_params_t::fsk_ed_rssi_threshold

RSSI threshold for energy detection

fsk_ed_duration_mus

uint16_t sid_pal_radio_fsk_cad_params_t::fsk_ed_duration_mus

Duration for energy detection in microseconds

fsk_cs_min_prm_det

uint8_t sid_pal_radio_fsk_cad_params_t::fsk_cs_min_prm_det

Minimum preamble detection for carrier sense

fsk_cs_duration_us

uint32_t sid_pal_radio_fsk_cad_params_t::fsk_cs_duration_us

Duration for carrier sense in microseconds

fsk_cs_lbt_rx_timeout

```
uint32_t sid_pal_radio_fsk_cad_params_t::fsk_cs_lbt_rx_timeout
```

RX timeout for Listen Before Talk (LBT)

fsk_cs_lbt_preamble_len

```
uint16_t sid_pal_radio_fsk_cad_params_t::fsk_cs_lbt_preamble_len
```

Preamble length for Listen Before Talk (LBT)

sid_pal_radio_fsk_modulation_params_t

Sidewalk phy fsk modulation parameters.

Public Attributes

uint32_t	bit_rate
uint32_t	freq_dev
enum sid_pal_radio_fsk_header_type	header_type
uint8_t	mod_shaping
uint8_t	bandwidth
uint8_t	custom_rate_idx

Public Attribute Documentation

bit_rate

uint32_t sid_pal_radio_fsk_modulation_params_t::bit_rate

Bit rate for the FSK modulation

freq_dev

uint32_t sid_pal_radio_fsk_modulation_params_t::freq_dev

Frequency deviation for the FSK modulation

header_type

enum sid_pal_radio_fsk_header_type sid_pal_radio_fsk_modulation_params_t::header_type

FSK header type

mod_shaping

uint8_t sid_pal_radio_fsk_modulation_params_t::mod_shaping

Modulation shaping parameter

bandwidth


```
uint8_t sid_pal_radio_fsk_modulation_params_t::bandwidth
```

Bandwidth for the FSK modulation

custom_rate_idx

```
uint8_t sid_pal_radio_fsk_modulation_params_t::custom_rate_idx
```

Rate index if the data rate is custom

sid_pal_radio_fsk_packet_params_t

Sidewalk phy fsk packet parameters.

Public Attributes

uint16_t	preamble_length
uint8_t	preamble_min_detect
uint8_t	sync_word_length
uint8_t	addr_comp
uint8_t	header_type
uint8_t	payload_length
uint8_t *	payload
uint8_t	crc_type
uint8_t	radio_whitening_mode

Public Attribute Documentation

preamble_length

uint16_t sid_pal_radio_fsk_packet_params_t::preamble_length

Length of the preamble

preamble_min_detect

uint8_t sid_pal_radio_fsk_packet_params_t::preamble_min_detect

Minimum preamble detection

sync_word_length

uint8_t sid_pal_radio_fsk_packet_params_t::sync_word_length

Length of the sync word

addr_comp

uint8_t sid_pal_radio_fsk_packet_params_t::addr_comp

Address comparison mode

header_type

uint8_t sid_pal_radio_fsk_packet_params_t::header_type

Type of the header

payload_length

uint8_t sid_pal_radio_fsk_packet_params_t::payload_length

Length of the payload

payload

uint8_t* sid_pal_radio_fsk_packet_params_t::payload

Pointer to the payload

crc_type

uint8_t sid_pal_radio_fsk_packet_params_t::crc_type

Type of the CRC

radio_whitening_mode

uint8_t sid_pal_radio_fsk_packet_params_t::radio_whitening_mode

Radio whitening mode

sid_pal_radio_fsk_phy_hdr_t

Radio FSK PHY HDR structure definition.

Public Attributes

radio_fsk_fcs_t	fcs_type
bool	is_data_whitening_enabled
bool	is_fec_enabled
uint8_t	phy_hdr_len
uint8_t	phy_header

Public Attribute Documentation

fcs_type

radio_fsk_fcs_t sid_pal_radio_fsk_phy_hdr_t::fcs_type

FCS (Frame Check Sequence) type

is_data_whitening_enabled

bool sid_pal_radio_fsk_phy_hdr_t::is_data_whitening_enabled

Flag to indicate if data whitening is enabled

is_fec_enabled

bool sid_pal_radio_fsk_phy_hdr_t::is_fec_enabled

Flag to indicate if FEC (Forward Error Correction) is enabled

phy_hdr_len

uint8_t sid_pal_radio_fsk_phy_hdr_t::phy_hdr_len

Length of the PHY header

phy_header

uint8_t sid_pal_radio_fsk_phy_hdr_t::phy_header[4]

PHY header

sid_pal_radio_fsk_pkt_cfg_t

Sidewalk Phy FSK packet configuration.

Public Attributes

sid_pal_radio_fsk_phy_hdr_t *	phy_hdr
sid_pal_radio_fsk_packet_params_t *	packet_params
uint32_t	packet_timeout
uint8_t *	sync_word
uint8_t *	payload

Public Attribute Documentation

phy_hdr

sid_pal_radio_fsk_phy_hdr_t* sid_pal_radio_fsk_pkt_cfg_t::phy_hdr

Pointer to the PHY header for the FSK modulation

packet_params

sid_pal_radio_fsk_packet_params_t* sid_pal_radio_fsk_pkt_cfg_t::packet_params

Pointer to the packet parameters for the FSK modulation

packet_timeout

uint32_t sid_pal_radio_fsk_pkt_cfg_t::packet_timeout

Packet timeout in microseconds

sync_word

uint8_t* sid_pal_radio_fsk_pkt_cfg_t::sync_word

Pointer to the sync word for the FSK modulation

payload

```
uint8_t* sid_pal_radio_fsk_pkt_cfg_t::payload
```

Pointer to the payload to be transmitted

sid_pal_radio_fsk_rx_packet_status_t

Sidewalk Phy received FSK packet status.

Public Attributes

- int8_t [rssi_avg](#)
- int8_t [rssi_sync](#)
- int8_t [snr](#)

Public Attribute Documentation

rssi_avg

```
int8_t sid_pal_radio_fsk_rx_packet_status_t::rssi_avg
```

Average RSSI (Received Signal Strength Indicator)

rssi_sync

```
int8_t sid_pal_radio_fsk_rx_packet_status_t::rssi_sync
```

RSSI during sync word detection

snr

```
int8_t sid_pal_radio_fsk_rx_packet_status_t::snr
```

Signal-to-Noise Ratio

sid_pal_radio_fsk_phy_settings_t

Sidewalk phy fsk configuration handle.

Public Attributes

uint32_t	freq
int8_t	power
uint8_t	sync_word
uint8_t	sync_word_len
uint16_t	whitening_seed
uint16_t	crc_polynomial
uint16_t	crc_seed
uint32_t	tx_timeout
uint32_t	symbol_timeout
sid_pal_radio_fsk_modulation_params_t	fsk_modulation_params
sid_pal_radio_fsk_packet_params_t	fsk_packet_params
sid_pal_radio_fsk_cad_params_t	fsk_cad_params
sid_pal_radio_fsk_phy_hdr_t	fsk_phy_hdr

Public Attribute Documentation

freq

uint32_t sid_pal_radio_fsk_phy_settings_t::freq

Frequency for the FSK modulation

power

int8_t sid_pal_radio_fsk_phy_settings_t::power

Transmission power in dBm

sync_word

```
uint8_t sid_pal_radio_fsk_phy_settings_t::sync_word[8]
```

Sync word for the FSK modulation

sync_word_len

```
uint8_t sid_pal_radio_fsk_phy_settings_t::sync_word_len
```

Length of the sync word

whitening_seed

```
uint16_t sid_pal_radio_fsk_phy_settings_t::whitening_seed
```

Whitening seed for the FSK modulation

crc_polynomial

```
uint16_t sid_pal_radio_fsk_phy_settings_t::crc_polynomial
```

CRC polynomial for the FSK modulation

crc_seed

```
uint16_t sid_pal_radio_fsk_phy_settings_t::crc_seed
```

CRC seed for the FSK modulation

tx_timeout

```
uint32_t sid_pal_radio_fsk_phy_settings_t::tx_timeout
```

Transmission timeout in microseconds

symbol_timeout

```
uint32_t sid_pal_radio_fsk_phy_settings_t::symbol_timeout
```

Symbol timeout in microseconds

fsk_modulation_params

```
sid_pal_radio_fsk_modulation_params_t sid_pal_radio_fsk_phy_settings_t::fsk_modulation_params
```

Modulation parameters for the FSK modulation

fsk_packet_params

```
sid_pal_radio_fsk_packet_params_t sid_pal_radio_fsk_phy_settings_t::fsk_packet_params
```

Packet parameters for the FSK modulation

fsk_cad_params

```
sid_pal_radio_fsk_cad_params_t sid_pal_radio_fsk_phy_settings_t::fsk_cad_params
```

CAD (Channel Activity Detection) parameters for the FSK modulation

fsk_phy_hdr

```
sid_pal_radio_fsk_phy_hdr_t sid_pal_radio_fsk_phy_settings_t::fsk_phy_hdr
```

PHY header for the FSK modulation

LoRa Interface

LoRa Interface

LoRa modulation defines for Sidewalk.

Modules

[Type definitions](#)

Type definitions

Type definitions

Modules

[sid_pal_radio_lora_modulation_params_t](#)

[sid_pal_radio_lora_packet_params_t](#)

[sid_pal_radio_lora_rx_packet_status_t](#)

[sid_pal_radio_lora_cad_params_t](#)

[sid_pal_radio_lora_phy_settings_t](#)

Enumerations

```
enum    sid\_pal\_radio\_lora\_crc\_present\_t {
    SID_PAL_RADIO_CRC_PRESENT_INVALID = 0
    SID_PAL_RADIO_CRC_PRESENT_OFF = 1
    SID_PAL_RADIO_CRC_PRESENT_ON = 2
    SID_PAL_RADIO_CRC_PRESENT_MAX_NUM = SID_PAL_RADIO_CRC_PRESENT_ON
}
```

Sidewalk phy lora crc present.

Macros

```
#define    SID_PAL_RADIO_LORA_SF5 0x05
    Spreading Factor.

#define    SID_PAL_RADIO_LORA_SF6 0x06

#define    SID_PAL_RADIO_LORA_SF7 0x07

#define    SID_PAL_RADIO_LORA_SF8 0x08

#define    SID_PAL_RADIO_LORA_SF9 0x09

#define    SID_PAL_RADIO_LORA_SF10 0x0A

#define    SID_PAL_RADIO_LORA_SF11 0x0B

#define    SID_PAL_RADIO_LORA_SF12 0x0C

#define    SID_PAL_RADIO_LORA_BW_7KHZ 0x00
    Bandwidth.

#define    SID_PAL_RADIO_LORA_BW_10KHZ 0x08

#define    SID_PAL_RADIO_LORA_BW_15KHZ 0x01

#define    SID_PAL_RADIO_LORA_BW_20KHZ 0x09

#define    SID_PAL_RADIO_LORA_BW_31KHZ 0x02
```

```
#define SID_PAL_RADIO_LORA_BW_41KHZ 0x0A

#define SID_PAL_RADIO_LORA_BW_62KHZ 0x03

#define SID_PAL_RADIO_LORA_BW_125KHZ 0x04

#define SID_PAL_RADIO_LORA_BW_250KHZ 0x05

#define SID_PAL_RADIO_LORA_BW_500KHZ 0x06

#define SID_PAL_RADIO_LORA_CODING_RATE_4_5 0x01
Coding Rate.

#define SID_PAL_RADIO_LORA_CODING_RATE_4_6 0x02

#define SID_PAL_RADIO_LORA_CODING_RATE_4_7 0x03

#define SID_PAL_RADIO_LORA_CODING_RATE_4_8 0x04

#define SID_PAL_RADIO_LORA_CODING_RATE_4_5_LI 0x05

#define SID_PAL_RADIO_LORA_CODING_RATE_4_6_LI 0x06

#define SID_PAL_RADIO_LORA_CODING_RATE_4_8_LI 0x07

#define SID_PAL_RADIO_LORA_HEADER_TYPE_VARIABLE_LENGTH 0x00
packet params header type

#define SID_PAL_RADIO_LORA_HEADER_TYPE_FIXED_LENGTH 0x01

#define SID_PAL_RADIO_LORA_CRC_OFF 0x00
packet params crc modes

#define SID_PAL_RADIO_LORA_CRC_ON 0x01

#define SID_PAL_RADIO_LORA_IQ_NORMAL 0x00
packet params IQ modes

#define SID_PAL_RADIO_LORA_IQ_INVERTED 0x01

#define SID_PAL_RADIO_LORA_LDR_LONG_INTERLEAVER_OFF 0x00
packet params LI modes

#define SID_PAL_RADIO_LORA_LDR_LONG_INTERLEAVER_ON 0x01

#define SID_PAL_RADIO_LORA_CAD_01_SYMBOL 0x00
cad params

#define SID_PAL_RADIO_LORA_CAD_02_SYMBOL 0x01

#define SID_PAL_RADIO_LORA_CAD_04_SYMBOL 0x02

#define SID_PAL_RADIO_LORA_CAD_08_SYMBOL 0x03

#define SID_PAL_RADIO_LORA_CAD_16_SYMBOL 0x04

#define SID_PAL_RADIO_LORA_CAD_EXIT_MODE_CAD_ONLY 0x00

#define SID_PAL_RADIO_LORA_CAD_EXIT_MODE_CAD_RX 0x01

#define SID_PAL_RADIO_LORA_CAD_EXIT_MODE_CAD_LBT 0x10
```

```
#define SID_PAL_RADIO_LORA_SF5_SF6_MIN_PREAMBLE_LEN 12

#define SECS_TO_MUS (X)
    timeout duration in usec.

#define SID_PAL_RADIO_LORA_CAD_DEFAULT_TX_TIMEOUT SECS_TO_MUS(5)
    set max radio timeout to 5sec

#define SID_PAL_RADIO_LORA_DEFAULT_TX_TIMEOUT SID_PAL_RADIO_LORA_CAD_DEFAULT_TX_TIMEOUT

#define SID_PAL_RADIO_LORA_TIMEOUT_DURATION_1_SEC SECS_TO_MUS(1)
    1 sec timeout used by diagnostics code

#define SID_PAL_RADIO_LORA_PRIVATE_NETWORK_SYNC_WORD 0x1424

#define SID_PAL_RADIO_LORA_PUBLIC_NETWORK_SYNC_WORD LORA_MAC_PUBLIC_SYNCWORD

#define SID_PAL_RADIO_LORA_MAX_PAYLOAD_LENGTH 250

#define SID_PAL_RADIO_LORA_ED_PREAMBLE_LENGTH_DEFAULT (250 << 3)

#define SID_PAL_RADIO_LORA_ED_MOD_SHAPING MOD_SHAPING_G_BT_1

#define SID_PAL_RADIO_LORA_ED_PREAMBLE_MIN_DETECT RADIO_PREAMBLE_DETECTOR_08_BITS

#define SID_PAL_RADIO_LORA_ED_SYNCWORD_LENGTH_DEFAULT (3 << 3)

#define SID_PAL_RADIO_LORA_ED_ADDRCOMP_DEFAULT RADIO_ADDRESSCOMP_FILTER_OFF

#define SID_PAL_RADIO_LORA_ED_HEADER_TYPE_DEFAULT RADIO_PACKET_VARIABLE_LENGTH

#define SID_PAL_RADIO_LORA_ED_CRC_LENGTH_DEFAULT RADIO_CRC_2_BYTES_CCIT

#define SID_PAL_RADIO_LORA_ED_RADIO_WHITENING_MODE_DEFAULT RADIO_DC_FREEWHITENING

#define SID_PAL_RADIO_LORA_ED_PAYLOAD_LENGTH_DEFAULT 0

#define SID_PAL_RADIO_LORA_ED_DEFAULT_WHITENING_SEED 0x01FF
```

Enumeration Documentation

sid_pal_radio_lora_crc_present_t

sid_pal_radio_lora_crc_present_t

Sidewalk phy lora crc present.

Enumerator	
SID_PAL_RADIO_CRC_PRESENT_INVALID	Invalid value
SID_PAL_RADIO_CRC_PRESENT_OFF	CRC not present
SID_PAL_RADIO_CRC_PRESENT_ON	CRC present
SID_PAL_RADIO_CRC_PRESENT_MAX_NUM	Maximum number of CRC present

sid_pal_radio_lora_modulation_params_t

Sidewalk phy lora modulation parameters.

Public Attributes

uint8_t	spreading_factor
uint8_t	bandwidth
uint8_t	coding_rate

Public Attribute Documentation

spreading_factor

uint8_t sid_pal_radio_lora_modulation_params_t::spreading_factor

Spreading factor

bandwidth

uint8_t sid_pal_radio_lora_modulation_params_t::bandwidth

Bandwidth

coding_rate

uint8_t sid_pal_radio_lora_modulation_params_t::coding_rate

Coding rate

sid_pal_radio_lora_packet_params_t

Sidewalk phy lora packet parameters.

Public Attributes

uint16_t	preamble_length
uint8_t	header_type
uint8_t	payload_length
uint8_t	crc_mode
uint8_t	invert_IQ

Public Attribute Documentation

preamble_length

uint16_t sid_pal_radio_lora_packet_params_t::preamble_length

Length of the preamble

header_type

uint8_t sid_pal_radio_lora_packet_params_t::header_type

Type of the header

payload_length

uint8_t sid_pal_radio_lora_packet_params_t::payload_length

Length of the payload

crc_mode

uint8_t sid_pal_radio_lora_packet_params_t::crc_mode

Type of the CRC

invert_IQ

uint8_t sid_pal_radio_lora_packet_params_t::invert_IQ

IQ inversion

sid_pal_radio_lora_rx_packet_status_t

Sidewalk Phy received LORA packet status.

Public Attributes

int16_t	rssi
int8_t	snr
int8_t	signal_rssi
sid_pal_radio_lora_crc_present_t	is_crc_present

Public Attribute Documentation

rssi

```
int16_t sid_pal_radio_lora_rx_packet_status_t::rssi
```

RSSI (Received Signal Strength Indicator)

snr

```
int8_t sid_pal_radio_lora_rx_packet_status_t::snr
```

Signal-to-Noise Ratio

signal_rssi

```
int8_t sid_pal_radio_lora_rx_packet_status_t::signal_rssi
```

Signal RSSI

is_crc_present

```
sid_pal_radio_lora_crc_present_t sid_pal_radio_lora_rx_packet_status_t::is_crc_present
```

Flag to indicate if CRC is present

sid_pal_radio_lora_cad_params_t

Sidewalk phy lora cad parameters.

Public Attributes

uint8_t	cad_symbol_num
uint8_t	cad_detect_peak
uint8_t	cad_detect_min
uint8_t	cad_exit_mode
uint32_t	cad_timeout

Public Attribute Documentation

cad_symbol_num

uint8_t sid_pal_radio_lora_cad_params_t::cad_symbol_num

Number of CAD symbols

cad_detect_peak

uint8_t sid_pal_radio_lora_cad_params_t::cad_detect_peak

CAD detection peak

cad_detect_min

uint8_t sid_pal_radio_lora_cad_params_t::cad_detect_min

CAD detection minimum

cad_exit_mode

uint8_t sid_pal_radio_lora_cad_params_t::cad_exit_mode

CAD exit mode

cad_timeout

uint32_t sid_pal_radio_lora_cad_params_t::cad_timeout

CAD timeout in microseconds

sid_pal_radio_lora_phy_settings_t

Sidewalk phy lora configuration handle.

Public Attributes

uint32_t	freq
int8_t	power
uint16_t	sync_word
uint8_t	symbol_timeout
uint32_t	tx_timeout
uint8_t	lora_ldr_long_interleaved_enable
sid_pal_radio_lora_modulation_params_t	lora_modulation_params
sid_pal_radio_lora_packet_params_t	lora_packet_params
sid_pal_radio_lora_cad_params_t	lora_cad_params

Public Attribute Documentation

freq

uint32_t sid_pal_radio_lora_phy_settings_t::freq

Frequency for the LoRa modulation

power

int8_t sid_pal_radio_lora_phy_settings_t::power

Transmission power in dBm

sync_word

uint16_t sid_pal_radio_lora_phy_settings_t::sync_word

Sync word for the LoRa modulation

symbol_timeout

```
uint8_t sid_pal_radio_lora_phy_settings_t::symbol_timeout
```

Symbol timeout value

tx_timeout

```
uint32_t sid_pal_radio_lora_phy_settings_t::tx_timeout
```

Transmission timeout in microseconds

lora_ldr_long_interleaved_enable

```
uint8_t sid_pal_radio_lora_phy_settings_t::lora_ldr_long_interleaved_enable
```

Flag to enable long interleaved mode

lora_modulation_params

```
sid_pal_radio_lora_modulation_params_t sid_pal_radio_lora_phy_settings_t::lora_modulation_params
```

Modulation parameters for the LoRa modulation

lora_packet_params

```
sid_pal_radio_lora_packet_params_t sid_pal_radio_lora_phy_settings_t::lora_packet_params
```

Packet parameters for the LoRa modulation

lora_cad_params

```
sid_pal_radio_lora_cad_params_t sid_pal_radio_lora_phy_settings_t::lora_cad_params
```

CAD (Channel Activity Detection) parameters for the LoRa modulation

Sub-GHz Interface

Sub-GHz Interface

The Sub-GHz Interface module provides interfaces for interacting with Sub-GHz radio hardware within the Sidewalk SDK.

Modules

[Type definitions](#)

Functions

`int32_t` [sid_pal_radio_init](#)(`sid_pal_radio_event_notify_t` notify, `sid_pal_radio_irq_handler_t` dio_irq_handler, `sid_pal_radio_rx_packet_t` *rx_packet)
Initializes the radio.

`int32_t` [sid_pal_radio_deinit](#)(`void`)
Deinitialize the radio.

`sid_pal_radio_irq_mask_t` [sid_pal_radio_configure_irq_mask](#)(`sid_pal_radio_irq_mask_t` irq_mask)
Configure irq mask.

`sid_pal_radio_irq_mask_t` [sid_pal_radio_get_current_config_irq_mask](#)(`void`)
Get current irq mask settings.

`int32_t` [sid_pal_radio_irq_process](#)(`void`)
Radio irq processing.

`int32_t` [sid_pal_radio_set_frequency](#)(`uint32_t` freq)
Set the frequency for the radio.

`int32_t` [sid_pal_radio_set_tx_power](#)(`int8_t` power)
Set the radio transmit power.

`int32_t` [sid_pal_radio_get_max_tx_power](#)(`sid_pal_radio_data_rate_t` data_rate, `int8_t` *tx_power)
Get the radio max transmit power setting for a given data rate.

`int32_t` [sid_pal_radio_set_region](#)(`sid_pal_radio_region_code_t` region)
Set the radio region.

`int32_t` [sid_pal_radio_sleep](#)(`uint32_t` sleep_us)
Set the radio to sleep.

`int32_t` [sid_pal_radio_standby](#)(`void`)
Set the radio to standby.

`int32_t` [sid_pal_set_radio_busy](#)(`void`)
Set the radio to busy state.

`int32_t` [sid_pal_radio_start_carrier_sense](#)(`const` `sid_pal_radio_fsk_cad_params_t` *cad_params, `sid_pal_radio_cad_param_exit_mode_t` exit_mode)
Set the radio in preamble detect mode.

`int32_t` [sid_pal_radio_start_rx](#)(`uint32_t` timeout)
Set the radio in receive mode.

int32_t	sid_pal_radio_start_continuous_rx (void) Set the radio to continuous receive.
int32_t	sid_pal_radio_set_rx_duty_cycle (uint32_t rx_time, uint32_t sleep_time) Set the receive duty cycle.
int32_t	sid_pal_radio_set_tx_continuous_wave (uint32_t freq, int8_t power) Set the transmit continuous wave.
int32_t	sid_pal_radio_set_tx_payload (const uint8_t *buffer, uint8_t size) Set transmit payload for the radio to transmit.
int32_t	sid_pal_radio_start_tx (uint32_t timeout) Start packet transmission.
uint8_t	sid_pal_radio_get_status (void) Get the radio state.
sid_pal_radio_modem_mode_t	sid_pal_radio_get_modem_mode (void) Get the current radio modem mode.
int32_t	sid_pal_radio_set_modem_mode (sid_pal_radio_modem_mode_t mode) Set radio modem mode.
int32_t	sid_pal_radio_is_channel_free (uint32_t freq, int16_t threshold, uint32_t delay_us, bool *is_channel_free) Check the channel noise level for a given rssi.
int32_t	sid_pal_radio_get_chan_noise (uint32_t freq, int16_t *noise) Compute the noise sensed by radio at a particular frequency.
int16_t	sid_pal_radio_rssi (void) Get RSSI at radio's current configured frequency.
int32_t	sid_pal_radio_random (uint32_t *random) Get a random number from radio.
int16_t	sid_pal_radio_get_ant_dbi (void) Get antenna gain in dBi.
int32_t	sid_pal_radio_get_cca_level_adjust (sid_pal_radio_data_rate_t data_rate, int8_t *adj_level) Get the cca adjustment in dB.
int32_t	sid_pal_radio_get_radio_state_transition_delays (sid_pal_radio_state_transition_timings_t *state_delay) Get the delay in microseconds to switch between different radio states.
int32_t	sid_pal_radio_is_cad_exit_mode (sid_pal_radio_cad_param_exit_mode_t mode) Check the CAD exit mode.
int32_t	sid_pal_radio_set_lora_symbol_timeout (uint8_t num_of_symbols) Radio LoRa Modulation specific APIs.
int32_t	sid_pal_radio_set_lora_sync_word (uint16_t sync_word) Set LoRa sync word.
int32_t	sid_pal_radio_set_lora_modulation_params (const sid_pal_radio_lora_modulation_params_t *mod_params) Set LoRa modulation parameters.
int32_t	sid_pal_radio_set_lora_packet_params (const sid_pal_radio_lora_packet_params_t *packet_params) Set LoRa packet parameters.
int32_t	sid_pal_radio_set_lora_cad_params (const sid_pal_radio_lora_cad_params_t *cad_params) Set LoRa CAD parameters.

int32_t	sid_pal_radio_lora_start_cad (void) Set the radio in CAD mode.
sid_pal_radio_data_rate_t	sid_pal_radio_lora_mod_params_to_data_rate (const sid_pal_radio_lora_modulation_params_t *mod_params) convert LoRa modulation params to data rate.
int32_t	sid_pal_radio_lora_data_rate_to_mod_params (sid_pal_radio_lora_modulation_params_t *mod_params, sid_pal_radio_data_rate_t data_rate, uint8_t li_enable) Convert data rate to LoRa modulation parameters.
uint32_t	sid_pal_radio_lora_time_on_air (const sid_pal_radio_lora_modulation_params_t *mod_params, const sid_pal_radio_lora_packet_params_t *packet_params, uint8_t packet_len) Get time on air for a LoRa packet.
uint32_t	sid_pal_radio_lora_cad_duration (uint8_t symbol, const sid_pal_radio_lora_modulation_params_t *mod_params) get CAD duration for a given number of symbols and lora mod_params
uint32_t	sid_pal_radio_lora_get_lora_number_of_symbols (const sid_pal_radio_lora_modulation_params_t *mod_params, uint32_t delay_micro_sec) Calculates the minimum number of symbols that takes more than delay_micro_sec of air time.
uint32_t	sid_pal_radio_get_lora_rx_done_delay (const sid_pal_radio_lora_modulation_params_t *mod_params, const sid_pal_radio_lora_packet_params_t *pkt_params) Get the time between the last bit sent (on Tx side) and the Rx done event (on Rx side)
uint32_t	sid_pal_radio_get_lora_tx_process_delay (void) Get the time between Tx schedule and the first bit of Tx.
uint32_t	sid_pal_radio_get_lora_rx_process_delay (void) Get the time of LoRa Rx processing delay.
uint32_t	sid_pal_radio_get_lora_symbol_timeout_us (sid_pal_radio_lora_modulation_params_t *mod_params, uint8_t number_of_symbol) Get LoRa symbol timeout in us.
int32_t	sid_pal_radio_set_fsk_sync_word (const uint8_t *sync_word, uint8_t sync_word_length) Radio FSK Modulation specific APIs.
int32_t	sid_pal_radio_set_fsk_whitening_seed (uint16_t seed) Set fsk whitening seed.
int32_t	sid_pal_radio_set_fsk_modulation_params (const sid_pal_radio_fsk_modulation_params_t *mod_params) Set fsk modulation parameters.
int32_t	sid_pal_radio_set_fsk_packet_params (const sid_pal_radio_fsk_packet_params_t *packet_params) Set fsk packet parameters.
sid_pal_radio_data_rate_t	sid_pal_radio_fsk_mod_params_to_data_rate (const sid_pal_radio_fsk_modulation_params_t *mp) convert fsk modulation params to data rate.
int32_t	sid_pal_radio_fsk_data_rate_to_mod_params (sid_pal_radio_fsk_modulation_params_t *mod_params, sid_pal_radio_data_rate_t data_rate) Convert data rate to fsk modulation parameters.
uint32_t	sid_pal_radio_fsk_time_on_air (const sid_pal_radio_fsk_modulation_params_t *mod_params, const sid_pal_radio_fsk_packet_params_t *packet_params, uint8_t packet_len) Get time on air for a fsk packet.
uint32_t	sid_pal_radio_fsk_get_fsk_number_of_symbols (const sid_pal_radio_fsk_modulation_params_t *mod_params, uint32_t delay_micro_secs) Calculates the minimum number of symbols that takes more than delay_micro_secs of air time.

uint32_t	sid_pal_radio_get_fsk_tx_process_delay (void) Get the time between Tx schedule and the first bit of Tx.
uint32_t	sid_pal_radio_get_fsk_rx_process_delay (void) Get the time of FSK Rx processing delay.
int32_t	sid_pal_radio_prepare_fsk_for_tx (sid_pal_radio_fsk_pkt_cfg_t *tx_pkt_cfg) Setup transmit in fsk mode.
int32_t	sid_pal_radio_prepare_fsk_for_rx (sid_pal_radio_fsk_pkt_cfg_t *rx_pkt_cfg) Setup receive in fsk mode.
int32_t	sid_pal_radio_set_fsk_crc_polynomial (uint16_t crc_polynomial, uint16_t crc_seed) Configure crc parameters.

Function Documentation

sid_pal_radio_init

```
int32_t sid_pal_radio_init (sid\_pal\_radio\_event\_notify\_t notify, sid\_pal\_radio\_irq\_handler\_t dio_irq_handler,
sid\_pal\_radio\_rx\_packet\_t * rx_packet)
```

Initializes the radio.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_event_notify_t	[in]	notify	routine called as part of bottom half processing of the radio interrupt.
sid_pal_radio_irq_handler_t	[in]	dio_irq_handler	irq handler to notify the protocol that interrupt has occurred. The protocol switches context to a task or a software interrupt to continue with the bottom half processing of the interrupt.
sid_pal_radio_rx_packet_t *	[in]	rx_packet	to protocol's receive packet buffer.

Registers the radio event callback, interrupt callback and pointer to the protocol's receive packet buffer. Sets and enables the radio interrupts.

sid_pal_radio_deinit

```
int32_t sid_pal_radio_deinit (void )
```

Deinitialize the radio.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Resets the radio and clear all configuration.

sid_pal_radio_configure_irq_mask

```
sid\_pal\_radio\_irq\_mask\_t sid_pal_radio_configure_irq_mask (sid\_pal\_radio\_irq\_mask\_t irq_mask)
```

Configure irq mask.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_irq_mask_t	[in]	irq_mask	

Configure the interrupts that the low level driver has to generate. The protocol configures the interrupts it is interested in based on modem mode. The radio driver has a default mask that can be retrieved with `sid_pal_radio_get_current_config_irq_mask`.

sid_pal_radio_get_current_config_irq_mask

```
sid_pal_radio_irq_mask_t sid_pal_radio_get_current_config_irq_mask (void )
```

Get current irq mask settings.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_irq_process

```
int32_t sid_pal_radio_irq_process (void )
```

Radio irq processing.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The function reads the irq status register and reports radio event to the phy layer through the callback routine registered as part of `sid_pal_radio_init`. The protocol after being notified on receiving a radio interrupt switches from hardware isr to software isr context to continue with bottom half processing of the radio interrupt. `sid_pal_radio_irq_process` should determine the cause of interrupt and notify the protocol of the phy event through the event handler registered as part of `sid_pal_radio_init`. On packet reception, this API has to copy the received packet from radio buffers to the rx packet registered as part of `sid_pal_radio_init`.

sid_pal_radio_set_frequency

```
int32_t sid_pal_radio_set_frequency (uint32_t freq)
```

Set the frequency for the radio.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	freq	frequency in Hz

sid_pal_radio_set_tx_power

```
int32_t sid_pal_radio_set_tx_power (int8_t power)
```

Set the radio transmit power.

Parameters

Type	Direction	Argument Name	Description
int8_t	[in]	power	tx power in dB

sid_pal_radio_get_max_tx_power

```
int32_t sid_pal_radio_get_max_tx_power (sid\_pal\_radio\_data\_rate\_t data_rate, int8_t * tx_power)
```

Get the radio max transmit power setting for a given data rate.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_data_rate_t	[in]	data_rate	rate
int8_t *	[out]	tx_power	tx power populated by this API

sid_pal_radio_set_region

```
int32_t sid_pal_radio_set_region (sid\_pal\_radio\_region\_code\_t region)
```

Set the radio region.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_region_code_t	[in]	region	region

sid_pal_radio_sleep

```
int32_t sid_pal_radio_sleep (uint32_t sleep_us)
```

Set the radio to sleep.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	sleep_us	time period for which the radio should be put in sleep mode

The protocol expects the radio to be set in lowest power consumption state possible.

sid_pal_radio_standby

```
int32_t sid_pal_radio_standby (void )
```

Set the radio to standby.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The protocol sets the radio to standby mode in the following scenarios: To wake the radio from sleep state To configure modulation parameters, packet parameters, cad, frequency, power etc

sid_pal_set_radio_busy

```
int32_t sid_pal_set_radio_busy (void )
```

Set the radio to busy state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Busy state is used for concurrent stacks operation. Set the state of radio to busy to block another stack from using the radio peripheral. Calling [sid_pal_radio_sleep\(\)](#) replaces SID_PAL_RADIO_BUSY state with SID_PAL_RADIO_SLEEP state.

sid_pal_radio_start_carrier_sense

```
int32_t sid_pal_radio_start_carrier_sense (const sid\_pal\_radio\_fsk\_cad\_params\_t * cad_params,  
sid\_pal\_radio\_cad\_param\_exit\_mode\_t exit_mode)
```

Set the radio in preamble detect mode.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_fsk_cad_params_t *	[in]	cad_params	in microseconds for how long radio is in receive mode. The upper bound of the timeout value is specific to each vendor's driver implementation.
sid_pal_radio_cad_param_exit_mode_t	[in]	exit_mode	exit mode of carrier sense operation

sid_pal_radio_start_rx

```
int32_t sid_pal_radio_start_rx (uint32_t timeout)
```

Set the radio in receive mode.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	timeout	in microseconds for how long radio is in receive mode. The upper bound of the timeout value is specific to each vendor's driver implementation.

sid_pal_radio_start_continuous_rx

```
int32_t sid_pal_radio_start_continuous_rx (void )
```

Set the radio to continuous receive.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_set_rx_duty_cycle

```
int32_t sid_pal_radio_set_rx_duty_cycle (uint32_t rx_time, uint32_t sleep_time)
```

Set the receive duty cycle.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	rx_time	time in milliseconds spent by radio in receive.
uint32_t	[in]	sleep_time	time in milliseconds spent by radio in sleep.

Configures the radio receive duty cycle. The protocol uses this API to set the radio to alternate between receive and sleep states. The radio in this mode should not interrupt the protocol unless it detects a valid packet.

sid_pal_radio_set_tx_continuous_wave

```
int32_t sid_pal_radio_set_tx_continuous_wave (uint32_t freq, int8_t power)
```

Set the transmit continuous wave.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	freq	frequency in Hz on which to transmit a continuous wave.
int8_t	[in]	power	power in dB at which the continuous wave has to be transmitted

Confiure the radio to transmit a continuous wave. This API is used for diagnostics mode only

sid_pal_radio_set_tx_payload

```
int32_t sid_pal_radio_set_tx_payload (const uint8_t * buffer, uint8_t size)
```

Set transmit payload for the radio to transmit.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	buffer	pointer to the buffer containing the tx packet.
uint8_t	[in]	size	length of the packet that needs to be transmitted.

Writes the payload and payload length to radio buffers but the packet is not transmitted on the air.

sid_pal_radio_start_tx

int32_t sid_pal_radio_start_tx (uint32_t timeout)

Start packet transmission.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	timeout	timeout in microseconds

Starts the packet transmission. This needs to be invoked after all the radio configuration viz modulation params, packet params, freq, power, payload and payload length are set atleast once. The radio should be able to transmit the packet within the timeout specified through this API. If it fails to transmit the packet within the stipulated timeout value, the radio driver should generate a interrupt with tx timeout as the reason

sid_pal_radio_get_status

uint8_t sid_pal_radio_get_status (void)

Get the radio state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The protocol uses this API to query its current state

sid_pal_radio_get_modem_mode

sid_pal_radio_modem_mode_t sid_pal_radio_get_modem_mode (void)

Get the current radio modem mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Supported modem modes are LoRa and FSK

sid_pal_radio_set_modem_mode

```
int32_t sid_pal_radio_set_modem_mode (sid_pal_radio_modem_mode_t mode)
```

Set radio modem mode.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_modem_mode_t	[in]	mode	LoRa = 1 or FSK = 0

The driver should configure all the parameters to operate in the desired mode. Supported modem modes are LoRa and FSK

sid_pal_radio_is_channel_free

```
int32_t sid_pal_radio_is_channel_free (uint32_t freq, int16_t threshold, uint32_t delay_us, bool * is_channel_free)
```

Check the channel noise level for a given rssi.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	freq	frequency in Hz to measure noise level.
int16_t	[in]	threshold	rssi threshold in dBm
uint32_t	[in]	delay_us	period in microseconds for radio to sense the medium
bool *	[out]	is_channel_free	boolean to store the result of the operation

The radio should sense the medium in the frequency specified by parameter freq for a period specified by delay_us, compute the rssi, and compare the computed rssi with the passed threshold specified by the parameter threshold. If the computed rssi is greater than threshold, the parameter is_channel_free is set to false otherwise it is set to true. If any of the above operations fail, appropriate error value will be returned.

sid_pal_radio_get_chan_noise

```
int32_t sid_pal_radio_get_chan_noise (uint32_t freq, int16_t * noise)
```

Compute the noise sensed by radio at a particular frequency.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	freq	frequency in Hz on which noise level is to be measured.
int16_t *	[out]	noise	pointer to variable to store the avg noise in dBm

sid_pal_radio_rssi

```
int16_t sid_pal_radio_rssi (void )
```

Get RSSI at radio's current configured frequency.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The frequency on which RSSI needs to be measured needs to be set before calling this API.

sid_pal_radio_random

```
int32_t sid_pal_radio_random (uint32_t * random)
```

Get a random number from radio.

Parameters

Type	Direction	Argument Name	Description
uint32_t *	[out]	random	pointer to store the random number

sid_pal_radio_get_ant_dbi

```
int16_t sid_pal_radio_get_ant_dbi (void )
```

Get antenna gain in dBi.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_get_cca_level_adjust

```
int32_t sid_pal_radio_get_cca_level_adjust (sid\_pal\_radio\_data\_rate\_t data_rate, int8_t * adj_level)
```

Get the cca adjustment in dB.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_data_rate_t	[in]	data_rate	rate
int8_t *	[out]	adj_level	pointer to CCA level adjustment in dB get by this API

sid_pal_radio_get_radio_state_transition_delays

```
int32_t sid_pal_radio_get_radio_state_transition_delays (sid\_pal\_radio\_state\_transition\_timings\_t * state_delay)
```

Get the delay in microseconds to switch between different radio states.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_state_transition_timings_t *	[out]	state_delay	of switching delay between different radio states

sid_pal_radio_is_cad_exit_mode

int32_t sid_pal_radio_is_cad_exit_mode ([sid_pal_radio_cad_param_exit_mode_t](#) mode)

Check the CAD exit mode.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_cad_param_exit_mode_t	[in]	mode	CAD exit mode.

Check with supported CAD exit modes

sid_pal_radio_set_lora_symbol_timeout

int32_t sid_pal_radio_set_lora_symbol_timeout (uint8_t num_of_symbols)

Radio LoRa Modulation specific APIs.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	num_of_symbols	number of symbols the radio has to detect before reporting a rx timeout interrupt

Set LoRa symbol timeout.

sid_pal_radio_set_lora_sync_word

int32_t sid_pal_radio_set_lora_sync_word (uint16_t sync_word)

Set LoRa sync word.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	sync_word	

sid_pal_radio_set_lora_modulation_params

int32_t sid_pal_radio_set_lora_modulation_params (const [sid_pal_radio_lora_modulation_params_t](#) * mod_params)

Set LoRa modulation parameters.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_lora_modulation_params_t *	[in]	mod_params	pointer to Sidewalk LoRa modulation params.

sid_pal_radio_set_lora_packet_params

int32_t sid_pal_radio_set_lora_packet_params (const [sid_pal_radio_lora_packet_params_t](#) * packet_params)

Set LoRa packet parameters.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_lora_packet_params_t *	[in]	packet_params	pointer to Sidewalk LoRa packet params.

sid_pal_radio_set_lora_cad_params

int32_t sid_pal_radio_set_lora_cad_params (const [sid_pal_radio_lora_cad_params_t](#) * cad_params)

Set LoRa CAD parameters.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_lora_cad_params_t *	[in]	cad_params	pointer to Sidewalk CAD params.

sid_pal_radio_lora_start_cad

int32_t sid_pal_radio_lora_start_cad (void)

Set the radio in CAD mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_lora_mod_params_to_data_rate

[sid_pal_radio_data_rate_t](#) sid_pal_radio_lora_mod_params_to_data_rate (const [sid_pal_radio_lora_modulation_params_t](#) * mod_params)

convert LoRa modulation params to data rate.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_lora_modulation_params_t *	[in]	mod_params	pointer to LoRa modulation params.

sid_pal_radio_lora_data_rate_to_mod_params

```
int32_t sid_pal_radio_lora_data_rate_to_mod_params (sid_pal_radio_lora_modulation_params_t * mod_params,
sid_pal_radio_data_rate_t data_rate, uint8_t li_enable)
```

Convert data rate to LoRa modulation parameters.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_lora_modulation_params_t *	[out]	mod_params	pointer to LoRa modulation params.
sid_pal_radio_data_rate_t	[in]	data_rate	rate.
uint8_t	[in]	li_enable	enable/disable long interleaver mode

sid_pal_radio_lora_time_on_air

```
uint32_t sid_pal_radio_lora_time_on_air (const sid_pal_radio_lora_modulation_params_t * mod_params, const
sid_pal_radio_lora_packet_params_t * packet_params, uint8_t packet_len)
```

Get time on air for a LoRa packet.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_lora_modulation_params_t *	[in]	mod_params	pointer to LoRa modulation params.
const sid_pal_radio_lora_packet_params_t *	[in]	packet_params	pointer to LoRa packet params.
uint8_t	[in]	packet_len	length of the packet that needs to be transmitted.

sid_pal_radio_lora_cad_duration

```
uint32_t sid_pal_radio_lora_cad_duration (uint8_t symbol, const sid_pal_radio_lora_modulation_params_t * mod_params)
```

get CAD duration for a given number of symbols and lora mod_params

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	symbol	symbol timeout for the CAD configuration.
const sid_pal_radio_lora_modulation_params_t *	[in]	mod_params	pointer to LoRa modulation params.

sid_pal_radio_lora_get_lora_number_of_symbols

```
uint32_t sid_pal_radio_lora_get_lora_number_of_symbols (const sid_pal_radio_lora_modulation_params_t * mod_params,
uint32_t delay_micro_sec)
```

Calculates the minimum number of symbols that takes more than delay_micro_sec of air time.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_lora_modulation_params_t *	[in]	mod_params	Current modulation parameters that phy uses. If null, zero will be returned.
uint32_t	[in]	delay_micro_sec	Input amount of time that will be translated to number of symbols.

In the case of a fractional number of symbols, the return value is rounded up to the next integer. Does not affect the radio state and can be executed without radio ownership. In the case of an error, 0 is returned.

sid_pal_radio_get_lora_rx_done_delay

```
uint32_t sid_pal_radio_get_lora_rx_done_delay (const sid\_pal\_radio\_lora\_modulation\_params\_t * mod_params, const sid\_pal\_radio\_lora\_packet\_params\_t * pkt_params)
```

Get the time between the last bit sent (on Tx side) and the Rx done event (on Rx side)

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_lora_modulation_params_t *	[in]	mod_params	Pointer to a structure holding the LoRa modulation parameters used for the computation
const sid_pal_radio_lora_packet_params_t *	[in]	pkt_params	Pointer to a structure holding the LoRa packet parameters used for the computation

sid_pal_radio_get_lora_tx_process_delay

```
uint32_t sid_pal_radio_get_lora_tx_process_delay (void )
```

Get the time between Tx schedule and the first bit of Tx.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_get_lora_rx_process_delay

```
uint32_t sid_pal_radio_get_lora_rx_process_delay (void )
```

Get the time of LoRa Rx processing delay.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_get_lora_symbol_timeout_us

```
uint32_t sid_pal_radio_get_lora_symbol_timeout_us (sid_pal_radio_lora_modulation_params_t * mod_params, uint8_t number_of_symbol)
```

Get LoRa symbol timeout in us.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_lora_modulation_params_t *	[in]	mod_params	pointer to Sidewalk LoRa modulation params.
uint8_t	[in]	number_of_symbol	number of symbol

sid_pal_radio_set_fsk_sync_word

```
int32_t sid_pal_radio_set_fsk_sync_word (const uint8_t * sync_word, uint8_t sync_word_length)
```

Radio FSK Modulation specific APIs.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	sync_word	
uint8_t	[in]	sync_word_length	

Set fsk sync word

sid_pal_radio_set_fsk_whitening_seed

```
int32_t sid_pal_radio_set_fsk_whitening_seed (uint16_t seed)
```

Set fsk whitening seed.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	seed	

sid_pal_radio_set_fsk_modulation_params

```
int32_t sid_pal_radio_set_fsk_modulation_params (const sid_pal_radio_fsk_modulation_params_t * mod_params)
```

Set fsk modulation parameters.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_fsk_modulation_params_t *	[in]	mod_params	pointer to Sidewalk fsk modulation params.

sid_pal_radio_set_fsk_packet_params

```
int32_t sid_pal_radio_set_fsk_packet_params (const sid_pal_radio_fsk_packet_params_t * packet_params)
```

Set fsk packet parameters.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_fsk_packet_params_t *	[in]	packet_params	pointer to Sidewalk fsk packet params.

sid_pal_radio_fsk_mod_params_to_data_rate

```
sid_pal_radio_data_rate_t sid_pal_radio_fsk_mod_params_to_data_rate (const sid_pal_radio_fsk_modulation_params_t * mp)
```

convert fsk modulation params to data rate.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_fsk_modulation_params_t *	[in]	mp	pointer to fsk modulation params.

sid_pal_radio_fsk_data_rate_to_mod_params

```
int32_t sid_pal_radio_fsk_data_rate_to_mod_params (sid_pal_radio_fsk_modulation_params_t * mod_params, sid_pal_radio_data_rate_t data_rate)
```

Convert data rate to fsk modulation parameters.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_fsk_modulation_params_t *	[out]	mod_params	pointer to fsk modulation params.
sid_pal_radio_data_rate_t	[in]	data_rate	data rate.

sid_pal_radio_fsk_time_on_air

```
uint32_t sid_pal_radio_fsk_time_on_air (const sid_pal_radio_fsk_modulation_params_t * mod_params, const sid_pal_radio_fsk_packet_params_t * packet_params, uint8_t packet_len)
```

Get time on air for a fsk packet.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_fsk_modulation_params_t *	[in]	mod_params	pointer to fsk modulation params.
const sid_pal_radio_fsk_packet_params_t *	[in]	packet_params	poiner to fsk packet params.

Type	Direction	Argument Name	Description
uint8_t	[in]	packet_len	length of the packet that needs to be transmitted in bytes.

sid_pal_radio_fsk_get_fsk_number_of_symbols

```
uint32_t sid_pal_radio_fsk_get_fsk_number_of_symbols (const sid\_pal\_radio\_fsk\_modulation\_params\_t * mod_params,
uint32_t delay_micro_secs)
```

Calculates the minimum number of symbols that takes more than delay_micro_secs of air time.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_radio_fsk_modulation_params_t *	[in]	mod_params	Current modulation parameters that phy uses. If null, zero will be returned.
uint32_t	[in]	delay_micro_secs	Input amount of time in uS that will be translated to number of symbols.

In the case of a fractional number of symbols, the return value is rounded up to the next integer. Does not affect the radio state and can be executed without radio ownership. In the case of an error, 0 is returned.

sid_pal_radio_get_fsk_tx_process_delay

```
uint32_t sid_pal_radio_get_fsk_tx_process_delay (void )
```

Get the time between Tx schedule and the first bit of Tx.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_get_fsk_rx_process_delay

```
uint32_t sid_pal_radio_get_fsk_rx_process_delay (void )
```

Get the time of FSK Rx processing delay.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_radio_prepare_fsk_for_tx

```
int32_t sid_pal_radio_prepare_fsk_for_tx (sid\_pal\_radio\_fsk\_pkt\_cfg\_t * tx_pkt_cfg)
```

Setup transmit in fsk mode.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_fsk_pkt_cfg_t *	[inout]	tx_pkt_cfg	tx packet config

This API is used to configure the sync word, packet params perform crc, and data whitening on the payload, and determine the packet length. This API needs to be called before calling `sid_pal_radio_set_payload` and `sid_pal_radio_start_tx` in FSK mode.

sid_pal_radio_prepare_fsk_for_rx

```
int32_t sid_pal_radio_prepare_fsk_for_rx (sid\_pal\_radio\_fsk\_pkt\_cfg\_t * rx_pkt_cfg)
```

Setup receive in fsk mode.

Parameters

Type	Direction	Argument Name	Description
sid_pal_radio_fsk_pkt_cfg_t *	[in]	rx_pkt_cfg	pointer to fsk packet config

This API is used to configure the sync word and packet params. This API needs to be called before calling `sid_pal_radio_start_rx` in FSK mode.

sid_pal_radio_set_fsk_crc_polynomial

```
int32_t sid_pal_radio_set_fsk_crc_polynomial (uint16_t crc_polynomial, uint16_t crc_seed)
```

Configure crc parameters.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	crc_polynomial	polynomial for crc calculation
uint16_t	[in]	crc_seed	seed value for crc calculation

This API is used to configure the crc polynomial and seed. This API needs to be called before tx and rx in FSK mode.

Type definitions

Type definitions

Modules

[sid_pal_radio_rx_packet_t](#)

[sid_pal_radio_packet_cfg_t](#)

[sid_pal_radio_tx_packet_t](#)

[sid_pal_radio_state_transition_timings_t](#)

Enumerations

```
enum    sid\_pal\_radio\_region\_code\_t {
        SID_PAL_RADIO_RC_NONE = 0
        SID_PAL_RADIO_RC_NA = 1
        SID_PAL_RADIO_RC_EU = 2
        SID_PAL_RADIO_RC_JP = 3
        SID_PAL_RADIO_RC_MAX
    }
    Sidewalk Phy Radio Region For Regional Config.

enum    sid\_pal\_radio\_modem\_mode\_t {
        SID_PAL_RADIO_MODEM_MODE_FSK = 0
        SID_PAL_RADIO_MODEM_MODE_LORA = 1
    }
    Sidewalk Phy Radio Modem Mode.

enum    sid\_pal\_radio\_events\_t {
        SID_PAL_RADIO_EVENT_UNKNOWN = 0
        SID_PAL_RADIO_EVENT_TX_DONE = 1
        SID_PAL_RADIO_EVENT_RX_DONE = 2
        SID_PAL_RADIO_EVENT_CAD_DONE = 3
        SID_PAL_RADIO_EVENT_CAD_TIMEOUT = 4
        SID_PAL_RADIO_EVENT_RX_ERROR = 5
        SID_PAL_RADIO_EVENT_TX_TIMEOUT = 6
        SID_PAL_RADIO_EVENT_RX_TIMEOUT = 7
        SID_PAL_RADIO_EVENT_CS_DONE = 8
        SID_PAL_RADIO_EVENT_CS_TIMEOUT = 9
        SID_PAL_RADIO_EVENT_HEADER_ERROR = 10
        SID_PAL_RADIO_EVENT_SYNC_DET = 11
    }
    Sidewalk Phy Radio Event.

enum    sid\_pal\_radio\_data\_rate\_t {
        SID_PAL_RADIO_DATA_RATE_INVALID = 0
        SID_PAL_RADIO_DATA_RATE_2KBPS = 1
        SID_PAL_RADIO_DATA_RATE_22KBPS = 2
        SID_PAL_RADIO_DATA_RATE_50KBPS = 3
        SID_PAL_RADIO_DATA_RATE_150KBPS = 4
        SID_PAL_RADIO_DATA_RATE_250KBPS = 5
        SID_PAL_RADIO_DATA_RATE_12_5KBPS = 6
        SID_PAL_RADIO_DATA_RATE_CUSTOM = 7
        SID_PAL_RADIO_DATA_RATE_MAX_NUM = SID_PAL_RADIO_DATA_RATE_CUSTOM
    }
```

```

}
Sidewalk Phy Radio
Data Rate.

enum sid_pal_radio_cad_param_exit_mode_t {
    SID_PAL_RADIO_CAD_EXIT_MODE_CS_ONLY = 0x00
    SID_PAL_RADIO_CAD_EXIT_MODE_CS_RX = 0x01
    SID_PAL_RADIO_CAD_EXIT_MODE_CS_LBT = 0x10
    SID_PAL_RADIO_CAD_EXIT_MODE_CS_LBT_RX = 0x11
    SID_PAL_RADIO_CAD_EXIT_MODE_ED_ONLY = 0x100
    SID_PAL_RADIO_CAD_EXIT_MODE_ED_RX = 0x101
    SID_PAL_RADIO_CAD_EXIT_MODE_ED_LBT = 0x110
    SID_PAL_RADIO_CAD_EXIT_MODE_NONE = 0x10000
}
Sidewalk Phy Radio CAD (Channel Activity Detection) Exit Mode.

enum sid_pal_radio_irq_mask_t {
    RADIO_IRQ_NONE = (0 << 0)
    RADIO_IRQ_TX_DONE = (1 << 0)
    RADIO_IRQ_RX_DONE = (1 << 1)
    RADIO_IRQ_PREAMBLE_DETECT = (1 << 2)
    RADIO_IRQ_VALID_SYNC_WORD = (1 << 3)
    RADIO_IRQ_VALID_HEADER = (1 << 4)
    RADIO_IRQ_ERROR_HEADER = (1 << 5)
    RADIO_IRQ_ERROR_CRC = (1 << 6)
    RADIO_IRQ_CAD_DONE = (1 << 7)
    RADIO_IRQ_CAD_DETECT = (1 << 8)
    RADIO_IRQ_TXRX_TIMEOUT = (1 << 9)
    RADIO_IRQ_ALL = ((1 << 10) - 1)
}
Radio IRQ MASK.

```

Typedefs

```

typedef void(*) sid_pal_radio_event_notify_t(sid_pal_radio_events_t events)
Radio event callback.

typedef void(*) sid_pal_radio_irq_handler_t(void)
Radio interrupt callback.

```

Macros

```

#define SID_PAL_RADIO_RX_PAYLOAD_MAX_SIZE 255

#define RADIO_ERROR_NONE 0
Radio Error Codes.

#define RADIO_ERROR_NOT_SUPPORTED -1

#define RADIO_ERROR_INVALID_PARAMS -2

#define RADIO_ERROR_IO_ERROR -3

#define RADIO_ERROR_BUSY -4

#define RADIO_ERROR_NOMEM -5

#define RADIO_ERROR_HARDWARE_ERROR -6

```

```
#define RADIO_ERROR_INVALID_STATE -7

#define RADIO_ERROR_GENERIC -8

#define RADIO_ERROR_PKT_CHECK_DCR_LIMIT -9

#define RADIO_ERROR_PKT_CHECK_REG_LIMIT -10

#define RADIO_ERROR_VENDOR_FIRST -64

#define RADIO_ERROR_VENDOR_LAST -255

#define SID_PAL_RADIO_UNKNOWN 0
Sidewalk Phy Radio State.

#define SID_PAL_RADIO_STANDBY 1

#define SID_PAL_RADIO_SLEEP 2

#define SID_PAL_RADIO_RX 3

#define SID_PAL_RADIO_TX 4

#define SID_PAL_RADIO_CAD 5

#define SID_PAL_RADIO_STANDBY_XOSC 6

#define SID_PAL_RADIO_RX_DC 7

#define SID_PAL_RADIO_BUSY 8
```

Enumeration Documentation

sid_pal_radio_region_code_t

sid_pal_radio_region_code_t

Sidewalk Phy Radio Region For Regional Config.

Enumerator	
SID_PAL_RADIO_RC_NONE	Region none
SID_PAL_RADIO_RC_NA	Region North America
SID_PAL_RADIO_RC_EU	Region EU
SID_PAL_RADIO_RC_JP	Region JP
SID_PAL_RADIO_RC_MAX	Region max

sid_pal_radio_modem_mode_t

sid_pal_radio_modem_mode_t

Sidewalk Phy Radio Modem Mode.

Enumerator	
SID_PAL_RADIO_MODEM_MODE_FSK	Frequency Shift Keying (FSK) modem mode
SID_PAL_RADIO_MODEM_MODE_LORA	Long Range (LoRa) modem mode

sid_pal_radio_events_t

sid_pal_radio_events_t

Sidewalk Phy Radio Event.

Enumerator	
SID_PAL_RADIO_EVENT_UNKNOWN	Unknown event
SID_PAL_RADIO_EVENT_TX_DONE	Transmission done event
SID_PAL_RADIO_EVENT_RX_DONE	Reception done event
SID_PAL_RADIO_EVENT_CAD_DONE	Channel Activity Detection (CAD) done event
SID_PAL_RADIO_EVENT_CAD_TIMEOUT	CAD timeout event
SID_PAL_RADIO_EVENT_RX_ERROR	Reception error event
SID_PAL_RADIO_EVENT_TX_TIMEOUT	Transmission timeout event
SID_PAL_RADIO_EVENT_RX_TIMEOUT	Reception timeout event
SID_PAL_RADIO_EVENT_CS_DONE	Carrier Sense (CS) done event
SID_PAL_RADIO_EVENT_CS_TIMEOUT	CS timeout event
SID_PAL_RADIO_EVENT_HEADER_ERROR	Header error event
SID_PAL_RADIO_EVENT_SYNC_DET	Sync detection event

sid_pal_radio_data_rate_t

sid_pal_radio_data_rate_t

Sidewalk Phy Radio Data Rate.

Enumerator	
SID_PAL_RADIO_DATA_RATE_INVALID	Invalid data rate
SID_PAL_RADIO_DATA_RATE_2KBPS	2 Kbps data rate
SID_PAL_RADIO_DATA_RATE_22KBPS	22 Kbps data rate
SID_PAL_RADIO_DATA_RATE_50KBPS	50 Kbps data rate
SID_PAL_RADIO_DATA_RATE_150KBPS	150 Kbps data rate
SID_PAL_RADIO_DATA_RATE_250KBPS	250 Kbps data rate
SID_PAL_RADIO_DATA_RATE_12_5KBPS	12.5 Kbps data rate
SID_PAL_RADIO_DATA_RATE_CUSTOM	Custom data rate
SID_PAL_RADIO_DATA_RATE_MAX_NUM	0 is not a valid data rate

sid_pal_radio_cad_param_exit_mode_t

sid_pal_radio_cad_param_exit_mode_t

Sidewalk Phy Radio CAD (Channel Activity Detection) Exit Mode.

Enumerator	
SID_PAL_RADIO_CAD_EXIT_MODE_CS_ONLY	Carrier sense only

SID_PAL_RADIO_CAD_EXIT_MODE_CS_RX	Carrier sense followed by Rx
SID_PAL_RADIO_CAD_EXIT_MODE_CS_LBT	Carrier sense followed by Tx
SID_PAL_RADIO_CAD_EXIT_MODE_CS_LBT_RX	Carrier sense followed by Tx then RX
SID_PAL_RADIO_CAD_EXIT_MODE_ED_ONLY	Energy detect only
SID_PAL_RADIO_CAD_EXIT_MODE_ED_RX	Energy detect followed by Rx
SID_PAL_RADIO_CAD_EXIT_MODE_ED_LBT	Energy detect followed by Tx
SID_PAL_RADIO_CAD_EXIT_MODE_NONE	No CAD mode set

sid_pal_radio_irq_mask_t

```
sid_pal_radio_irq_mask_t
```

Radio IRQ MASK.

Enumerator	
RADIO_IRQ_NONE	No interrupt
RADIO_IRQ_TX_DONE	Transmission done interrupt
RADIO_IRQ_RX_DONE	Reception done interrupt
RADIO_IRQ_PREAMBLE_DETECT	Preamble detection interrupt
RADIO_IRQ_VALID_SYNC_WORD	Valid sync word interrupt
RADIO_IRQ_VALID_HEADER	Valid header interrupt
RADIO_IRQ_ERROR_HEADER	Header error interrupt
RADIO_IRQ_ERROR_CRC	CRC error interrupt
RADIO_IRQ_CAD_DONE	Channel Activity Detection (CAD) done interrupt
RADIO_IRQ_CAD_DETECT	CAD detection interrupt
RADIO_IRQ_TXRX_TIMEOUT	Transmission or reception timeout interrupt
RADIO_IRQ_ALL	All interrupts

Typedef Documentation

sid_pal_radio_event_notify_t

```
typedef void(* sid_pal_radio_event_notify_t) (sid_pal_radio_events_t events) )(sid_pal_radio_events_t events)
```

Radio event callback.

sid_pal_radio_irq_handler_t

```
typedef void(* sid_pal_radio_irq_handler_t) (void) )(void)
```

Radio interrupt callback.

sid_pal_radio_rx_packet_t

Sidewalk Phy received LoRa packet status.

Public Attributes

sid_pal_radio_data_rate_t	data_rate
sid_pal_radio_lora_rx_packet_status_t	lora_rx_packet_status
sid_pal_radio_fsk_rx_packet_status_t	fsk_rx_packet_status
struct sid_timespec	rcv_tm
uint8_t	rcv_payload
uint8_t	payload_len

Public Attribute Documentation

data_rate

```
sid_pal_radio_data_rate_t sid_pal_radio_rx_packet_t::data_rate
```

Data rate at which the packet was received

lora_rx_packet_status

```
sid_pal_radio_lora_rx_packet_status_t sid_pal_radio_rx_packet_t::lora_rx_packet_status
```

LoRa-specific packet status

fsk_rx_packet_status

```
sid_pal_radio_fsk_rx_packet_status_t sid_pal_radio_rx_packet_t::fsk_rx_packet_status
```

FSK-specific packet status

rcv_tm

```
struct sid_timespec sid_pal_radio_rx_packet_t::rcv_tm
```

Timestamp when the packet was received

rcv_payload

```
uint8_t sid_pal_radio_rx_packet_t::rcv_payload[255]
```

Buffer to store the received payload

payload_len

```
uint8_t sid_pal_radio_rx_packet_t::payload_len
```

Length of the received payload

sid_pal_radio_packet_cfg_t

Sidewalk Phy settings to configure radio prior to transmission and reception.

Public Attributes

sid_pal_radio_data_rate_t	data_rate
uint8_t	channel
uint8_t	invert_IQ
uint16_t	preamble_len
uint32_t	symbol_timeout
bool	crc_enabled

Public Attribute Documentation

data_rate

sid_pal_radio_data_rate_t sid_pal_radio_packet_cfg_t::data_rate

Data rate for the packet

channel

uint8_t sid_pal_radio_packet_cfg_t::channel

Channel on which the packet will be transmitted or received

invert_IQ

uint8_t sid_pal_radio_packet_cfg_t::invert_IQ

Flag to indicate if IQ inversion is enabled

preamble_len

uint16_t sid_pal_radio_packet_cfg_t::preamble_len

Length of the preamble

symbol_timeout

```
uint32_t sid_pal_radio_packet_cfg_t::symbol_timeout
```

Symbol timeout value

crc_enabled

```
bool sid_pal_radio_packet_cfg_t::crc_enabled
```

Flag to indicate if CRC is enabled

sid_pal_radio_tx_packet_t

Sidewalk Phy transmit packet configuration.

Public Attributes

sid_pal_radio_packet_cfg_t	packet_cfg
int8_t	tx_power_in_dbm
uint8_t *	tx_payload
uint8_t	payload_len
uint32_t	timeout

Public Attribute Documentation

packet_cfg

sid_pal_radio_packet_cfg_t sid_pal_radio_tx_packet_t::packet_cfg

Configuration for the packet

tx_power_in_dbm

int8_t sid_pal_radio_tx_packet_t::tx_power_in_dbm

Transmit power in dBm

tx_payload

uint8_t* sid_pal_radio_tx_packet_t::tx_payload

Pointer to the payload to be transmitted

payload_len

uint8_t sid_pal_radio_tx_packet_t::payload_len

Length of the payload

timeout

uint32_t sid_pal_radio_tx_packet_t::timeout

Timeout for the transmission

sid_pal_radio_state_transition_timings_t

Sidewalk radio state transition timings.

Public Attributes

uint32_t	sleep_to_full_power_us
uint32_t	full_power_to_sleep_us
uint32_t	rx_to_tx_us
uint32_t	tx_to_rx_us
uint32_t	tcxo_delay_us

Public Attribute Documentation

sleep_to_full_power_us

uint32_t sid_pal_radio_state_transition_timings_t:sleep_to_full_power_us

Time to transition from sleep to full power state, in microseconds

full_power_to_sleep_us

uint32_t sid_pal_radio_state_transition_timings_t:full_power_to_sleep_us

Time to transition from full power to sleep state, in microseconds

rx_to_tx_us

uint32_t sid_pal_radio_state_transition_timings_t:rx_to_tx_us

Time to transition from receive to transmit state, in microseconds

tx_to_rx_us

uint32_t sid_pal_radio_state_transition_timings_t:tx_to_rx_us

Time to transition from transmit to receive state, in microseconds

tcxo_delay_us

uint32_t sid_pal_radio_state_transition_timings_t:tcxo_delay_us

Time delay for TCXO (Temperature Compensated Crystal Oscillator), in microseconds

SWI Interface

SWI Interface

The SWI Interface module provides interfaces for implementing software interrupts (SWI) within the Sidewalk SDK. These interfaces enable developers to handle asynchronous events and improve the responsiveness of their applications by leveraging platform-specific SWI mechanisms.

⚠ WARNING ⚠: SWI thread priority in RTOS must stay high enough if you don't want to miss radio event

Typedefs

typedef void(*

sid_pal_swi_cb_t)(void)

SWI callback.

Functions

- sid_error_t

sid_pal_swi_init(void)

Init the SWI handler for protocol processing.
- sid_error_t

sid_pal_swi_trigger(void)

Trigger the SWI to run.
- sid_error_t

sid_pal_swi_start(sid_pal_swi_cb_t event_callback)

Start the SWI.
- sid_error_t

sid_pal_swi_stop(void)

Stop the SWI.
- sid_error_t

sid_pal_swi_deinit(void)

Init the SWI handler for protocol processing.

Typedef Documentation

sid_pal_swi_cb_t

typedef void(* sid_pal_swi_cb_t) (void))(void)

SWI callback.

Note

- The callback which will be executed in SWI context

Function Documentation

sid_pal_swi_init

sid_error_t sid_pal_swi_init (void)

Init the SWI handler for protocol processing.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Function initializes SWI for triggering events.

sid_pal_swi_trigger

```
sid_error_t sid_pal_swi_trigger (void )
```

Trigger the SWI to run.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Function triggers SWI to run.

sid_pal_swi_start

```
sid_error_t sid_pal_swi_start (sid\_pal\_swi\_cb\_t event_callback)
```

Start the SWI.

Parameters

Type	Direction	Argument Name	Description
sid_pal_swi_cb_t	[in]	event_callback	Pointer to the callback function the SWI will trigger

Function triggers SWI to run.

sid_pal_swi_stop

```
sid_error_t sid_pal_swi_stop (void )
```

Stop the SWI.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Function triggers SWI to run.

sid_pal_swi_deinit

```
sid_error_t sid_pal_swi_deinit (void )
```

Init the SWI handler for protocol processing.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Function initializes SWI for triggering events.

Security and Crypto

Security and Crypto

The Security and Crypto module provides interfaces for implementing cryptographic operations within the Sidewalk SDK. These interfaces ensure secure data transmission and storage by offering functionalities such as encryption, decryption, hashing, and key management, enabling developers to protect sensitive information across different hardware platforms.

Modules

[Type definitions](#)

Functions

sid_error_t	sid_pal_crypto_init (void) Initialize sid_pal crypto HAL.
sid_error_t	sid_pal_crypto_deinit (void) Deinitialize crypto HAL.
sid_error_t	sid_pal_crypto_rand (uint8_t *rand, size_t size) Generate random number.
sid_error_t	sid_pal_crypto_hash (sid_pal_hash_params_t *params) Generate hash.
sid_error_t	sid_pal_crypto_hmac (sid_pal_hmac_params_t *params) Generate HMAC.
sid_error_t	sid_pal_crypto_aes_crypt (sid_pal_aes_params_t *params) Encrypt or decrypt using following AES algorithm.
sid_error_t	sid_pal_crypto_aead_crypt (sid_pal_aead_params_t *params) Encrypt or decrypt using AEAD algorithm.
sid_error_t	sid_pal_crypto_ecc_dsa (sid_pal_dsa_params_t *params) Sign or verify elliptic curve digital signature using given algorithm.
sid_error_t	sid_pal_crypto_ecc_ecdh (sid_pal_ecdh_params_t *params) Generate shared secret using private key and public key.
sid_error_t	sid_pal_crypto_ecc_key_gen (sid_pal_ecc_key_gen_params_t *params) Generate ECC key pair using given algorithm.

Function Documentation

sid_pal_crypto_init

```
sid_error_t sid_pal_crypto_init (void )
```

Initialize sid_pal crypto HAL.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function initializes the SID PAL crypto HAL and prepares it for cryptographic operations.

sid_pal_crypto_deinit

```
sid_error_t sid_pal_crypto_deinit (void )
```

Deinitialize crypto HAL.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function deinitializes the SID PAL crypto HAL and releases any resources that were allocated.

sid_pal_crypto_rand

```
sid_error_t sid_pal_crypto_rand (uint8_t * rand, size_t size)
```

Generate random number.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	rand	Pointer to rand buffer.
size_t	[in]	size	Size of rand number

This function generates a random number and stores it in the provided buffer.

sid_pal_crypto_hash

```
sid_error_t sid_pal_crypto_hash (sid_pal_hash_params_t * params)
```

Generate hash.

Parameters

Type	Direction	Argument Name	Description
sid_pal_hash_params_t *	[inout]	params	Pointer to the hash parameters.

SHA256 and SHA512 is now supported.

This function generates a hash using the specified parameters.

sid_pal_crypto_hmac

```
sid_error_t sid_pal_crypto_hmac (sid\_pal\_hmac\_params\_t * params)
```

Generate HMAC.

Parameters

Type	Direction	Argument Name	Description
sid_pal_hmac_params_t *	[inout]	params	Pointer to the hash parameters.

HMAC/SHA256 and HMAC/SHA512 is now supported.

This function generates an HMAC using the specified parameters.

sid_pal_crypto_aes_crypt

```
sid_error_t sid_pal_crypto_aes_crypt (sid\_pal\_aes\_params\_t * params)
```

Encrypt or decrypt using following AES algorithm.

Parameters

Type	Direction	Argument Name	Description
sid_pal_aes_params_t *	[inout]	params	Pointer to AES parameters.

AES-CMAC AES-CTR are supported.

This function encrypts or decrypts data using the specified AES algorithm.

sid_pal_crypto_aead_crypt

```
sid_error_t sid_pal_crypto_aead_crypt (sid\_pal\_aead\_params\_t * params)
```

Encrypt or decrypt using AEAD algorithm.

Parameters

Type	Direction	Argument Name	Description
sid_pal_aead_params_t *	[inout]	params	Pointer to the AEAD parameters.

This function encrypts or decrypts data using the specified AEAD algorithm.

sid_pal_crypto_ecc_dsa

```
sid_error_t sid_pal_crypto_ecc_dsa (sid\_pal\_dsa\_params\_t * params)
```

Sign or verify elliptic curve digital signature using given algorithm.

Parameters

Type	Direction	Argument Name	Description
sid_pal_dsa_params_t *	[inout]	params	Pointer to the ECC DSA parameters.

This function signs or verifies an elliptic curve digital signature using the specified parameters.

sid_pal_crypto_ecc_ecdh

```
sid_error_t sid_pal_crypto_ecc_ecdh (sid_pal_ecdh_params_t * params)
```

Generate shared secret using private key and public key.

Parameters

Type	Direction	Argument Name	Description
sid_pal_ecdh_params_t *	[inout]	params	Pointer to the ECDH parameters.

This function generates a shared secret using the specified private key and public key.

sid_pal_crypto_ecc_key_gen

```
sid_error_t sid_pal_crypto_ecc_key_gen (sid_pal_ecc_key_gen_params_t * params)
```

Generate ECC key pair using given algorithm.

Parameters

Type	Direction	Argument Name	Description
sid_pal_ecc_key_gen_params_t *	[inout]	params	Generate ECC key pair using given algorithm.

This function generates an ECC key pair using the specified algorithm.

Type definitions

Type definitions

Modules

[sid_pal_hash_params_t](#)

[sid_pal_hmac_params_t](#)

[sid_pal_aes_params_t](#)

[sid_pal_aead_params_t](#)

[sid_pal_dsa_params_t](#)

[sid_pal_ecdh_params_t](#)

[sid_pal_ecc_key_gen_params_t](#)

Enumerations

```
enum    sid\_pal\_hash\_algo\_t {  
        SID_PAL_HASH_SHA256 = 1  
        SID_PAL_HASH_SHA512  
    }  
    Enumeration of hash algorithms supported by SID PAL.
```

```
enum    sid\_pal\_aes\_algo\_t {  
        SID_PAL_AES_CMAC_128 = 1  
        SID_PAL_AES_CTR_128  
    }  
    Enumeration of AES algorithms supported by SID PAL.
```

```
enum    sid\_pal\_aead\_algo\_t {  
        SID_PAL_AEAD_GCM_128 = 1  
        SID_PAL_AEAD_CCM_128  
        SID_PAL_AEAD_CCM_STAR_128  
    }  
    Enumeration of AEAD algorithms supported by SID PAL.
```

```
enum    sid\_pal\_ecc\_algo\_t {  
        SID_PAL_ECDH_CURVE25519 = 1  
        SID_PAL_ECDH_SECP256R1  
        SID_PAL_EDDSA_ED25519  
        SID_PAL_ECDSA_SECP256R1  
    }  
    Enumeration of ECC algorithms supported by SID PAL.
```

```
enum    sid\_pal\_aes\_mode\_t {  
        SID_PAL_CRYPTO_ENCRYPT = 1  
        SID_PAL_CRYPTO_DECRYPT  
        SID_PAL_CRYPTO_MAC_CALCULATE  
    }  
    Enumeration of AES modes supported by SID PAL.
```

```
enum sid_pal_dsa_mode_t {
    SID_PAL_CRYPTTO_SIGN = 1
    SID_PAL_CRYPTTO_VERIFY
}
```

Enumeration of DSA modes supported by SID PAL.

Enumeration Documentation

sid_pal_hash_algo_t

sid_pal_hash_algo_t

Enumeration of hash algorithms supported by SID PAL.

This enumeration defines the hash algorithms that are supported by the SID PAL (Platform Abstraction Layer) for cryptographic operations.

Enumerator	
SID_PAL_HASH_SHA256	SHA-256 hash algorithm
SID_PAL_HASH_SHA512	SHA-512 hash algorithm

sid_pal_aes_algo_t

sid_pal_aes_algo_t

Enumeration of AES algorithms supported by SID PAL.

This enumeration defines the AES algorithms that are supported by the SID PAL (Platform Abstraction Layer) for cryptographic operations.

Enumerator	
SID_PAL_AES_CMAC_128	AES-CMAC-128 algorithm
SID_PAL_AES_CTR_128	AES-CTR-128 algorithm

sid_pal_aead_algo_t

sid_pal_aead_algo_t

Enumeration of AEAD algorithms supported by SID PAL.

This enumeration defines the AEAD algorithms that are supported by the SID PAL (Platform Abstraction Layer) for cryptographic operations.

Enumerator	
SID_PAL_AEAD_GCM_128	AEAD-GCM-128 algorithm
SID_PAL_AEAD_CCM_128	AEAD-CCM-128 algorithm
SID_PAL_AEAD_CCM_STAR_128	AEAD-CCM-STAR-128 algorithm

sid_pal_ecc_algo_t

sid_pal_ecc_algo_t

Enumeration of ECC algorithms supported by SID PAL.

This enumeration defines the ECC algorithms that are supported by the SID PAL (Platform Abstraction Layer) for cryptographic operations.

Enumerator	
SID_PAL_ECDH_CURVE25519	ECDH-Curve25519 algorithm
SID_PAL_ECDH_SECP256R1	ECDH-SECP256R1 algorithm
SID_PAL_EDDSA_ED25519	EdDSA-Ed25519 algorithm
SID_PAL_ECDSA_SECP256R1	ECDSA-SECP256R1 algorithm

sid_pal_aes_mode_t

sid_pal_aes_mode_t

Enumeration of AES modes supported by SID PAL.

This enumeration defines the AES modes that are supported by the SID PAL (Platform Abstraction Layer) for cryptographic operations.

Enumerator	
SID_PAL_CRYPTTO_ENCRYPT	AES encryption mode
SID_PAL_CRYPTTO_DECRYPT	AES decryption mode
SID_PAL_CRYPTTO_MAC_CALCULATE	AES MAC calculation mode

sid_pal_dsa_mode_t

sid_pal_dsa_mode_t

Enumeration of DSA modes supported by SID PAL.

This enumeration defines the DSA modes that are supported by the SID PAL (Platform Abstraction Layer) for cryptographic operations.

Enumerator	
SID_PAL_CRYPTTO_SIGN	DSA signing mode
SID_PAL_CRYPTTO_VERIFY	DSA verification mode

sid_pal_hash_params_t

Parameters for hash operations.

This structure defines the parameters required for hash operations supported by the SID PAL (Platform Abstraction Layer).

Public Attributes

sid_pal_hash_algo_t	algo
uint8_t const *	data
size_t	data_size
uint8_t *	digest
size_t	digest_size

Public Attribute Documentation

algo

sid_pal_hash_algo_t sid_pal_hash_params_t::algo

Hash algorithm to be used

data

uint8_t const* sid_pal_hash_params_t::data

Pointer to the input data

data_size

size_t sid_pal_hash_params_t::data_size

Size of the input data

digest

uint8_t* sid_pal_hash_params_t::digest

Pointer to the output digest

digest_size

size_t sid_pal_hash_params_t::digest_size

Size of the output digest

sid_pal_hmac_params_t

Parameters for HMAC operations.

This structure defines the parameters required for HMAC operations supported by the SID PAL (Platform Abstraction Layer).

Public Attributes

sid_pal_hash_algo_t	algo
uint8_t const *	key
size_t	key_size
uint8_t const *	data
size_t	data_size
uint8_t *	digest
size_t	digest_size

Public Attribute Documentation

algo

```
sid_pal_hash_algo_t sid_pal_hmac_params_t::algo
```

HMAC algorithm to be used

key

```
uint8_t const* sid_pal_hmac_params_t::key
```

Pointer to the key

key_size

```
size_t sid_pal_hmac_params_t::key_size
```

Size of the key

data

```
uint8_t const* sid_pal_hmac_params_t::data
```

Pointer to the input data

data_size

```
size_t sid_pal_hmac_params_t::data_size
```

Size of the input data

digest

```
uint8_t* sid_pal_hmac_params_t::digest
```

Pointer to the output digest

digest_size

```
size_t sid_pal_hmac_params_t::digest_size
```

Size of the output digest

sid_pal_aes_params_t

Parameters for AES cryptographic operations.

This structure defines the parameters required for AES cryptographic operations supported by the SID PAL (Platform Abstraction Layer).

Public Attributes

sid_pal_aes_algo_t	algo
sid_pal_aes_mode_t	mode
uint8_t const *	key
size_t	key_size
uint8_t const *	iv
size_t	iv_size
uint8_t const *	in
size_t	in_size
uint8_t *	out
size_t	out_size

Public Attribute Documentation

algo

```
sid_pal_aes_algo_t sid_pal_aes_params_t::algo
```

AES algorithm to be used

mode

```
sid_pal_aes_mode_t sid_pal_aes_params_t::mode
```

AES mode of operation

key

```
uint8_t const* sid_pal_aes_params_t::key
```

Pointer to the key

key_size

```
size_t sid_pal_aes_params_t::key_size
```

Size of the key

iv

```
uint8_t const* sid_pal_aes_params_t::iv
```

Pointer to the initialization vector

iv_size

```
size_t sid_pal_aes_params_t::iv_size
```

Size of the initialization vector

in

```
uint8_t const* sid_pal_aes_params_t::in
```

Pointer to the input data

in_size

```
size_t sid_pal_aes_params_t::in_size
```

Size of the input data

out

```
uint8_t* sid_pal_aes_params_t::out
```

Pointer to the output data

out_size

```
size_t sid_pal_aes_params_t::out_size
```

Size of the output data

sid_pal_aead_params_t

Parameters for AEAD cryptographic operations.

This structure defines the parameters required for AEAD cryptographic operations supported by the SID PAL (Platform Abstraction Layer).

Public Attributes

sid_pal_aead_algo_t	algo
sid_pal_aes_mode_t	mode
uint8_t const *	key
size_t	key_size
uint8_t const *	iv
size_t	iv_size
uint8_t const *	aad
size_t	aad_size
uint8_t const *	in
size_t	in_size
uint8_t *	out
size_t	out_size
uint8_t *	mac
size_t	mac_size

Public Attribute Documentation

algo

sid_pal_aead_algo_t sid_pal_aead_params_t::algo

AEAD algorithm to be used

mode

sid_pal_aes_mode_t sid_pal_aead_params_t::mode

AEAD mode of operation

```
uint8_t const* sid_pal_aead_params_t::key
```

Pointer to the key

key_size

```
size_t sid_pal_aead_params_t::key_size
```

Size of the key

iv

```
uint8_t const* sid_pal_aead_params_t::iv
```

Pointer to the initialization vector

iv_size

```
size_t sid_pal_aead_params_t::iv_size
```

Size of the initialization vector

aad

```
uint8_t const* sid_pal_aead_params_t::aad
```

Pointer to the additional authenticated data

aad_size

```
size_t sid_pal_aead_params_t::aad_size
```

Size of the additional authenticated data

in

```
uint8_t const* sid_pal_aead_params_t::in
```

Pointer to the input data

in_size

```
size_t sid_pal_aead_params_t::in_size
```

Size of the input data

out

```
uint8_t* sid_pal_aead_params_t::out
```

Pointer to the output data

out_size

```
size_t sid_pal_aead_params_t::out_size
```

Size of the output data

mac

```
uint8_t* sid_pal_aead_params_t::mac
```

Pointer to the message authentication code

mac_size

```
size_t sid_pal_aead_params_t::mac_size
```

Size of the message authentication code

sid_pal_dsa_params_t

Parameters for ECC DSA cryptographic operations.

This structure defines the parameters required for ECC DSA cryptographic operations supported by the SID PAL (Platform Abstraction Layer).

Public Attributes

sid_pal_ecc_algo_t	algo
sid_pal_dsa_mode_t	mode
uint8_t const *	key
size_t	key_size
uint8_t const *	in
size_t	in_size
uint8_t *	signature
size_t	sig_size

Public Attribute Documentation

algo

```
sid_pal_ecc_algo_t sid_pal_dsa_params_t::algo
```

ECC algorithm to be used

mode

```
sid_pal_dsa_mode_t sid_pal_dsa_params_t::mode
```

DSA mode of operation

key

```
uint8_t const* sid_pal_dsa_params_t::key
```

Pointer to the key

key_size

```
size_t sid_pal_dsa_params_t::key_size
```

Size of the key

in

```
uint8_t const* sid_pal_dsa_params_t::in
```

Pointer to the input data

in_size

```
size_t sid_pal_dsa_params_t::in_size
```

Size of the input data

signature

```
uint8_t* sid_pal_dsa_params_t::signature
```

Pointer to the signature

sig_size

```
size_t sid_pal_dsa_params_t::sig_size
```

Size of the signature

sid_pal_ecdh_params_t

Parameters for ECDH cryptographic operations.

This structure defines the parameters required for ECDH cryptographic operations supported by the SID PAL (Platform Abstraction Layer).

Public Attributes

sid_pal_ecc_algo_t	algo
uint8_t const *	prk
size_t	prk_size
uint8_t const *	puk
size_t	puk_size
uint8_t *	shared_secret
size_t	shared_secret_sz

Public Attribute Documentation

algo

```
sid_pal_ecc_algo_t sid_pal_ecdh_params_t::algo
```

ECC algorithm to be used

prk

```
uint8_t const* sid_pal_ecdh_params_t::prk
```

Pointer to the private key

prk_size

```
size_t sid_pal_ecdh_params_t::prk_size
```

Size of the private key

puk

```
uint8_t const* sid_pal_ecdh_params_t::puk
```

Pointer to the public key

puk_size

```
size_t sid_pal_ecdh_params_t::puk_size
```

Size of the public key

shared_secret

```
uint8_t* sid_pal_ecdh_params_t::shared_secret
```

Pointer to the shared secret

shared_secret_sz

```
size_t sid_pal_ecdh_params_t::shared_secret_sz
```

Size of the shared secret

sid_pal_ecc_key_gen_params_t

Parameters for ECC key generation.

This structure defines the parameters required for ECC key generation supported by the SID PAL (Platform Abstraction Layer).

Public Attributes

sid_pal_ecc_algo_t	algo
uint8_t *	prk
size_t	prk_size
uint8_t *	puk
size_t	puk_size

Public Attribute Documentation

algo

sid_pal_ecc_algo_t sid_pal_ecc_key_gen_params_t::algo

ECC algorithm to be used

prk

uint8_t* sid_pal_ecc_key_gen_params_t::prk

Pointer to the private key

prk_size

size_t sid_pal_ecc_key_gen_params_t::prk_size

Size of the private key

puk

uint8_t* sid_pal_ecc_key_gen_params_t::puk

Pointer to the public key

puk_size

size_t sid_pal_ecc_key_gen_params_t::puk_size

Size of the public key

Storage Interface

Storage Interface

Storage Interface

The Storage Interface module provides interfaces for managing persistent storage within the Sidewalk SDK. This module includes interfaces for key-value storage and manufacturing page storage. Developers can ensure consistent and platform-independent access to storage functionalities, enabling their applications to store and retrieve data reliably across different hardware platforms.

Modules

[KV Storage](#)

[Manufacturing Page](#)

KV Storage

KV Storage

Provides persistent storage interface for key mapped values.

Functions

sid_error_t	sid_pal_storage_kv_init(void) Initialize the key value storage subsystem.
sid_error_t	sid_pal_storage_kv_deinit(void) Deinitialize the key value storage subsystem.
sid_error_t	sid_pal_storage_kv_record_get(uint16_t group, uint16_t key, void *p_data, uint32_t len) Get a value using its group and key IDs.
sid_error_t	sid_pal_storage_kv_record_get_len(uint16_t group, uint16_t key, uint32_t *p_len) Get the size of a value using its group and key IDs.
sid_error_t	sid_pal_storage_kv_record_set(uint16_t group, uint16_t key, void const *p_data, uint32_t len) Set a value using its group and key IDs.
sid_error_t	sid_pal_storage_kv_record_delete(uint16_t group, uint16_t key) Delete a value using its group and key IDs.
sid_error_t	sid_pal_storage_kv_group_delete(uint16_t group) Delete all values in a group.

Macros

#define	SID_PAL_KV_STORE_MAX_LENGTH_BYTES 48 Maximum length in bytes for the Key-Value store.
#define	SID_PAL_STORAGE_KV_INTERNAL_PROTOCOL_GROUP_ID 0x2000 Group ID for the internal protocol of the Key-Value storage.
#define	SID_PAL_STORAGE_KV_INTERNAL_CONFIG_GROUP_ID 0x3456 Group ID for the internal configuration of the Key-Value storage.
#define	SID_PAL_STORAGE_KV_INTERNAL_BULK_DATA_TRANSFER_GROUP_ID 0x4567 Group ID for the internal bulk data transfer of the Key-Value storage.

Function Documentation

sid_pal_storage_kv_init

sid_error_t sid_pal_storage_kv_init (void)

Initialize the key value storage subsystem.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_storage_kv_deinit

```
sid_error_t sid_pal_storage_kv_deinit (void )
```

Deinitialize the key value storage subsystem.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_storage_kv_record_get

```
sid_error_t sid_pal_storage_kv_record_get (uint16_t group, uint16_t key, void * p_data, uint32_t len)
```

Get a value using its group and key IDs.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	group	Group
uint16_t	[in]	key	Key
void *	[out]	p_data	Pointer to output buffer to contain the value
uint32_t	[in]	len	Maximum length of buffer pointed to by p_data in bytes

sid_pal_storage_kv_record_get_len

```
sid_error_t sid_pal_storage_kv_record_get_len (uint16_t group, uint16_t key, uint32_t * p_len)
```

Get the size of a value using its group and key IDs.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	group	Group
uint16_t	[in]	key	Key
uint32_t *	[out]	p_len	Pointer to integer to contain the size of the value in bytes

sid_pal_storage_kv_record_set

```
sid_error_t sid_pal_storage_kv_record_set (uint16_t group, uint16_t key, void const * p_data, uint32_t len)
```

Set a value using its group and key IDs.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	group	Group
uint16_t	[in]	key	Key
void const *	[in]	p_data	Pointer to input buffer which contains the value
uint32_t	[in]	len	The size of the input value in bytes

sid_pal_storage_kv_record_delete

```
sid_error_t sid_pal_storage_kv_record_delete (uint16_t group, uint16_t key)
```

Delete a value using its group and key IDs.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	group	Group
uint16_t	[in]	key	Key

sid_pal_storage_kv_group_delete

```
sid_error_t sid_pal_storage_kv_group_delete (uint16_t group)
```

Delete all values in a group.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	group	Group

Manufacturing Page

Manufacturing Page

Provides manufacturing store interface and implementation to store the manufacturing page into the EFR32 flash.

Modules

[Type definitions](#)

Functions

void	sid_pal_mfg_store_init (sid_pal_mfg_store_region_t mfg_store_region) Prepare the manufacturing store for use.
void	sid_pal_mfg_store_deinit (void) Deinitialize previously initialized mfg region.
int32_t	sid_pal_mfg_store_erase (void) Erase the manufacturing store.
bool	sid_pal_mfg_store_is_empty (void) Check if the manufacturing store is empty.
int32_t	sid_pal_mfg_store_write (uint16_t value, const uint8_t *buffer, uint16_t length) Write to mfg store.
void	sid_pal_mfg_store_read (uint16_t value, uint8_t *buffer, uint16_t length) Read from mfg store.
uint16_t	sid_pal_mfg_store_get_length_for_value (uint16_t value) Get length of a tag ID.
bool	sid_pal_mfg_store_is_tlv_support (void) Check if the manufacturing store supports TLV based storage.
uint32_t	sid_pal_mfg_store_get_version (void) Get version of values stored in mfg store.
bool	sid_pal_mfg_store_dev_id_get (uint8_t dev_id[SID_PAL_MFG_STORE_DEVID_SIZE]) Get the device ID from the mfg store.
bool	sid_pal_mfg_store_serial_num_get (uint8_t serial_num[SID_PAL_MFG_STORE_SERIAL_NUM_SIZE]) Get the device serial number from the mfg store.
void	sid_pal_mfg_store_apid_get (uint8_t apid[SID_PAL_MFG_STORE_APID_SIZE]) Get the APID.
void	sid_pal_mfg_store_app_pub_key_get (uint8_t app_pub[SID_PAL_MFG_STORE_APP_PUB_ED25519_SIZE]) Get the Application public key.

Function Documentation

sid_pal_mfg_store_init

```
void sid_pal_mfg_store_init (sid_pal_mfg_store_region_t mfg_store_region)
```

Prepare the manufacturing store for use.

Parameters

Type	Direction	Argument Name	Description
sid_pal_mfg_store_region_t	[in]	mfg_store_region	Structure containing start and end addresses of the manufacturing store.

Must be called before any of the other sid_pal_mfg_store functions.

sid_pal_mfg_store_deinit

```
void sid_pal_mfg_store_deinit (void )
```

Deinitialize previously initialized mfg region.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sid_pal_mfg_store_erase

```
int32_t sid_pal_mfg_store_erase (void )
```

Erase the manufacturing store.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Because the manufacturing store is backed by flash memory, and flash memory can only be erased in large chunks (pages), this interface only supports erasing the entire manufacturing store.

Note

- This function is only supported for diagnostic builds.

Returns

- 0 on success, negative value on failure.

sid_pal_mfg_store_is_empty

```
bool sid_pal_mfg_store_is_empty (void )
```

Check if the manufacturing store is empty.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function is only supported for diagnostic builds.

sid_pal_mfg_store_write

```
int32_t sid_pal_mfg_store_write (uint16_t value, const uint8_t * buffer, uint16_t length)
```

Write to mfg store.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	value	Enum constant for the desired value. Use values from sid_pal_mfg_store_value_t or application defined values here.
const uint8_t *	[in]	buffer	Buffer containing the value to be stored.
uint16_t	[in]	length	Length of the value in bytes. Use values from sid_pal_mfg_store_value_size_t here.

sid_pal_mfg_store_read

```
void sid_pal_mfg_store_read (uint16_t value, uint8_t * buffer, uint16_t length)
```

Read from mfg store.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	value	Enum constant for the desired value. Use values from sid_pal_mfg_store_value_t or application defined values here.
uint8_t *	[out]	buffer	Buffer to which the value will be copied.
uint16_t	[in]	length	Length of the value in bytes. Use values from sid_pal_mfg_store_value_size_t here.

sid_pal_mfg_store_get_length_for_value

```
uint16_t sid_pal_mfg_store_get_length_for_value (uint16_t value)
```

Get length of a tag ID.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	value	Enum constant for the desired value. Use values from sid_pal_mfg_store_value_t or application defined values here.

sid_pal_mfg_store_is_tlv_support

```
bool sid_pal_mfg_store_is_tlv_support (void )
```

Check if the manufacturing store supports TLV based storage.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function only indicates that the platform supports TLV, but the device may have storage with fixed offsets that was flashed during production.

sid_pal_mfg_store_get_version

```
uint32_t sid_pal_mfg_store_get_version (void )
```

Get version of values stored in mfg store.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- Functions specific to Sidewalk with special handling The version of the mfg values is stored along with all the values in mfg store. This API retrieves the value by reading the address at which the version is stored.

sid_pal_mfg_store_dev_id_get

```
bool sid_pal_mfg_store_dev_id_get (uint8_t dev_id)
```

Get the device ID from the mfg store.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[out]	dev_id	The device ID

sid_pal_mfg_store_serial_num_get

```
bool sid_pal_mfg_store_serial_num_get (uint8_t serial_num)
```

Get the device serial number from the mfg store.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[out]	serial_num	The device serial number

sid_pal_mfg_store_apid_get

```
void sid_pal_mfg_store_apid_get (uint8_t apid)
```

Get the APID.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[out]	apid	The apid

Applicable only for products with short form certificate chain.

sid_pal_mfg_store_app_pub_key_get

```
void sid_pal_mfg_store_app_pub_key_get (uint8_t app_pub)
```

Get the Application public key.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[out]	app_pub	The Application public key

Applicable only for products with short form certificate chain.

Type definitions

Type definitions

Modules

[sid_pal_mfg_store_region_t](#)

Enumerations

```
enum @0 {
    SID_PAL_MFG_STORE_INVALID_OFFSET = UINT32_MAX
}

enum sid\_pal\_mfg\_store\_value\_t {
    SID_PAL_MFG_STORE_DEVID = 1
    SID_PAL_MFG_STORE_VERSION = 2
    SID_PAL_MFG_STORE_SERIAL_NUM = 3
    SID_PAL_MFG_STORE_SMSN = 4
    SID_PAL_MFG_STORE_APP_PUB_ED25519 = 5
    SID_PAL_MFG_STORE_DEVICE_PRIV_ED25519 = 6
    SID_PAL_MFG_STORE_DEVICE_PUB_ED25519 = 7
    SID_PAL_MFG_STORE_DEVICE_PUB_ED25519_SIGNATURE = 8
    SID_PAL_MFG_STORE_DEVICE_PRIV_P256R1 = 9
    SID_PAL_MFG_STORE_DEVICE_PUB_P256R1 = 10
    SID_PAL_MFG_STORE_DEVICE_PUB_P256R1_SIGNATURE = 11
    SID_PAL_MFG_STORE_DAK_PUB_ED25519 = 12
    SID_PAL_MFG_STORE_DAK_PUB_ED25519_SIGNATURE = 13
    SID_PAL_MFG_STORE_DAK_ED25519_SERIAL = 14
    SID_PAL_MFG_STORE_DAK_PUB_P256R1 = 15
    SID_PAL_MFG_STORE_DAK_PUB_P256R1_SIGNATURE = 16
    SID_PAL_MFG_STORE_DAK_P256R1_SERIAL = 17
    SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519 = 18
    SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519_SIGNATURE = 19
    SID_PAL_MFG_STORE_PRODUCT_ED25519_SERIAL = 20
    SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1 = 21
    SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1_SIGNATURE = 22
    SID_PAL_MFG_STORE_PRODUCT_P256R1_SERIAL = 23
    SID_PAL_MFG_STORE_MAN_PUB_ED25519 = 24
    SID_PAL_MFG_STORE_MAN_PUB_ED25519_SIGNATURE = 25
    SID_PAL_MFG_STORE_MAN_ED25519_SERIAL = 26
    SID_PAL_MFG_STORE_MAN_PUB_P256R1 = 27
    SID_PAL_MFG_STORE_MAN_PUB_P256R1_SIGNATURE = 28
    SID_PAL_MFG_STORE_MAN_P256R1_SERIAL = 29
    SID_PAL_MFG_STORE_SW_PUB_ED25519 = 30
    SID_PAL_MFG_STORE_SW_PUB_ED25519_SIGNATURE = 31
    SID_PAL_MFG_STORE_SW_ED25519_SERIAL = 32
    SID_PAL_MFG_STORE_SW_PUB_P256R1 = 33
    SID_PAL_MFG_STORE_SW_PUB_P256R1_SIGNATURE = 34
    SID_PAL_MFG_STORE_SW_P256R1_SERIAL = 35
    SID_PAL_MFG_STORE_AMZN_PUB_ED25519 = 36
    SID_PAL_MFG_STORE_AMZN_PUB_P256R1 = 37
    SID_PAL_MFG_STORE_APIID = 38
    SID_PAL_MFG_STORE_CORE_VALUE_MAX = 4000
    SID_PAL_MFG_STORE_VALUE_MAX = 0x6FFE
}
```

Values available to all users of the manufacturing store.

```
enum sid_pal_mfg_store_value_size_t {
    SID_PAL_MFG_STORE_DEVID_SIZE = 5
    SID_PAL_MFG_STORE_VERSION_SIZE = 4
    SID_PAL_MFG_STORE_SERIAL_NUM_SIZE = 17
    SID_PAL_MFG_STORE_SMSN_SIZE = 32
    SID_PAL_MFG_STORE_APP_PUB_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_DEVICE_PRIV_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_DEVICE_PUB_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_DEVICE_PUB_ED25519_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_DEVICE_PRIV_P256R1_SIZE = 32
    SID_PAL_MFG_STORE_DEVICE_PUB_P256R1_SIZE = 64
    SID_PAL_MFG_STORE_DEVICE_PUB_P256R1_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_DAK_PUB_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_DAK_PUB_ED25519_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_DAK_ED25519_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_DAK_PUB_P256R1_SIZE = 64
    SID_PAL_MFG_STORE_DAK_PUB_P256R1_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_DAK_P256R1_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_PRODUCT_ED25519_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1_SIZE = 64
    SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_PRODUCT_P256R1_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_MAN_PUB_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_MAN_PUB_ED25519_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_MAN_ED25519_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_MAN_PUB_P256R1_SIZE = 64
    SID_PAL_MFG_STORE_MAN_PUB_P256R1_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_MAN_P256R1_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_SW_PUB_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_SW_PUB_ED25519_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_SW_ED25519_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_SW_PUB_P256R1_SIZE = 64
    SID_PAL_MFG_STORE_SW_PUB_P256R1_SIGNATURE_SIZE = 64
    SID_PAL_MFG_STORE_SW_P256R1_SERIAL_SIZE = 4
    SID_PAL_MFG_STORE_AMZN_PUB_ED25519_SIZE = 32
    SID_PAL_MFG_STORE_AMZN_PUB_P256R1_SIZE = 64
    SID_PAL_MFG_STORE_APID_SIZE = 4
}
```

Value sizes in bytes.

Typedefs

```
typedef sid_pal_mfg_store_app_value_to_offset_t(int value)
uint32_t(*)
Function pointer type for converting an application-specific value to an offset.
```

Macros

```
#define SID_PAL_MFG_STORE_EMPTY_VERSION_NUMBER 0xFFFFFFFF
The current version of the MFG storage.

#define SID_PAL_MFG_STORE_FIXED_OFFSETS_VERSION 7

#define SID_PAL_MFG_STORE_TLV_VERSION 8

#define SID_PAL_MFG_STORE_MAX_FLASH_WRITE_LEN 64
Maximum length for flash write operations in the manufacturing store.
```

Enumeration Documentation

@0

@0

Enumerator

SID_PAL_MFG_STORE_INVALID_OFFSET

sid_pal_mfg_store_value_t

sid_pal_mfg_store_value_t

Values available to all users of the manufacturing store.

Enumerator

SID_PAL_MFG_STORE_DEVID	use sid_pal_mfg_store_dev_id_get
SID_PAL_MFG_STORE_VERSION	<div>Note<ul style="list-style-type: none">Version is stored in network order use sid_pal_mfg_store_get_version</div>
SID_PAL_MFG_STORE_SERIAL_NUM	use sid_pal_mfg_store_dev_id_get
SID_PAL_MFG_STORE_SMSN	
SID_PAL_MFG_STORE_APP_PUB_ED25519	
SID_PAL_MFG_STORE_DEVICE_PRIV_ED25519	
SID_PAL_MFG_STORE_DEVICE_PUB_ED25519	
SID_PAL_MFG_STORE_DEVICE_PUB_ED25519_SIGNATURE	
SID_PAL_MFG_STORE_DEVICE_PRIV_P256R1	
SID_PAL_MFG_STORE_DEVICE_PUB_P256R1	
SID_PAL_MFG_STORE_DEVICE_PUB_P256R1_SIGNATURE	
SID_PAL_MFG_STORE_DAK_PUB_ED25519	
SID_PAL_MFG_STORE_DAK_PUB_ED25519_SIGNATURE	
SID_PAL_MFG_STORE_DAK_ED25519_SERIAL	
SID_PAL_MFG_STORE_DAK_PUB_P256R1	
SID_PAL_MFG_STORE_DAK_PUB_P256R1_SIGNATURE	
SID_PAL_MFG_STORE_DAK_P256R1_SERIAL	
SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519	
SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519_SIGNATURE	
SID_PAL_MFG_STORE_PRODUCT_ED25519_SERIAL	
SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1	
SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1_SIGNATURE	
SID_PAL_MFG_STORE_PRODUCT_P256R1_SERIAL	
SID_PAL_MFG_STORE_MAN_PUB_ED25519	
SID_PAL_MFG_STORE_MAN_PUB_ED25519_SIGNATURE	
SID_PAL_MFG_STORE_MAN_ED25519_SERIAL	
SID_PAL_MFG_STORE_MAN_PUB_P256R1	
SID_PAL_MFG_STORE_MAN_PUB_P256R1_SIGNATURE	
SID_PAL_MFG_STORE_MAN_P256R1_SERIAL	
SID_PAL_MFG_STORE_SW_PUB_ED25519	
SID_PAL_MFG_STORE_SW_PUB_ED25519_SIGNATURE	

SID_PAL_MFG_STORE_SW_ED25519_SERIAL	
SID_PAL_MFG_STORE_SW_PUB_P256R1	
SID_PAL_MFG_STORE_SW_PUB_P256R1_SIGNATURE	
SID_PAL_MFG_STORE_SW_P256R1_SERIAL	
SID_PAL_MFG_STORE_AMZN_PUB_ED25519	
SID_PAL_MFG_STORE_AMZN_PUB_P256R1	
SID_PAL_MFG_STORE_APID	
SID_PAL_MFG_STORE_CORE_VALUE_MAX	Note <ul style="list-style-type: none">This arbitrary value is the number of value identifiers reserved by Sidewalk. The range of these value identifiers is: [0, SID_PAL_MFG_STORE_CORE_VALUE_MAX]. Applications may use identifiers outside of that range.
SID_PAL_MFG_STORE_VALUE_MAX	Note <ul style="list-style-type: none">The value 0x6FFF is reserved for internal use

sid_pal_mfg_store_value_size_t

sid_pal_mfg_store_value_size_t

Value sizes in bytes.

This enum defines the sizes of various values stored in the manufacturing store.

Enumerator	
SID_PAL_MFG_STORE_DEVID_SIZE	
SID_PAL_MFG_STORE_VERSION_SIZE	
SID_PAL_MFG_STORE_SERIAL_NUM_SIZE	
SID_PAL_MFG_STORE_SMSN_SIZE	
SID_PAL_MFG_STORE_APP_PUB_ED25519_SIZE	
SID_PAL_MFG_STORE_DEVICE_PRIV_ED25519_SIZE	
SID_PAL_MFG_STORE_DEVICE_PUB_ED25519_SIZE	
SID_PAL_MFG_STORE_DEVICE_PUB_ED25519_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_DEVICE_PRIV_P256R1_SIZE	
SID_PAL_MFG_STORE_DEVICE_PUB_P256R1_SIZE	
SID_PAL_MFG_STORE_DEVICE_PUB_P256R1_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_DAK_PUB_ED25519_SIZE	
SID_PAL_MFG_STORE_DAK_PUB_ED25519_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_DAK_ED25519_SERIAL_SIZE	
SID_PAL_MFG_STORE_DAK_PUB_P256R1_SIZE	
SID_PAL_MFG_STORE_DAK_PUB_P256R1_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_DAK_P256R1_SERIAL_SIZE	
SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519_SIZE	
SID_PAL_MFG_STORE_PRODUCT_PUB_ED25519_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_PRODUCT_ED25519_SERIAL_SIZE	
SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1_SIZE	
SID_PAL_MFG_STORE_PRODUCT_PUB_P256R1_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_PRODUCT_P256R1_SERIAL_SIZE	
SID_PAL_MFG_STORE_MAN_PUB_ED25519_SIZE	

SID_PAL_MFG_STORE_MAN_PUB_ED25519_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_MAN_ED25519_SERIAL_SIZE	
SID_PAL_MFG_STORE_MAN_PUB_P256R1_SIZE	
SID_PAL_MFG_STORE_MAN_PUB_P256R1_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_MAN_P256R1_SERIAL_SIZE	
SID_PAL_MFG_STORE_SW_PUB_ED25519_SIZE	
SID_PAL_MFG_STORE_SW_PUB_ED25519_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_SW_ED25519_SERIAL_SIZE	
SID_PAL_MFG_STORE_SW_PUB_P256R1_SIZE	
SID_PAL_MFG_STORE_SW_PUB_P256R1_SIGNATURE_SIZE	
SID_PAL_MFG_STORE_SW_P256R1_SERIAL_SIZE	
SID_PAL_MFG_STORE_AMZN_PUB_ED25519_SIZE	
SID_PAL_MFG_STORE_AMZN_PUB_P256R1_SIZE	
SID_PAL_MFG_STORE_APID_SIZE	

Typedef Documentation

sid_pal_mfg_store_app_value_to_offset_t

```
sid_pal_mfg_store_app_value_to_offset_t )(int value)
```

Function pointer type for converting an application-specific value to an offset.

Parameters

Type	Direction	Argument Name	Description
	N/A	value	The application-specific value to be converted to an offset.

Returns

- A 32-bit unsigned integer representing the offset corresponding to the input value.

sid_pal_mfg_store_region_t

Type which holds the start and end addresses of the manufacturing store.

Public Attributes

uintptr_t	addr_start
uintptr_t	addr_end
sid_pal_mfg_store_app_value_to_offset_t	app_value_to_offset This function allows applications to extend the manufacturing store to be be used for non-Sidewalk values.

Public Attribute Documentation

addr_start

```
uintptr_t sid_pal_mfg_store_region_t::addr_start
```

The start address of the manufacturing store region.

addr_end

```
uintptr_t sid_pal_mfg_store_region_t::addr_end
```

The end address of the manufacturing store region.

app_value_to_offset

```
sid_pal_mfg_store_app_value_to_offset_t sid_pal_mfg_store_region_t::app_value_to_offset
```

This function allows applications to extend the manufacturing store to be be used for non-Sidewalk values.

Applications should provide an implementation of this function if they wish to extend the manufacturing store for their own use. Its responsibility is to convert a value identifier to an offset (in bytes) from the beginning of the manufacturing store. Sidewalk owns the first SID_PAL_MFG_STORE_CORE_VALUE_MAX identifiers, So this function's input should be greater than SID_PAL_MFG_STORE_CORE_VALUE_MAX and its output should a valid offset below the mfg_store end address. If no mapping from the value identifier to an offset can be found, this function should return SID_PAL_MFG_STORE_INVALID_OFFSET value, which will cause the manufacturing store implementation to reject any operation on the provided value.

Timer Interfaces

Timer Interfaces

Timer Interfaces

The Timer Interfaces module provides interfaces for managing timer functionalities within the Sidewalk SDK. These interfaces ensure consistent and platform-independent access to timer operations, such as setting, starting, stopping, and resetting timers. Developers can create applications that are portable across different hardware platforms while maintaining consistent timing behavior and performance.

Modules

[Delay](#)

[Timer](#)

[Uptime](#)

Delay

Delay

Provides a way for Sub-GHz protocol to control delay.

Functions

- void

[sid_pal_delay_us](#)(uint32_t delay)

Implements a busy wait delay safe to be used in SWI context.
- void

[sid_pal_scheduler_delay_ms](#)(uint32_t delay)

Implements a delay function using RTOS API call.

Function Documentation

sid_pal_delay_us

```
void sid_pal_delay_us (uint32_t delay)
```

Implements a busy wait delay safe to be used in SWI context.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	delay	Time in us to delay

Due to busy wait implementation, should not be used for long durations. If you need delay >1ms then use sid_pal_scheduler_delay_ms instead.

sid_pal_scheduler_delay_ms

```
void sid_pal_scheduler_delay_ms (uint32_t delay)
```

Implements a delay function using RTOS API call.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	delay	Time in ms to delay

This function will block the calling thread, do not use in ISR context. This will allow the RTOS scheduler to run other tasks or switch to an IDLE state while the delay is pending.

If you need sub-millisecond delays then use sid_pal_delay_us instead.

Do not use this function for internal Sidewalk stack delays - use the sid_pal_timer API instead.

Timer

Timer

Interface for timers for sidewalk SDK.

Modules

Type definitions

Functions

- sid_error_t

sid_pal_timer_init

(sid_pal_timer_t *timer, sid_pal_timer_cb_t event_callback, void *event_callback_arg)

Initialize a timer object.
- sid_error_t

sid_pal_timer_deinit

(sid_pal_timer_t *timer)

De-initialize a timer object.
- sid_error_t

sid_pal_timer_arm

(sid_pal_timer_t *timer, sid_pal_timer_prio_class_t type, const struct sid_timespec *when, const struct sid_timespec *period)

Arm a timer object.
- sid_error_t

sid_pal_timer_cancel

(sid_pal_timer_t *timer)

Disarm a timer object.
- bool

sid_pal_timer_is_armed

(const sid_pal_timer_t *timer)

Check a timer object is valid and armed.
- sid_error_t

sid_pal_timer_facility_init

(void *arg)

Init the timer facility.
- void

sid_pal_timer_event_callback

(void *arg, const struct sid_timespec *now)

HW event callback.

Function Documentation

sid_pal_timer_init

```
sid_error_t sid_pal_timer_init (sid_pal_timer_t * timer, sid_pal_timer_cb_t event_callback, void * event_callback_arg)
```

Initialize a timer object.

Parameters

Type	Direction	Argument Name	Description
sid_pal_timer_t *	[in]	timer	Timer object to initialize
sid_pal_timer_cb_t	[in]	event_callback	Pointer to the callback function the timer event will be delivered to
void *	[in]	event_callback_arg	Argument to be provided to the event_callback during call

sid_pal_timer_deinit

```
sid_error_t sid_pal_timer_deinit (sid_pal_timer_t * timer)
```

De-initialize a timer object.

Parameters

Type	Direction	Argument Name	Description
sid_pal_timer_t *	[in]	timer	Timer object to de-initialize

Function fully de-initializes the `timer` object. If it is armed, it will be canceled and then de-initialized.

sid_pal_timer_arm

```
sid_error_t sid_pal_timer_arm (sid_pal_timer_t * timer, sid_pal_timer_prio_class_t type, const struct sid_timespec * when, const struct sid_timespec * period)
```

Arm a timer object.

Parameters

Type	Direction	Argument Name	Description
sid_pal_timer_t *	[in]	timer	Timer object to arm
sid_pal_timer_prio_class_t	[in]	type	Priority class specifier for the timer to be armed
const struct sid_timespec *	[in]	when	Pointer to struct sid_timespec identifying the time for the first event generation
const struct sid_timespec *	[in]	period	Pointer to struct sid_timespec identifying the period between event generation

Function will initialize the `timer` object for first shot at time provided in `when` (required). If the `period` is not NULL and is not TIMESPEC_INFINITY, the `timer` object will be armed to repeat events generation periodically with the period according to the time provided in `period`.

sid_pal_timer_cancel

```
sid_error_t sid_pal_timer_cancel (sid_pal_timer_t * timer)
```

Disarm a timer object.

Parameters

Type	Direction	Argument Name	Description
sid_pal_timer_t *	[in]	timer	Timer object to disarm

Function will disarm the `timer` object. If it is not armed, function does no operation.

sid_pal_timer_is_armed

```
bool sid_pal_timer_is_armed (const sid_pal_timer_t * timer)
```

Check a timer object is valid and armed.

Parameters

Type	Direction	Argument Name	Description
const sid_pal_timer_t *	[in]	timer	Timer object to check

sid_pal_timer_facility_init

```
sid_error_t sid_pal_timer_facility_init (void * arg)
```

Init the timer facility.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	arg	Pointer to implementation-specific arguments, can be NULL if not used.

This function must be called before before [sid_pal_timer_init\(\)](#).

OPTIONAL This function is typically used to init HW or SW resources needed for the timer. If none are needed by the timer implementation then this function is unnecessary.

sid_pal_timer_event_callback

```
void sid_pal_timer_event_callback (void * arg, const struct sid_timespec * now)
```

HW event callback.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	arg	Pointer to implementation-specific arguments, can be NULL if not used.
const struct sid_timespec *	[in]	now	Pointer to the current time when the event occurred.

OPTIONAL If sid_timer is implemented as a SW timer, this is the callback that can be registered with the HW resource to provide noritification of HW timer expiry.

Type definitions

Type definitions

Enumerations

```
enum sid_pal_timer_prio_class_t {
    SID_PAL_TIMER_PRIO_CLASS_PRECISE
    SID_PAL_TIMER_PRIO_CLASS_LOWPPOWER
}
```

Timer priority class enumeration.

Enumeration Documentation

sid_pal_timer_prio_class_t

sid_pal_timer_prio_class_t

Timer priority class enumeration.

This enum defines the priority classes for timers, which determine the precision and power consumption characteristics of timer events.

Enumerator	
SID_PAL_TIMER_PRIO_CLASS_PRECISE	Events to be generated with the maximum supported on this platform precision
SID_PAL_TIMER_PRIO_CLASS_LOWPPOWER	Events can be delayed for up to 1 second to optimize power consumption

Uptime

Uptime

The Uptime Interface module provides interfaces for tracking and managing system uptime within the Sidewalk SDK. These interfaces ensure consistent and platform-independent access to uptime information, allowing developers to monitor the duration for which the system has been running since the last reset or power cycle.

Functions

- sid_error_t

sid_pal_uptime_now

(struct sid_timespec *time)

Get the current time of specified clock source.
- void

sid_pal_uptime_set_xtal_ppm

(int16_t ppm)

Set crystal offset for RTC compensation.
- int16_t

sid_pal_uptime_get_xtal_ppm

(void)

Get current crystal offset.

Function Documentation

sid_pal_uptime_now

sid_error_t sid_pal_uptime_now (struct sid_timespec * time)

Get the current time of specified clock source.

Parameters

Type	Direction	Argument Name	Description
struct sid_timespec *	[out]	time	current time

NOTE: drift may be NULL. In this case time should be set and drift ignored. Success should be returned.

sid_pal_uptime_set_xtal_ppm

void sid_pal_uptime_set_xtal_ppm (int16_t ppm)

Set crystal offset for RTC compensation.

Parameters

Type	Direction	Argument Name	Description
int16_t	[in]	ppm	offset in PPM

sid_pal_uptime_get_xtal_ppm

int16_t sid_pal_uptime_get_xtal_ppm (void)

Get current crystal offset.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Sidewalk PAL

Sidewalk PAL

PAL

The Sidewalk Platform Abstraction Layer (PAL) provides a set of standardized interfaces that abstract the underlying hardware and platform-specific details. This layer ensures that the Sidewalk SDK can operate seamlessly across different hardware platforms by providing consistent APIs for common functionalities such as GPIO, timers, logging, and storage. Developers can create portable applications that maintain consistent behavior and performance across various hardware configurations.

Modules

[BLE Adaptation](#)

[GPIO](#)

[NVM3 Manager](#)

[Timer Types](#)

BLE Adaptation

BLE Adaptation

BLE Adaptation

The BLE adapter module provides the necessary interfaces and functionalities to integrate Bluetooth Low Energy (BLE) capabilities into applications using the Sidewalk SDK. This module includes support for BLE advertising, connection management, and data transmission. It also defines the configuration structures and callback mechanisms required to handle BLE events and operations. By utilizing the BLE adapter, developers can enable their devices to communicate over BLE within the Amazon Sidewalk network.

Key Features:

- **BLE Advertising:** Supports BLE advertising to broadcast data to nearby devices.
- **Connection Management:** Manages BLE connections, including establishing and terminating connections.
- **Data Transmission:** Facilitates data transmission over BLE, including sending and receiving data.
- **Event Handling:** Provides callback mechanisms to handle various BLE events such as connection, disconnection, and data reception.
- **Configuration Structures:** Defines configuration structures for setting up BLE parameters.
- **Security:** Supports secure BLE communication through encryption and authentication mechanisms.

Modules

[BLE adapter](#)

BLE adapter

BLE adapter

Modules

Type definitions

Functions

- sid_error_t

[sid_pal_ble_adapter_create](#)(sid_pal_ble_adapter_interface_t *handle)
Creates a BLE adapter interface.
- void

[sl_ble_adapter_on_event](#)(sl_bt_msg_t *evt)
Handles BLE events.
- void

[sl_ble_adapter_on_kernel_start](#)(void)
Callback for kernel start event.
- sid_error_t

[ble_adapter_get_advertiser_address](#)(uint8_t *addr)
This function returns the MAC address of the BLE advertiser.
- sid_error_t

[ble_adapter_get_connection_address](#)(uint8_t *addr)
This function returns the MAC address of the BLE connection.

Function Documentation

sid_pal_ble_adapter_create

```
sid_error_t sid_pal_ble_adapter_create (sid_pal_ble_adapter_interface_t * handle)
```

Creates a BLE adapter interface.

Parameters

Type	Direction	Argument Name	Description
sid_pal_ble_adapter_interface_t *	[inout]	handle	Pointer to the BLE adapter interface handle.

This function initializes the BLE adapter interface and assigns it to the provided handle.

Returns

- sid_error_t Error code indicating the result of the operation.

sl_ble_adapter_on_event

```
void sl_ble_adapter_on_event (sl_bt_msg_t * evt)
```

Handles BLE events.

Parameters

Type	Direction	Argument Name	Description
sl_bt_msg_t *	[in]	evt	Pointer to the Bluetooth event message.

This function processes BLE events received from the Bluetooth stack.

sl_ble_adapter_on_kernel_start

```
void sl_ble_adapter_on_kernel_start (void )
```

Callback for kernel start event.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is called when the kernel starts.

ble_adapter_get_advertiser_address

```
sid_error_t ble_adapter_get_advertiser_address (uint8_t * addr)
```

This function returns the MAC address of the BLE advertiser.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	addr	Pointer to the buffer where the MAC address will be stored.

This MAC address is used during BLE advertising.

Returns

- sid_error_t Error code indicating the result of the operation.

ble_adapter_get_connection_address

```
sid_error_t ble_adapter_get_connection_address (uint8_t * addr)
```

This function returns the MAC address of the BLE connection.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	addr	Pointer to the buffer where the MAC address will be stored.

This MAC address is used during BLE connection.

Returns

- sid_error_t Error code indicating the result of the operation.

Type definitions

Type definitions

Modules

[sid_pal_ble_profile_config_t](#)

[sid_pal_ble_adapter_ctx_t](#)

sid_pal_ble_profile_config_t

Configuration structure for BLE profile in SID PAL.

This structure holds the handles for the current BLE service, characteristic, and descriptor.

Public Attributes

- uint16_t [current_service_handle](#)
- uint16_t * [current_characteristic_handle](#)
- uint16_t * [current_descriptor_handle](#)

Public Attribute Documentation

current_service_handle

```
uint16_t sid_pal_ble_profile_config_t::current_service_handle
```

The service declaration attribute handle

current_characteristic_handle

```
uint16_t* sid_pal_ble_profile_config_t::current_characteristic_handle
```

The characteristic value attribute handle

current_descriptor_handle

```
uint16_t* sid_pal_ble_profile_config_t::current_descriptor_handle
```

The descriptor attribute handle

sid_pal_ble_adapter_ctx_t

BLE Adapter Context Structure.

This structure holds the context information for the BLE adapter.

Public Attributes

const sid_ble_config_t *	cfg
const sid_pal_ble_adapter_callbacks_t *	callback
uint16_t	mtu_size
bool	is_connected
uint16_t	conn_id
uint8_t	bt_addr
sid_ble_cfg_adv_param_t	current_adv_config
sid_ble_cfg_conn_param_t	current_conn_config
sid_ble_cfg_conn_param_t	last_conn_config

Public Attribute Documentation

cfg

```
const sid_ble_config_t* sid_pal_ble_adapter_ctx_t::cfg
```

Configuration parameters for the BLE adapter.

callback

```
const sid_pal_ble_adapter_callbacks_t* sid_pal_ble_adapter_ctx_t::callback
```

Callback functions for the BLE adapter.

mtu_size

```
uint16_t sid_pal_ble_adapter_ctx_t::mtu_size
```

Maximum Transmission Unit (MTU) size.

is_connected

```
bool sid_pal_ble_adapter_ctx_t::is_connected
```

Connection status flag.

conn_id

```
uint16_t sid_pal_ble_adapter_ctx_t::conn_id
```

Connection identifier.

bt_addr

```
uint8_t sid_pal_ble_adapter_ctx_t::bt_addr[BLE_ADDR_MAX_LEN]
```

Bluetooth address.

current_adv_config

```
sid_ble_cfg_adv_param_t sid_pal_ble_adapter_ctx_t::current_adv_config
```

Current advertising configuration parameters.

current_conn_config

```
sid_ble_cfg_conn_param_t sid_pal_ble_adapter_ctx_t::current_conn_config
```

Current connection configuration parameters.

last_conn_config

```
sid_ble_cfg_conn_param_t sid_pal_ble_adapter_ctx_t::last_conn_config
```

Last connection configuration parameters.

GPIO

GPIO

Modules

[Type definitions](#)

Type definitions

Type definitions

Modules

- [GPIO_PinConfig](#)
- [GPIO_LookupItem](#)

Enumerations

```
enum SL_PINout {
    SL_PIN_BUSY = 0
    SL_PIN_ANTSW
    SL_PIN_DIO
    SL_PIN_NRESET
    SL_PIN_NSS
    SL_PIN_MAX
}
Enumeration for GPIO pin assignments.
```

Enumeration Documentation

SL_PINout

SL_PINout

Enumeration for GPIO pin assignments.

This enumeration defines the various GPIO pin assignments.

Enumerator	
SL_PIN_BUSY	GPIO pin for BUSY signal.
SL_PIN_ANTSW	GPIO pin for antenna switch.
SL_PIN_DIO	GPIO pin for DIO signal.
SL_PIN_NRESET	GPIO pin for reset signal.
SL_PIN_NSS	GPIO pin for NSS signal.
SL_PIN_MAX	Maximum number of GPIO pins.

GPIO_PinConfig

Configuration structure for a GPIO.

This structure defines the configuration parameters for a GPIO.

Public Attributes

<code>sid_pal_gpio_direction_t</code>	<code>dir</code>
<code>sid_pal_gpio_input_t</code>	<code>input_mode</code>
<code>sid_pal_gpio_output_t</code>	<code>output_mode</code>
<code>sid_pal_gpio_pull_t</code>	<code>pull_mode</code>

Public Attribute Documentation

dir

`sid_pal_gpio_direction_t GPIO_PinConfig::dir`

Direction of the GPIO pin (input/output).

input_mode

`sid_pal_gpio_input_t GPIO_PinConfig::input_mode`

Input mode configuration for the GPIO pin.

output_mode

`sid_pal_gpio_output_t GPIO_PinConfig::output_mode`

Output mode configuration for the GPIO pin.

pull_mode

`sid_pal_gpio_pull_t GPIO_PinConfig::pull_mode`

Pull mode configuration for the GPIO pin.

GPIO_LookupItem

Lookup item structure for GPIO configuration.

This structure defines the lookup item for GPIO configuration, including port, pin, pin configuration, mode, interrupt settings, and callback information.

Public Attributes

sl_gpio_t	gpio
struct GPIO_PinConfig	PinConfig
sl_gpio_mode_t	mode
sid_pal_gpio_irq_h andler_t	callback
bool	falling
bool	rising
struct GPIO_LookupItem: @1	irq
void *	callbackarg

Public Attribute Documentation

gpio

sl_gpio_t GPIO_LookupItem::gpio

gpio port and pin number.

PinConfig

struct GPIO_PinConfig GPIO_LookupItem::PinConfig

Configuration parameters for the GPIO pin.

mode

sl_gpio_mode_t GPIO_LookupItem::mode

Mode of the GPIO pin.

callback

```
sid_pal_gpio_irq_handler_t GPIO_LookupItem::callback
```

Callback function to be called on GPIO interrupt.

falling

```
bool GPIO_LookupItem::falling
```

Indicates if the interrupt is triggered on falling edge.

rising

```
bool GPIO_LookupItem::rising
```

Indicates if the interrupt is triggered on rising edge.

irq

```
struct GPIO_LookupItem::@1 GPIO_LookupItem::irq
```

Interrupt configuration for the GPIO pin.

callbackarg

```
void* GPIO_LookupItem::callbackarg
```

Argument to be passed to the callback function.

NVM3 Manager

NVM3 Manager

Modules

[Type definitions](#)

Functions

sid_error_t [sli_sid_nvm3_convert_ecode_to_sid_error](#)(Ecode_t nvm3_return_code)
Translates Ecode_t type error codes to sid_error_t type.

Function Documentation

sli_sid_nvm3_convert_ecode_to_sid_error

sid_error_t sli_sid_nvm3_convert_ecode_to_sid_error (Ecode_t nvm3_return_code)

Translates Ecode_t type error codes to sid_error_t type.

Parameters

Type	Direction	Argument Name	Description
Ecode_t	[in]	nvm3_return_code	type error code from nvm3

Returns

- translated error code to sid_error_t

Type definitions

Type definitions

Macros

```
#define SLI_SID_NVM3_KEY_BASE 0xA0000

#define SLI_SID_NVM3_KEY_MIN_APP_REL 0x0

#define SLI_SID_NVM3_KEY_MAX_APP_REL 0x1FFF

#define SLI_SID_NVM3_KEY_MIN_KV_REL 0x0

#define SLI_SID_NVM3_KEY_MAX_KV_REL 0x6FFF

#define SLI_SID_NVM3_KEY_MIN_MFG_REL 0x0

#define SLI_SID_NVM3_KEY_MAX_MFG_REL 0x6FFF

#define SLI_SID_NVM3_KEY_MIN_APP (SLI_SID_NVM3_KEY_BASE + SLI_SID_NVM3_KEY_MIN_APP_REL)

#define SLI_SID_NVM3_KEY_MAX_APP (SLI_SID_NVM3_KEY_BASE + SLI_SID_NVM3_KEY_MAX_APP_REL)

#define SLI_SID_NVM3_KEY_MIN_KV (SLI_SID_NVM3_KEY_MAX_APP + 1)

#define SLI_SID_NVM3_KEY_MAX_KV (SLI_SID_NVM3_KEY_MAX_APP + 1 + SLI_SID_NVM3_KEY_MAX_KV_REL)

#define SLI_SID_NVM3_KEY_MIN_MFG (SLI_SID_NVM3_KEY_MAX_KV + 1)

#define SLI_SID_NVM3_KEY_MAX_MFG (SLI_SID_NVM3_KEY_MAX_KV + 1 + SLI_SID_NVM3_KEY_MAX_MFG_REL)

#define SLI_SID_NVM3_KEY_BASE_APP SLI_SID_NVM3_KEY_MIN_APP

#define SLI_SID_NVM3_KEY_BASE_KV SLI_SID_NVM3_KEY_MIN_KV

#define SLI_SID_NVM3_KEY_BASE_MFG SLI_SID_NVM3_KEY_MIN_MFG

#define SID_PAL_MFG_STORE_SL_NVM3_VERSION (SID_PAL_MFG_STORE_CORE_VALUE_MAX + 1)
    SiLabs-specific MFG key for the NVM3 version MFG object.

#define SID_PAL_MFG_STORE_SL_NVM3_VERSION_SIZE 4
    Size of the NVM3 version in bytes.

#define SLI_SID_NVM3_VALIDATE_KEY (region, key)
    Validate if a key is within the specified region's range.

#define SLI_SID_NVM3_MAP_KEY (region, key)
    Map a key to the specified region's base key.
```


Timer Types

Timer Types

Modules

[Type definitions](#)

Type definitions

Type definitions

Modules

[sid_pal_timer_impl_t](#)

Typedefs

```
typedef void(* sid\_pal\_timer\_cb\_t)(void *arg, sid_pal_timer_t *originator)  
Timer callback type.
```

Typedef Documentation

sid_pal_timer_cb_t

```
typedef void(* sid_pal_timer_cb_t) (void *arg, sid_pal_timer_t *originator) )(void *arg, sid_pal_timer_t *originator)
```

Timer callback type.

Note

- The callback is allowed to execute absolute minimum amount of work and return as soon as possible
- Implementer of the callback should consider the callback is executed from ISR context

sid_pal_timer_impl_t

Timer storage type.

Typedef for the timer implementation structure.

Note

- This is the implementor defined storage type for timers.

This typedef defines a type alias for the timer implementation structure.

Public Attributes

struct sid_timespec	alarm
struct sid_timespec	period
sl_sleeptimer_time_r_handle_t	sleeptimer_handle
sid_pal_timer_cb_t	callback
bool	is_periodic
bool	has_started
uint32_t	period_in_ms
void *	callback_arg

Public Attribute Documentation

alarm

```
struct sid_timespec sid_pal_timer_t::alarm
```

The alarm time for the timer.

period

```
struct sid_timespec sid_pal_timer_t::period
```

The period of the timer.

sleeptimer_handle

```
sl_sleeptimer_timer_handle_t sid_pal_timer_t::sleeptimer_handle
```

Handle for the sleep timer.

callback

```
sid_pal_timer_cb_t sid_pal_timer_t::callback
```

Callback function to be called when the timer expires.

is_periodic

```
bool sid_pal_timer_t::is_periodic
```

Indicates if the timer is periodic.

has_started

```
bool sid_pal_timer_t::has_started
```

Indicates if the timer has started.

period_in_ms

```
uint32_t sid_pal_timer_t::period_in_ms
```

The period of the timer in milliseconds.

callback_arg

```
void* sid_pal_timer_t::callback_arg
```

Argument to be passed to the callback function.

Sidewalk Sample Applications

Sample Applications

Introduction

A number of examples are provided with Simplicity Studio and the Proprietary Flex SDK. Each example has an associated README that explains the purpose of the example and how to use it. Some of the examples have more extensive documentation, and that is included in this section.

SoC Dynamic Multiprotocol Light

This is a Dynamic Multiprotocol reference application demonstrating a light bulb that can be switched via Bluetooth or Amazon Sidewalk (BLE or FSK radio layer).

It allows a BLE central device to control the LED on the mainboard and receive button press notifications. To test this demo, install Simplicity Connect mobile application. Simultaneously, this sample application leverages the Amazon Sidewalk protocol to connect to the cloud using either BLE or sub-GHz FSK modulation. The Sidewalk endpoint connects to a gateway, allowing it to exchange data with the AWS cloud.

You interact with the endpoint either by pressing the mainboard buttons, through the BLE Simplicity Connect application or through the AWS cloud by issuing CLI commands.

You can learn more about Silicon Labs Multiprotocol libraries on [Silicon Labs website](#).

You can find more details about SoC Dynamic Multiprotocol Light on the [Silicon Labs Amazon Sidewalk Github repository](#)

SoC Bluetooth Sub-GHz CLI Application

The Bluetooth sub-GHz Command Line Interface (CLI) sample application allows the user to interact with the endpoint using CLI commands. The application leverages the Amazon Sidewalk protocol to exchange data between the endpoint and the AWS Cloud using one of the 3 radio layers. It is possible to initialize and start the stack using any one of the 3 radio layers (BLE, FSK, or CSS). A one-time registration phase (using either BLE or FSK, as registration does not occur over CSS) is required at first boot.

You can find more details about SoC Bluetooth Sub-GHz CLI Application on the [Silicon Labs Amazon Sidewalk Github repository](#).

SoC Empty

The Amazon Sidewalk Empty sample application is a minimalist template designed for developing Amazon Sidewalk applications. It can be used alongside the developer's guide, which helps you use and configure the Amazon Sidewalk solution and implement your own application. Before diving into this application, it is recommended to review the Getting Started guide to become familiar with the Amazon Sidewalk workflow, and then refer to the [Developer's Guide](#).

You can find more details about SoC Empty App on the [Silicon Labs Amazon Sidewalk Github repository](#).

SoC Bluetooth Sub-GHz Hello Neighbor

The Hello Neighbor sample application leverages the Amazon Sidewalk protocol to connect to the cloud using either BLE or sub-GHz FSK / CSS modulation (after an initial registration phase over BLE, if necessary). The Sidewalk endpoint connects to a gateway, allowing it to exchange data with the AWS cloud. You interact with the endpoint either by pressing the mainboard buttons (not supported when using KG100S) or issuing CLI commands.

You can find more details about SoC Bluetooth Sub-GHz Hello Neighbor on the [Silicon Labs Amazon Sidewalk Github repository](#).

SoC Out-of-the-Box (OOB) Demo

SoC Bluetooth Out-of-the-Box (OOB) Demo

INFO: This application image is provided for the sole purpose of restoring the factory-default OOB demo application on EFR32xG24 2.4 GHz 20 dBm Radio Boards (BRD4187C) included in the [Silicon Labs Pro Kit for Amazon Sidewalk](#). It will not function on any other device, nor on any BRD4187C boards sourced independently from the above kits.

The OOB (Bluetooth) sample application leverages the Amazon Sidewalk protocol to connect to the cloud using a Bluetooth connection. The Sidewalk endpoint connects to a gateway, allowing it to exchange data with the AWS cloud. The user interacts with the endpoint either by pressing the main board buttons or through GUI elements in the associated web-based application running in AWS.

You can find more details about Bluetooth OOB on the [Silicon Labs Amazon Sidewalk Github repository](#).

SoC Bluetooth Sub-GHz Out-of-the-Box (OOB) Demo

INFO: This application image is provided for the sole purpose of restoring the factory-default OOB demo application on KG100S Sidewalk Module Radio Boards (BRD4332A) included in the [Silicon Labs Pro Kit for Amazon Sidewalk](#). It will not function on any other device, nor on any BRD4332A boards sourced independently from the above kits.

The OOB (Bluetooth & sub-GHz) sample application leverages the Amazon Sidewalk protocol to connect to the cloud using sub-GHz FSK / CSS modulation (after an initial registration phase over BLE, if necessary). The Sidewalk endpoint connects to a gateway, allowing it to exchange data with the AWS cloud. The user interacts with the endpoint either by pressing the main board buttons or through GUI elements in the associated web-based application running in AWS.

You can find more details about Bluetooth Sub-GHz OOB on the [Silicon Labs Amazon Sidewalk Github repository](#).

SoC Production Device Provisioner (PDP)

The Production Device Provisioner application is a tool to enable your production product to leverage Secure Vault, generating the key pair directly on the device thus, limiting the exposure of the private key.

In Sidewalk, you can leverage Secure Vault to store sensitive data (private keys) in a secure place. A set of scripts and this application is provided to use the Secure Element in the Amazon Sidewalk context.

The PDP application is used to exchange certificate data and to communicate with the Secure Element through APIs. The PDP application can be used for provisioning and is automatically deleted upon reboot, as it is a transient application running in RAM.

For more information on product manufacturing in the Sidewalk context, refer to [Manufacturing a Product](#).

You can find more details about SoC Production Device Provisioner on the [Silicon Labs Amazon Sidewalk Github repository](#)

SoC Qualification

The SoC Qualification example is a command-line interface (CLI) based sample application. It enables testing of the Sidewalk API functions via the command line. This is the reference application for Sidewalk qualification and contains all the commands needed to pass the certification with Amazon.

Amazon fully manages the qualification process. For any questions regarding this process, reach out to Amazon support directly. This sample application serves as a guide for developers to implement the interface with Amazon's qualification tests. The qualification sample application can be used to port custom hardware and should be included in the application submitted to Amazon for approval.

You can find more details about SoC Qualification on the [Silicon Labs Amazon Sidewalk Github repository](#).

Overview

Platform Resources

When you develop in the Silicon Labs Simplicity SDK, you have additional resources available to you through the Gecko Platform. This section includes information on the following topics.

- **Bootloading**: Bootloading allows you to update application firmware images on Wi-SUN devices. This section provides background information about bootloading using the Silicon Labs Gecko Bootloader.
- **Non-Volatile Memory Use**: This section offers an introduction to non-volatile data storage and describes how to use NVM3 data storage.
- **Security**: Silicon Labs offers a range of security features depending on the part you are using and your application and production needs.

Overview

Bootloading Amazon Sidewalk Applications

Bootloading allows you to update application firmware images on Sidewalk devices. This section provides background information about bootloading using the Silicon Labs Gecko Bootloader.

- [Bootloader Fundamentals \(PDF\)](#): Introduces bootloading for Silicon Labs networking devices. Discusses the Gecko Bootloader and bootloader file formats.
- [Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher \(PDF\)](#): Describes the high-level implementation of the Silicon Labs Gecko Bootloader for EFR32 SoCs and NCPs, and provides information on how to get started using the Gecko Bootloader with Silicon Labs wireless protocol stacks in GSDK 4.0 and higher or Simplicity SDK 2024.6.0 and higher.

Overview

Non-Volatile Memory Use

This section offers an introduction to non-volatile data storage and describes how to use NVM3 data storage.

- [Non-Volatile Data Storage Fundamentals \(PDF\)](#): Introduces non-volatile data storage using flash and the three different storage implementations offered for Silicon Labs microcontrollers and SoCs: Simulated EEPROM, PS Store, and NVM3.
- [Using NVM3 Data Storage \(PDF\)](#): Explains how NVM3 can be used as non-volatile data storage in various protocol implementations.

Overview

Security

Silicon Labs offers a range of security features depending on the part you are using and your application and production needs.

- [IoT Security Fundamentals \(PDF\)](#): Introduces the security concepts that must be considered when implementing an Internet of Things (IoT) system. Using the ioXt Alliance's eight security principles as a structure, it clearly delineates the solutions Silicon Labs provides to support endpoint security and what you must do outside of the Silicon Labs framework.
- [Integrating Crypto Functionality with PSA Crypto vs. Mbed TLS \(PDF\)](#): Describes how to integrate crypto functionality into applications using PSA Crypto compared to Mbed TLS.

Manufacturing a Product

Manufacturing a Product

This section provides information on how to mass manufacture a Silicon Labs Sidewalk-enabled product. It is based on Amazon's documentation called [Manufacturing Amazon Sidewalk devices for mass production](#) and describes the Silicon Labs specific steps in detail as well as a short summary of the global production flow in the factory.

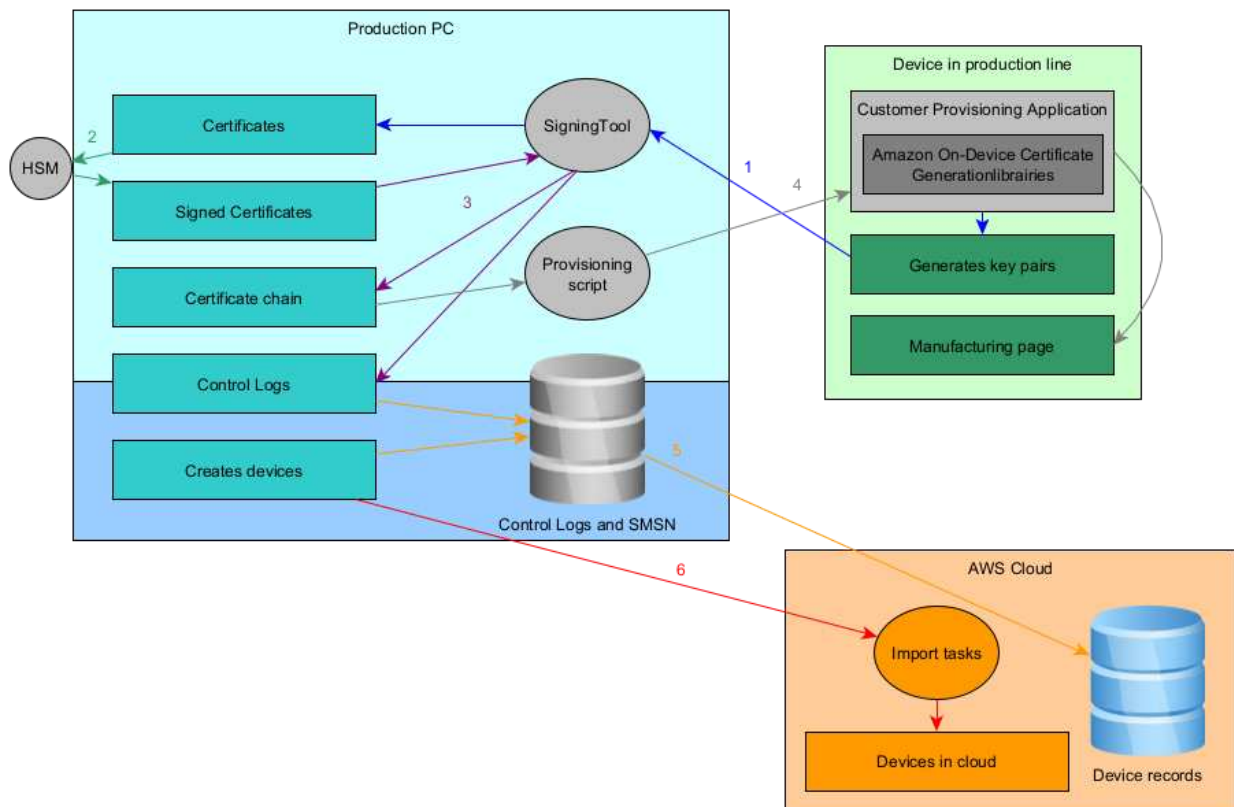
To enable the production of your Sidewalk-based product, it needs to be certified (see corresponding [documentation on qualification](#)). Amazon provides a set of tools to facilitate production and this section describes the specific Silicon Labs tools that integrate with them. For more information on those tools and how to obtain them, check [Amazon documentation on manufacturing](#) or contact your Amazon Sidewalk business representative.

Standard Amazon Manufacturing Flow

Amazon documentation describes how a manufacturer can mass produce Sidewalk products. This flow is used to manufacture a large number of devices, and can be divided into 4 phases:

- [Setting up the factory infrastructure](#)
- [Manufacturing at the factory](#)
- [Uploading the device records \(control logs\) to Amazon](#)
- [Creating the recorded devices in your AWS cloud account](#)

The Silicon Labs added-value elements described in the following sections impact the "Manufacturing at the factory" phase. All other steps stay unchanged and follow the Amazon standard flow described in the figure below.



1. The endpoint key pair is generated directly on the device. A custom application (leveraging Amazon Provisioning Libraries) triggers the certificates generation by calling the Signing Tool provided by Amazon.
2. The HSM, provisioned by Amazon, is used to sign the Certificate Signing Request (CSR) and outputs the certificate.
3. The signing tool is used along with the HSM to create the certificate chain from the endpoint certificate to the Sidewalk Network server and Application certificates. Control logs are generated as well.
4. Once the certificate chain is generated, it can be added to the manufacturing page generated during this flow.
5. Control logs are assembled into a database and can be uploaded to the cloud to populate the devices record.
6. Once all devices are entered into the record, the devices are known to the backend and can be created using the import tasks. A list of all SMSNs of created devices can be uploaded to the cloud to create the corresponding virtual devices in IoT Core.

For added security, the step requiring AWS connection can be done periodically on another computer. This way, the production PC can be totally offline and only the control logs database is shared to the computer handling AWS and connectivity tasks.

Silicon Labs Security Added Value

On Silicon Labs EFR32 Series 2 platforms, you can leverage Secure Vault to store sensitive data (private keys) in a secure place. A set of scripts and a device application, called Production Device Provisioner (PDP), is provided to use the Secure Element in the Amazon Sidewalk context.

The goal of the production scripts is to facilitate data provisioning by providing ready-to-use scripts and a way of communicating with the embedded device from a host computer. The production flow for development and manufacturing setups stays the same, which makes the transition from development to manufacturing transparent.

In addition to the production scripts, to communicate between the device and the production line PC, Silicon Labs provides a PDP application. It is used to exchange certificate data and to communicate with the Secure Element through APIs. The PDP application can be used for provisioning and is automatically deleted upon reboot, as it is a transient application running in RAM.

Generating the key pair directly on the device limits the exposure of the private key. Amazon security requirements are evolving to require this "on-device" generation of the private key or new products. Starting with Amazon Sidewalk SDK 1.16, all products must be produced using the on-device certification generation. YubiHSM requested after Sidewalk SDK 1.16 release must use the on-device certification generation. For products released before Sidewalk SDK 1.16, keys can be generated on the production machine following the legacy flow described in the [OpenSSL Private Key Provisioning](#) section.

Provisioning Methods

All available methods to provision a device appear in the following table. It includes the prototyping method used during development, and the two manufacturing methods (depending on your Amazon Sidewalk SDK version). All of these methods can be used alongside the Secure Vault to store the credentials.

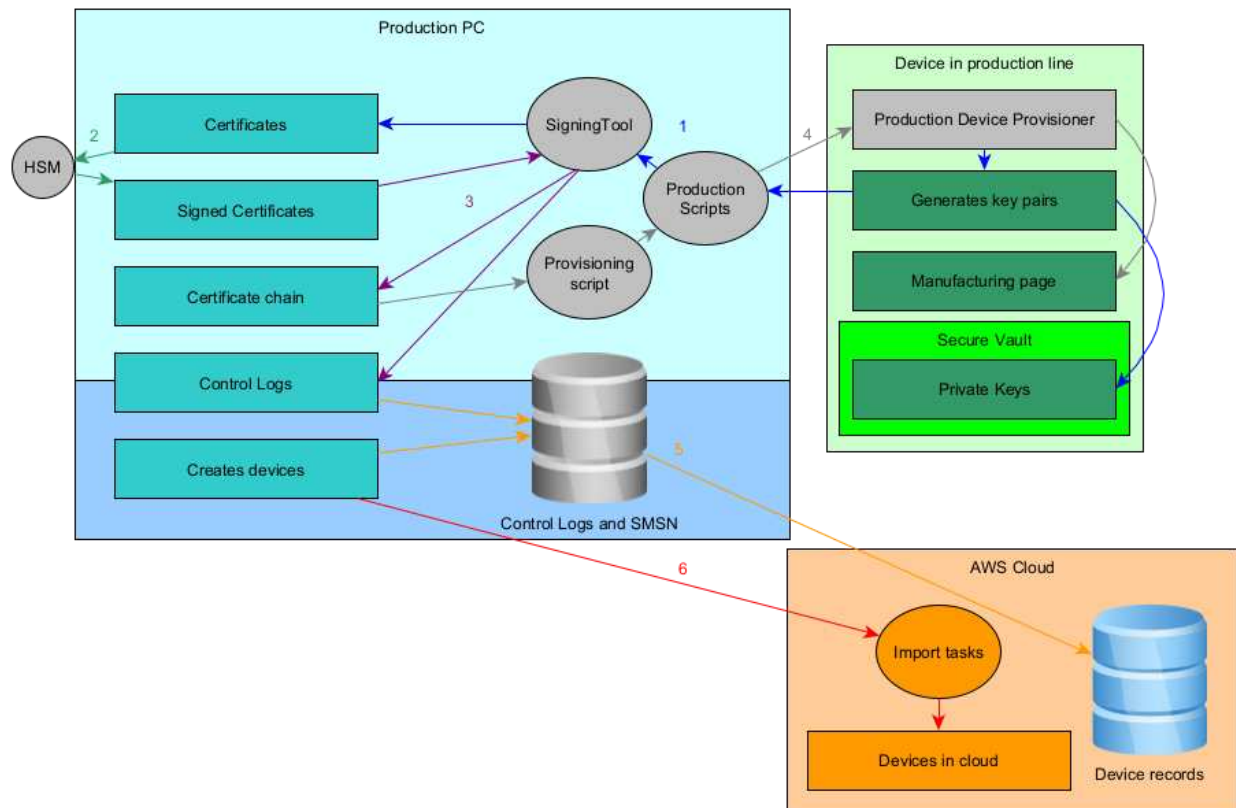
Method Name	Certificate Generation Method	Flow	YubiHSM	Crypto Library	Security Level	Usage
On-device Certificate Generation	On-device	Manufacturing	Yes	Silicon Labs PSA Crypto	HIGH	Volume Production
Prototyping	Cloud	Prototyping	No	OpenSSL	LOW	Development only
OpenSSL Private Key Provisioning	Offline/Production PC	Legacy Manufacturing	Yes	OpenSSL	MEDIUM	Volume Production (Deprecated as of 1.16)

The prototyping flow leveraging Secure Vault is covered in the [OpenSSL Private Key Provisioning](#) section below.

On-Device Certificate Generation

In Amazon Sidewalk SDK 1.16, Amazon introduced a new flow called On-Device Certificate Generation. The main point of this flow is to generate the private key directly on the device and only communicate the CSR and public certificate to the

production PC. The private keys never leave the device. Silicon Labs provides a tool that builds on the On-Device Certificate Generation flow and leverages Secure Vault to store the keys. This enhanced flow is described in the figure below.



1. The endpoint key pair is generated directly on the device. The private keys are wrapped by the Secure Element as described in the [Secure Vault documentation section 3.3.2](#) using the Production Device Provisioner application provided by Silicon Labs.
2. The HSM, provisioned by Amazon, is used to sign the Certificate Signing Request (CSR) and outputs the certificate.
3. The signing tool is used along with the HSM to create the certificate chain from the endpoint certificate to the Sidewalk Network server and Application certificates. Control logs are generated as well.
4. Once the certificate chain is generated, it can be added to the manufacturing page generated during this flow. The manufacturing page, containing the private keys ID from Secure Vault, is written to the device flash.
5. Control logs are assembled into a database and can be uploaded to the cloud to populate the devices record.
6. Once all devices are entered into the record, the devices are known to the backend and can be created using the import tasks. A list of all SMSNs of created devices can be uploaded to the cloud to create the corresponding virtual devices in IoT Core.

Silicon Labs Production Provisioning Walkthrough

Both on-device and offline manufacturing certificate generation flows use the production scripts and application.

Production provisioning scripts consist of two functions: initialization and provisioning.

The **initialization script** generates an image with the static data of a Sidewalk device (the common part of the Amazon Sidewalk certificate) and optionally a Sidewalk application. The generated image is identical for all the devices in question. In the case of on-device certificate generation, the output of this script is just the custom application binary without any static data added.

The **provisioning script**, on the other hand, flashes the initialization image generated in the previous step, flashes the Production Device Provisioner RAM application, and provisions device-specific information via this application. Each device is provisioned with a unique set of security credentials (device-specific part of the Sidewalk certificate).

The scripts can be used in two certificate generation modes:

- On-device certificate generation: Device certificate is generated and private keys are wrapped by the Secure Element on the device.
- Private key provisioning: Device certificate is generated outside of the device and private keys are wrapped by the Secure Element. Deprecated since Amazon Sidewalk SDK 1.16, see [the section at the end of this page](#).

The production scripts are located in `tools/scripts/public/pdp` folder of the extension.

Enabling the Secure Element

For starters, the Secure Vault should be enabled in your Sidewalk application. To enable Secure Vault in the application, you must modify the `config/sl_sidewalk_common_config.h` file in your project as follows:

In file `config/sl_sidewalk_common_config.h`, add definition for `SV_ENABLED`.

```
#ifndef SV_ENABLED
#define SV_ENABLED 1
#endif //SV_ENABLED
```

Compiling the Production Device Provisioner Application

The production scripts communicate between the Amazon Signing Tool and with a device through the Production Device Provisioner application.

To create a Production Device Provisioner application, follow the getting started documentation and compile the **Production Device Provisioner** instead of the Hello Neighbor application: [Create an Amazon Sidewalk Project](#).

To compile the application, go to your newly created Production Device Provisioner application folder in the Simplicity IDE perspective. In the left explorer view, right-click the project and select **Build Project**. When the compilation is finished, you will find the compiled binary in your Simplicity Studio workspace under `your_pdp_project_name/GNU ARM v10.3.1 - Default/your_project_name.s37`. This file is needed by the production provisioning script in the next step.

On-Device Certificate Generation

For On-Device Certificate Generation, we generate the private keys on the device; only the CSR and public certificate are communicated to the production PC. Private data never leaves the device. Only the public data needed to create the device on the cloud is communicated to the production PC.

Here is a detailed description of the steps:

1. Production Device Provisioner (PDP) script sends a command to the device that triggers Amazon on-device certificate generation module's init API, which initializes the process context.
2. PDP script sends a command to the device that triggers amazon on-device certificate generation module's generate SMSN (Sidewalk MANufacturing Serial Number) API. It sends the command along with the device type (provided by Amazon), device serial number (generated per device), and ApID (provided by Amazon) parameters. The device sends back generated SMSN.
3. PDP script sends a command to the device that triggers Amazon on-device certificate generation module's generate CSR API, two times in a row, one for ED25519 elliptic curve and another for P256R1 elliptic curve. It's at this stage that the device private keys (two keys, one for each curve) are generated in Secure Vault and wrapped with the device private key before being stored on the default NVM3 instance.
4. PDP script calls Sidewalk signing tool provided by Amazon. It passes production tag, HSM connection address, HSM PIN, ED25519 CSR, p256r1 CSR and ApID parameters to the signing tool.

5. Signing tool sends back signed CSRs (certificate chain for each curve) to the PDP script. PDP script sends a command to the device that triggers Amazon on-device certificate generation module's write certificate chain API to insert signed CSRs to the device.
6. PDP script sends a command to the device that triggers Amazon on-device certificate generation module's write application server public key API to insert the application server public key to the device.
7. PDP script sends a command to the device that triggers Amazon on-device certificate generation module's verify and store API. This basically writes all the generated manufacturing information onto the manufacturing image NVM3 instance.

In this walkthrough, we are going to use the manufacturing provisioning script that leverages Secure Vault to store the private keys.

Running the Initialization Script

For On-Device Certificate Generation, the initialization script is not needed anymore. Instead, directly use the Sidewalk application as input to the provisioning script.

Running the Provisioning Script

The provisioning script takes an SoC family, Sidewalk application image, Production Device Provisioner image, and YubiHSM related information as input arguments, and provisions the end device with device credentials generated on the device. This script must be run on each device in production, using the following steps:

1. Connect your device with a supported EFR32 chip.
2. Connect the YubiHSM key.
3. Run the following command in as administrator: `sudo yubihsm-connector -d`.
4. Navigate to `tools/scripts/public/pdp` directory.
5. Fill the configuration file with the product information.
6. Run the following commands.

INFO ⓘ: If you prefer to flash the initialization image before the provisioning, leave the `sid-init-img` field empty.

WARNING ⚠: Simplicity Commander version 1v16 and JLink RTT version 7.96 are the minimum required versions to run the scripts.

```
{
  "part": "part_number_like_efr32zg28b322f1024im68",
  "sid_init_img": "out/sid_init_img.s37",
  "pdp_img": "/path/to/pdp/app.s37",
  "dev_type": "dev_type_like_A232AX65BNIW2J",
  "apid": "advertised_product_id_like_zGhh",
  "app_srv_pub_key": "app_srv_pub_key_like_887fc49bb23ffbbdb98550040506c7eddf696707519b0c1c603e4bf8801631c6",
  "sst_prod_tag": "prod_tag_like_RNET_DAK_DUMMY",
  "sst_hsm_conn_addr": "hsm_yubi_connector_addr_like_http://localhost:12345",
  "sst_hsm_pin": "yubi_hsm_pin_like_1234"
}
```

```
python3 provision_silabs.py \
--dsn <device_serial_no_like_G6F1JN06119201GP> \
--prod-config template_prod_config.json
--pdp-mode on_dev_cert_gen
```

With fields as follows:

- **part:** Part number (efr32zg28b322f1024im68)
- **sid_init_img:** Sidewalk custom application binary (`.s37` file)
- **pdp_img:** Sidewalk Production Device Provisioner application binary (`.s37` file)
- **dev_type:** This field is provided by Amazon upon HSM provisioning request
- **apid:** This value can be extracted from the device profile associated to your HSM
- **app_srv_pub_key:** This value can be extracted from the device profile associated to your HSM
- **sst_prod_tag:** This field is provided by Amazon upon HSM provisioning request
- **sst_hsm_conn_addr:** This is the socket address connected to your HSM usb key, it is `http://localhost:12345` by default in yubico toolkit
- **sst_hsm_pin:** This field is provided by Amazon upon HSM provisioning request

And arguments as follows:

- **dsn:** This field is used to generate the Sidewalk SMSN, it shall be unique for each device
- **pdp-mode:** To choose between On-device certificate generation and OpenSSL private key provisioning
- **prod-config:** Path to the configuration file

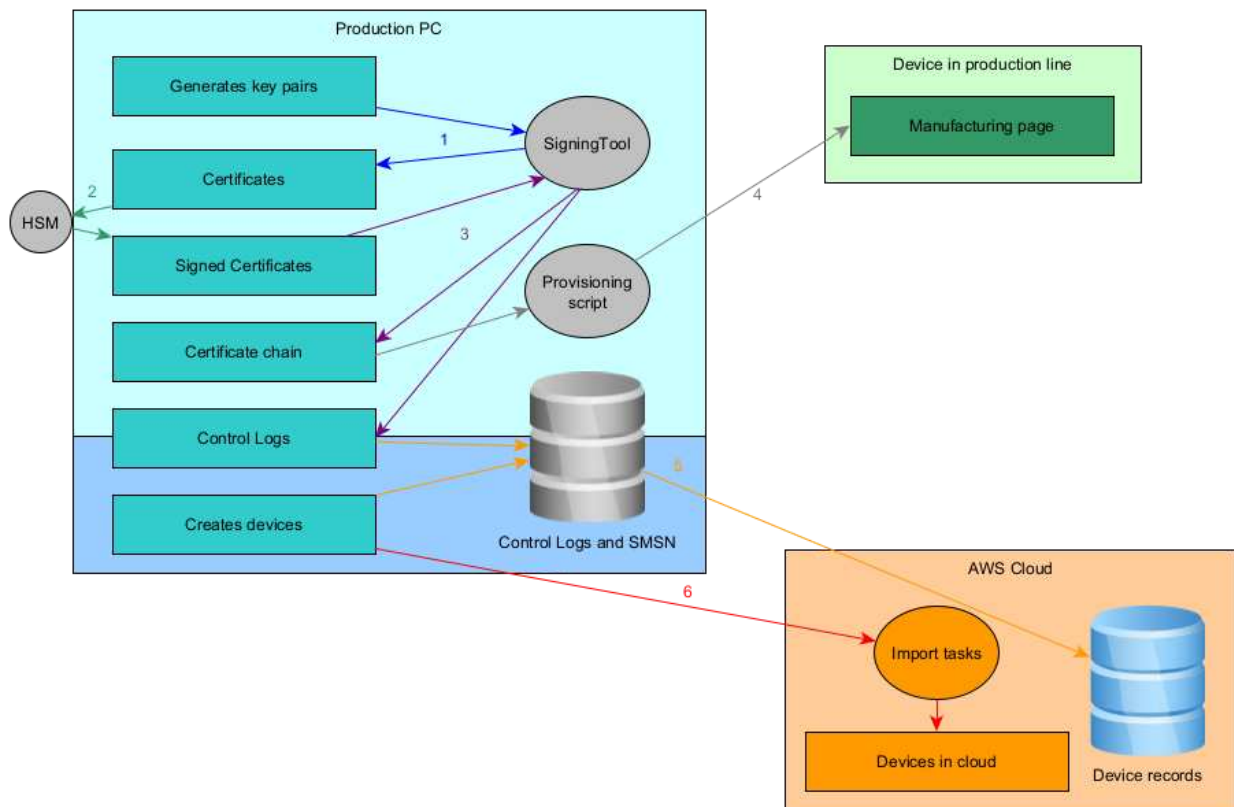
To use the script with command-line arguments instead of a configuration file, simply drop the `--prod-config` parameter and add all configuration fields as arguments.

After completing this step, the Sidewalk end device is now provisioned and all public information is stored in the `out/` folder for the device upload to the cloud following the standard Amazon flow.

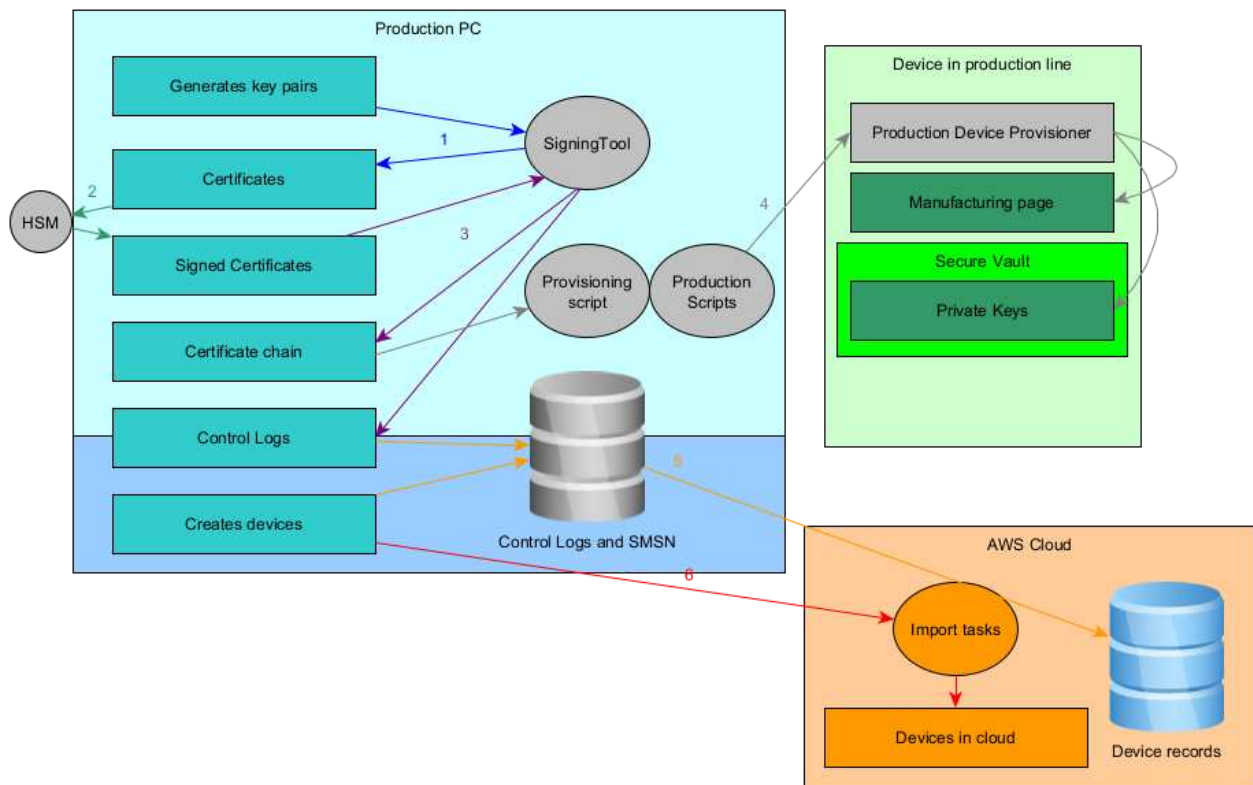
OpenSSL Private Key Provisioning (Deprecated)

This manufacturing flow is deprecated as of Amazon Sidewalk SDK 1.16. See the [security section](#) above for information on whether or not you can use this flow for your product.

This flow is an extension of the now deprecated Standard Amazon flow below.



In the Silicon Lab version of this flow, the private key are wrapped by the Secure Vault instead of the flash. The manufacturing page contains references to the Secure Vault slot storing the private keys.



1. The endpoint key pair is generated on the production computer and the Signing Tool provided by Amazon is called to start certificate generation.
2. The HSM, provisioned by Amazon, is used to sign the Certificate Signing Request (CSR) and outputs the certificate.
3. The signing tool is used along with the HSM to create the certificate chain from the endpoint certificate to the Sidewalk Network server and Application certificates. Control logs are generated as well.
4. **Once the certificate chain is generated, it can be used to generate the manufacturing page with the provisioning script. The private keys are wrapped by the Secure Element as described in the [Secure Vault documentation section 3.3.1](#). The manufacturing page, containing the private keys ID from Secure Vault, is flashed to the device.**
5. Control logs are assembled into a database and can be uploaded to the cloud to populate the devices record.
6. Once all devices are entered into the record, the devices are known to the backend and can be created using the import tasks. A list of all SMSNs of created devices can be uploaded to the cloud to create the corresponding virtual devices in IoT Core.

Leveraging Secure Vault does not use the manufacturing page (MFG) generated by the prototyping scripts. Instead it uses the `WirelessDevice.json` and `DeviceProfile.json` files obtained as outputs from the prototyping scripts described in the [Prototyping APIs section](#). The steps are as follows:

1. The initialization script creates an output file (.s37) containing a part of the manufacturing page that is identical for all the devices created using the same Device Profile.
2. The output of this initialization script can optionally include the Sidewalk application binary.
3. The device-specific information is provisioned to the device using the provisioning script. Here, the rest of the manufacturing page on the device will be completed with the device-specific information.

To summarize, the initialization script generates an output file that can be flashed to all devices. Then the provisioning script is used to provision the device credentials. The credentials are wrapped by keys contained in the Secure Vault and stored in the Default NVM3 Instance as described in the [Non-Volatile Memory Use section](#).

OpenSSL Private Key Provisioning Walkthrough (Deprecated)

Device certificate is generated outside of the device so it is intrinsically less secure. Amazon's signing server tool generates device certificates using OpenSSL and YubiHSM.

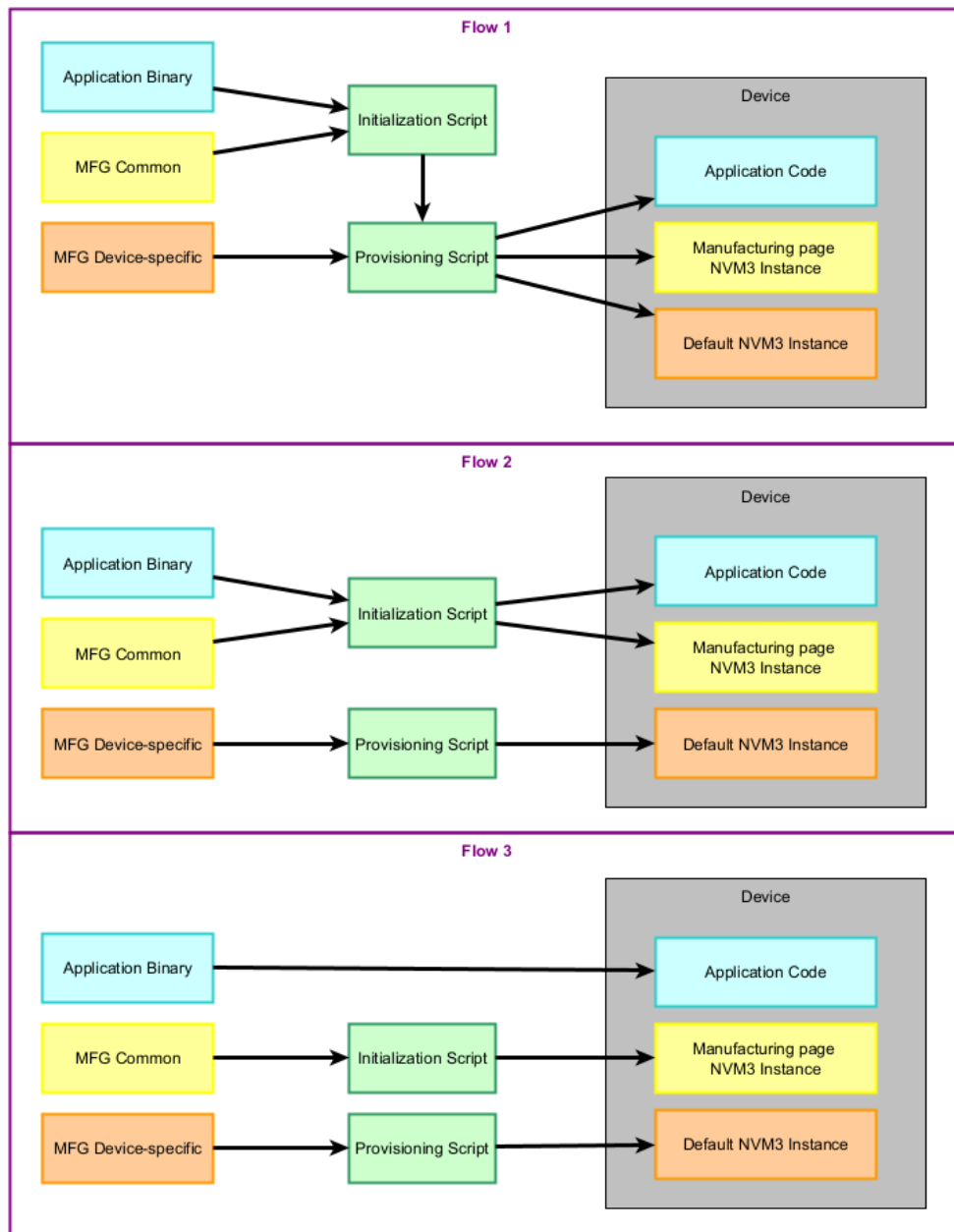
Here is an overview of what happens in this mode:

1. All dynamic data is sent to the device.
2. ED25519 and P256R1 device private keys are wrapped by the Secure Vault.

Production provisioning scripts consist of two functions: initialization and provisioning. The scripts can be used in three different configurations:

1. You can include the application binary in the initialization script and use the output of the script in the provisioning script. This flashes both binaries containing the application, MFG common section, and MFG device-specific section.
2. You can call the initialization script with the application binary. Flash the resulting image, which contains the application and MFG common section, on your device. Then call the provisioning script without the initialization image (sid-init-img) option to flash the remaining MFG device-specific information.
3. You can omit the application binary in the initialization script (sid-app option). You will have to flash the output of the initialization script (MFG common section) before calling the provisioning one. Then you can call the provisioning script (MFG device-specific section). The application binary can be flashed at any point or added as an argument of the provisioning script (using the sid-init-img option).

In all the flows, the only hard requirement is to flash the output of `initialize_silabs.py` (sid.s37) **before** calling `silabs_provision.py`. The common information in the manufacturing page must be on the device before the device-specific one. See the diagram below for more details.



The production script folder contains several python scripts (`initialization_silabs.py` and `provision_silabs.py`). In the following instructions on each script, two commands are shown, one for prototyping and one for manufacturing. The prototyping one can be used with the output files of the Amazon prototyping flow (`DeviceProfile.json` and `WirelessDevice.json`) while the manufacturing one can be used with Amazon manufacturing flow output files containing the private keys.

Before provisioning your device, you will need to [enable Secure Vault in your Sidewalk application](#) and to [compile the Production Device Provisioner application](#) needed in the next steps.

⚠ WARNING ⚠: Simplicity Commander version 1v16 and JLink RTT version 7.96 are the minimum required versions to run the scripts.

Running the Initialization Script

The initialization script takes an Amazon Sidewalk device certificate, SoC family and Sidewalk application binary as input arguments and generates a binary file that contains the static (common) part of the Sidewalk certificate and the Sidewalk application provided as input arguments. The output binary file is common to all the product instances so it does not contain any device-specific information (device-specific information is provisioned in the next step). This script can be run once to generate the common binary in the production, following these steps:

1. Navigate to `tools/scripts/public/pdp` directory.
2. Run the commands below.

```
pip3 install -r requirements.txt

# For manufacturing
python3 initialize_silabs.py --sid-cert </path/to/certificate.json> --sid-cert-type prod --part <part> --sid-usr-app-img </path/to/user/sid/app.s37> -
-pdp-mode priv_key_prov

# For prototyping
python3 initialize_silabs.py --sid-cert </path/to/WirelessDevice.json> --sid-cert-type proto --sid-dev-prof </path/to/DeviceProfile.json> --part
<part> --sid-usr-app-img </path/to/user/sid/app.s37> --pdp-mode priv_key_prov
```

With arguments as follows:

- **sid-cert:** Sidewalk device certificate containing private key pairs or Wireless Device JSON file from AWS.
- **sid-cert-type:** Either *prod* or *proto* depending on the flow in use (manufacturing or prototyping)
- **sid-dev-prof:** If prototyping flow, we need to give the Device Profile JSON file
- **part:** Part number (like efr32zg28b322f1024im68)
- **sid-usr-app-img:** Sidewalk custom application
- **pdp-mode:** To choose between On-device certificate generation and OpenSSL private key provisioning

At the end of this step, an output file (in `tools/scripts/public/pdp/out`) that contains the static (common) part of the Sidewalk certificate and the Sidewalk application is generated. It will be used as an input argument in the next step.

Running the Provisioning Script

The provisioning script takes a Sidewalk device certificate, SoC family, initialization image (generated in the previous step), and provisioning image (.bin image generated in Simplicity Studio by compiling the Production Device Provisioner sample application) as input arguments and provisions the end device with device credentials extracted from the Sidewalk device certificate. This script must be run per device in production, using the following steps:

1. Connect your device with a supported EFR32 chip.
2. Navigate to `tools/scripts/public/pdp` directory.
3. Run the following commands.

INFO: If you prefer to flash the initialization image before the provisioning, you can skip the `--sid-init-img` argument.

```
# For manufacturing
python3 provision_silabs.py --sid-cert </path/to/certificate.json> --sid-cert-type prod --part <part> --sid-init-img out/sid_init_img.s37 --pdp-img
</path/to/pdp/app.s37> --pdp-mode priv_key_prov

# For prototyping
python3 provision_silabs.py --sid-cert </path/to/WirelessDevice.json> --sid-cert-type proto --part <part> --sid-init-img out/sid_init_img.s37 --pdp-
img </path/to/pdp/app.s37> --pdp-mode priv_key_prov
```

With arguments as follows:

- **sid-cert:** Sidewalk device certificate containing private key pairs or Wireless Device JSON file from AWS.

- **sid-cert-type:** Either *prod* or *proto* depending on the flow in use (manufacturing or prototyping)
- **part:** Part number (like efr32zg28b322f1024im68)
- **sid-init-img:** Initialization image generated in the previous step
- **pdp-img:** Sidewalk Production Device Provisioner application
- **pdp-mode:** To choose between On-device certificate generation and OpenSSL private key provisioning

After completing this step, the Sidewalk end device is now provisioned with device-specific credentials.