amazon sidewalk

Amazon Sidewalk Location Library Developer Guide

Protocol Stack 1.0, Document Revision A

September 30, 2025

©2025 Amazon Technologies, Inc.
Non-confidential

Use of these Amazon Sidewalk specifications (the "Specifications") is subject to your compliance with the AWS Customer Agreement and the Service Terms (collectively, the "Agreement"), including all disclaimers and limitations as to such use contained therein.

All statements, information, and data contained herein is subject to change without further notice to improve reliability, function, or design. Certain parameters may vary in different applications and performance may vary over time. It is your responsibility to validate that Amazon Sidewalk is suitable for your particular device or application.

No license, express or implied, by estoppel or otherwise, to any intellectual property rights are granted by this document.

Amazon Sidewalk is not intended for use in, or in association with, the operation of any hazardous environments or critical systems that may lead to serious bodily injury or death or cause environmental or property damage, and you are solely responsible for all liability that may arise in connection with any such use.

This document is Non-Confidential.

©2024 Amazon Technologies, Inc. Amazon and all related marks are trademarks of Amazon.com, Inc. or its affiliates.

Contents

Chapter 1

Overview

1.1 Introduction

The Amazon Sidewalk Location Library enables enables Sidewalk customers to use a range of cloud-based location solvers. Available in SDK >=1.19, this library provides an API that automatically selects the most power and time efficient location mechanism based on available hardware and network conditions. Developers will also have finer grain control to configure and override the mechanism used to determine location as required.

1.2 Supported Location Methods

The location library supports three types of location resolution methods:

1. **Sidewalk Network Location over BLE** - Leverages existing BLE connections to determine location through the Sidewalk network
2. **WiFi Scan** - Scans for nearby WiFi access points and sends MAC addresses to the cloud for location resolution
3. **GNSS Scan** - Uses Global Navigation Satellite System to collect satellite vehicle data for coordinate resolution

These methods are supported on two available link types:

- BLE (Bluetooth Low Energy)
- LoRa (Long Range)

Note: FSK with WiFi/GNSS scanning enabled is not currently supported.

WiFi and GNSS scanning leverage the Semtech LoRa Basics Modem middleware to manage and conduct scans on the device.

1.3 Location Levels

Location levels are ordered by lowest effort first (power consumption and time) and are defined by the available hardware on the device. The library automatically steps down through levels at configurable timeouts.

1.3.1 `Level 1`: Connected to Sidewalk via BLE

- **Power Consumption:** No additional power draw
- **Description:** If the device can be located via BLE over the Sidewalk network, the device notifies the cloud to resolve the location with no further effort. If the device is not connected via BLE or location cannot be resolved, the next level is attempted.

1.3.2 `Level 2`: Reserved

- **Power Consumption:** N/A
- **Description:** This level is not yet supported and will default to the next level if reached.

1.3.3 `Level 3`: Send WiFi Scan

- **Power Consumption:** Low power consumption
- **Description:** Conduct and send a WiFi scan. Minimum of 1 AP must be found in scan or next level will be attempted. If the cloud receives a low confidence score or no resolution is found, the device application must be notified to increment and try the next level.

1.3.4 `Level 4`: Send GNSS Scan

- **Power Consumption:** Higher power consumption than WiFi scanning
- **Description:** Conduct and send a GNSS scan. Minimum 4 Satellite Vehicles must be found for scan to be valid.

1.4 Architecture

The library provides a single interface for application developers to consume APIs for uplinking location data. WiFi and GNSS have corresponding Platform Abstraction Layers (PALs) that are required when these features are enabled.

Chapter 2

Usage Guide

2.1 Getting Started

2.1.1 Compilation Configuration

To enable the location library in your project, the following CMake variable definitions must be set:

```
set(SID_SDK_CONFIG_ENABLE_LOCATION 1)
set(SID_SDK_CONFIG_ENABLE_WIFI 1)
set(SID_SDK_CONFIG_ENABLE_GNSS 1)
add_compile_definitions(SID_SDK_CONFIG_ENABLE_LOCATION=1)
add_compile_definitions(SID_SDK_CONFIG_ENABLE_WIFI=1)
add_compile_definitions(SID_SDK_CONFIG_ENABLE_GNSS=1)
```

`SID_SDK_CONFIG_ENABLE_LOCATION` enables the location library as part of the build. `SID_SDK_CONFIG_ENABLE_WIFI` and `SID_SDK_CONFIG_ENABLE_GNSS` are features that can be enabled if available hardware supports WiFi and GNSS scanning.

Note: Additional configurations may be required depending on the build system of the target.

2.1.2 Library Overview

The Amazon Sidewalk Location Library provides a set of APIs for configuring, initializing, and uplinking location data to the cloud. The library follows a two-phase programming model:

2.1.2.1 Initialization

The location library must be initialized using `sid_location_init()` after `sid_init()` with the appropriate handle and location configuration.

- **handle**: A pointer to the handle returned by `sid_init()`
- **config**: Location configuration structure containing your desired location methods, effort levels, and callback function

Key configuration fields for the location config:

- **sid_location_type_mask**: Specifies which location methods to enable (BLE, WiFi, GNSS).
- **manage_effort**: Whether to allow automatic effort level management.
- **max_effort**: Maximum effort level the library can use.

- **callbacks.on_update**: Your callback function to receive location results including scan payloads in the case of WiFi or GNSS.
- **stepdowns**: Timeout configurations for effort level transitions (only applicable if WiFi or GNSS is enabled)
- **fragmentation**: Settings for message fragmentation handling. Setting timeout to 0 will result in default.

2.1.2.2 Setting Up the Location Callback

The `on_update` callback provides an asynchronous notification of location operation results. This callback must be implemented to handle the outcomes of location scanning and uplink operations.

Parameters:

- **result**: Pointer to `struct sid_location_result` containing operation results
- **context**: User-defined context pointer (optional, can be NULL)

The `sid_location_result` Structure:

The result structure contains the following fields:

- **status**: Operation status (`enum sid_location_status`)
  - `SID_LOCATION_LVL1_READY`: BLE gateway location service is available on current gateway connection after init
  - `SID_LOCATION_LVL1_UNAVAILABLE`: BLE gateway location service is not available on current gateway connection after init
  - `SID_LOCATION_SEND_DONE`: Location data uplink completed
  - `SID_LOCATION_SCAN_DONE`: Location scanning completed
- **err**: Error code indicating success or failure (`sid_error_t`)
- **mode**: The effort mode used for this operation (`enum sid_location_effort_mode`)
- **link**: The link type used for uplink (`enum sid_link_type`)
- **payload**: Scan data buffer (WiFi or GNSS scan results)
- **size**: Size of the payload data in bytes

Configuring the Callback:

```
// Define your callback function
void location_callback(const struct sid_location_result *const result, void *context)
{
    // Handle location operation results
    // Check result->status, result->err, and process result->payload if applicable
}

static struct sid_location_config cfg = {
    .sid_location_type_mask = SID_LOCATION_METHOD_ALL,
    .manage_effort = true,
    .max_effort = SID_LOCATION_EFFORT_L4,
    .callbacks = {
        .on_update = location_callback,
        .context = &app_ctx,  // Optional context pointer
    },
    // ... other configuration fields
```

```
};
```

The function returns `SID_ERROR_NONE` in case of success.

2.1.2.3 Runtime

Location operations can be executed using `sid_location_run()` after the Sidewalk loop has been started (`sid_start()`). This API initiates location scanning and/or uplink operations.

```
sid_error_t sid_location_run(struct sid_handle *handle,
                             struct sid_location_run_config *config,
                             uint32_t delay_s);
```

Parameters:

- **handle**: A pointer to the handle returned by `sid_init()`
- **config**: Struct containing the configuration for the location request
- **delay_s**: Delay in seconds to start scan (if applicable) and uplink data

Configuration fields in `struct sid_location_run_config`:

- **mode**: Effort level for this specific operation (`enum sid_location_effort_mode`)
- **type**: Operation type (`enum sid_location_run_type`):
  - `SID_LOCATION_SEND_ONLY`: Send-only operation
  - `SID_LOCATION_SCAN_ONLY`: Scan-only operation
  - `SID_LOCATION_SCAN_AND_SEND`: Combined scan and send operation
- **buffer**: Pointer to buffer for location data (uint8_t *)
- **size**: Size of the buffer (size_t)

The function returns `SID_ERROR_NONE` in case of success. If effort mode is NOT managed, `SID_LOCATION_EFFORT_DEFAULT` cannot be used for the mode.

2.1.2.4 De-Initialization

The location library must be deinitialized using `sid_location_deinit()` before calling `sid_deinit()`.

The function takes the following parameter:

- **handle**: A pointer to the handle returned by `sid_init()`

The function returns `SID_ERROR_NONE` in case of success.

For detailed API documentation including function signatures, parameters, return values, and other available apis, please refer to the Amazon Sidewalk API Developer Guide.

2.2 WiFi and GNSS Location Example (LR1110)

The following example shows how to enable GNSS and WiFi with the location library for use with the LR1110 shield.

 ©2025 Amazon Technologies, Inc.

2.2.1 Example Platform Configuration

```
// file: app_location_lr11xx_config.h
#include <lr11xx_gnss_wifi_config.h>
const lr11xx_gnss_wifi_config_t *get_location_cfg(void);
```

The GNSS configuration supports two key parameters:

Constellation Type:

- `SMTC_MODEM_GNSS_CONSTELLATION_GPS` - Use GPS only constellation
- `SMTC_MODEM_GNSS_CONSTELLATION_BEIDOU` - Use BEIDOU only constellation
- `SMTC_MODEM_GNSS_CONSTELLATION_GPS_BEIDOU` - Use both GPS and BEIDOU constellations

Scan Mode:

- `SMTC_MODEM_GNSS_MODE_STATIC` - Scanning mode for non-moving objects
- `SMTC_MODEM_GNSS_MODE_MOBILE` - Scanning mode for moving objects

```
// file: app_location_lr11xx_config.c
static lr11xx_gnss_wifi_config_t gnss_wifi_config = {
    .constellation_type = SMTC_MODEM_GNSS_CONSTELLATION_GPS_BEIDOU,
    .scan_mode = SMTC_MODEM_GNSS_MODE_MOBILE
};

const lr11xx_gnss_wifi_config_t *get_location_cfg(void)
{
    return &gnss_wifi_config;
}

// Platform parameters setup
platform_parameters_t platform_parameters = {
    .mfg_store_region = sid_mfg_config_get(),
    .platform_init_parameters.radio_cfg =
        (radio_lr11xx_device_config_t *)get_radio_cfg(),
    .platform_init_parameters.gnss_wifi_cfg =
        (lr11xx_gnss_wifi_config_t *)get_location_cfg(),
};
```

2.2.2 Location Library Usage

```
// Create a location config with user supplied callback
void location_callback(const struct sid_location_result *const result, void *context)
{
    // User can pull the payload for WiFi or GNSS in result->payload
    // Handle different location methods based on result->mode
}

static struct sid_location_config cfg = {
    .sid_location_type_mask = SID_LOCATION_METHOD_ALL,
    .manage_effort = true,
    .max_effort = SID_LOCATION_EFFORT_L4,
    .callbacks = {
        .on_update = location_callback,
    },
```

```
        .stepdowns = { // Timeout(ms) to use before stepping down location level when using
        .l2_to_l1 = 0,
        .l3_to_l2 = 0,
        .l4_to_l3 = 0,
    },
    .fragmentation = { // Fragmentation configurations to use when for location payload upl
        .timeout_ms = 0, // 0 indicates use default timeout value
        .max_retries = 1,
    }
};

// Initialize after sid_init()
sid_error_t res = sid_location_init(*app_context->sidewalk_handle, &cfg);

// WiFi scan and send uplink
struct sid_location_run_config config = {
    .type = SID_LOCATION_SCAN_AND_SEND,
    .mode = SID_LOCATION_EFFORT_L3
};
sid_error_t res = sid_location_run(*app_context->sidewalk_handle, &config, 0);

// GNSS scan and send uplink
config.mode = SID_LOCATION_EFFORT_L4;
sid_error_t res = sid_location_run(*app_context->sidewalk_handle, &config, 0);
```

2.2.3 Fragmentation Timeout Configuration

If configuring the fragmentation timeout, the `timeout_ms` parameter should be considered carefully. The current cloud implementation calculates the timeout based on the total message fragmentation requirements and link type characteristics.

The timeout will be calculated automatically if set to 0 and follows the following formula that mirrors the cloud algorithm to wait/reassemble fragments:

$$\texttt{timeout} = (\texttt{total\_size} / \texttt{MTU}) \times \texttt{link\_type\_duration} \times 2$$

Where:

- `total_size / MTU` determines the number of fragments required
- `link_type_duration` is the maximum time the specified link type can take to uplink one fragment
- The factor of 2 provides additional margin for retries to reach the cloud.

Current supported link type durations:

- **LoRa**: Up to 5 seconds per fragment
- **BLE**: BLE does not utilize fragmentation for location as the MTU is large enough. These configurations will not apply

For example, a message requiring 4 fragments over LoRa would be: $4 \times 5000 \times 2 = 40000$ milliseconds timeout at the cloud. Consider setting `timeout_ms` appropriately based on your expected use case. Using a timeout independent from the cloud fragmentation timeout may be desirable based on the use case.

2.3 BLE-Only Location Example

The following example shows how to use the location library with just Sidewalk Network Location over BLE.

 ©2025 Amazon Technologies, Inc.

```
// Create a location config with user supplied callback
void location_callback(const struct sid_location_result *const result, void *context)
{
    // User can get the results of the uplink here
    // Handle the location result based on status
}

static struct sid_location_config cfg = {
    .sid_location_type_mask = SID_LOCATION_METHOD_BLE_GATEWAY,
    .manage_effort = true,
    .max_effort = SID_LOCATION_EFFORT_L1,
    .callbacks = {
        .on_update = location_callback,
    },
    .stepdowns = { // Stepdowns are not applicable in the BLE-Only mode.
        .l2_to_l1 = 0,
        .l3_to_l2 = 0,
        .l4_to_l3 = 0,
    },
    .fragmentation = { // Fragmentation is not applicable in the BLE-Only mode.
        .timeout_ms = 0,
        .max_retries = 0,
    }
};

// Initialize after sid_init()
sid_error_t res = sid_location_init(*app_context->sidewalk_handle, &cfg);

// Send location request after sid_start()
// (requires BLE connection to opted-in gateway)
struct sid_location_run_config config = {
    .type = SID_LOCATION_SCAN_AND_SEND,
    .mode = SID_LOCATION_EFFORT_DEFAULT
};
sid_error_t res = sid_location_run(*app_context->sidewalk_handle, &config, 0);
```

2.4 Platform Abstraction Layers

If the device supports WiFi and GNSS scanning (levels 3 and 4), the WiFi and GNSS Platform Abstraction Layers (PALs) must be implemented and linked. Reference implementations are provided that utilize the Semtech LoRa Basics Modem middleware APIs.

The implementation available in the Sidewalk MCU SDK combines both WiFi and GNSS PALs as they interface with the LoRa Basics Modem and utilize shared resources.

2.5 Best Practices

- Use `sid_location_set_max_mode()` to limit power consumption during low-power modes
- Configure appropriate stepdown timeouts based on your application requirements
- Always check return values from location API functions
- Handle different status codes in your callback function appropriately

- Allow the library to manage effort levels automatically when possible
- Consider using scan-only operations when immediate uplink is not required

 ©2025 Amazon Technologies, Inc.

Glossary

GNSS	Global Navigation Satellite System used for satellite-based location determination.
Location Level	Hierarchical effort modes for location resolution, ordered by power consumption and time requirements.
LR1110	Semtech transceiver with integrated GNSS and WiFi scanning capabilities.
MTU	Maximum Transmission Unit.
PAL	Platform Abstraction Layer.