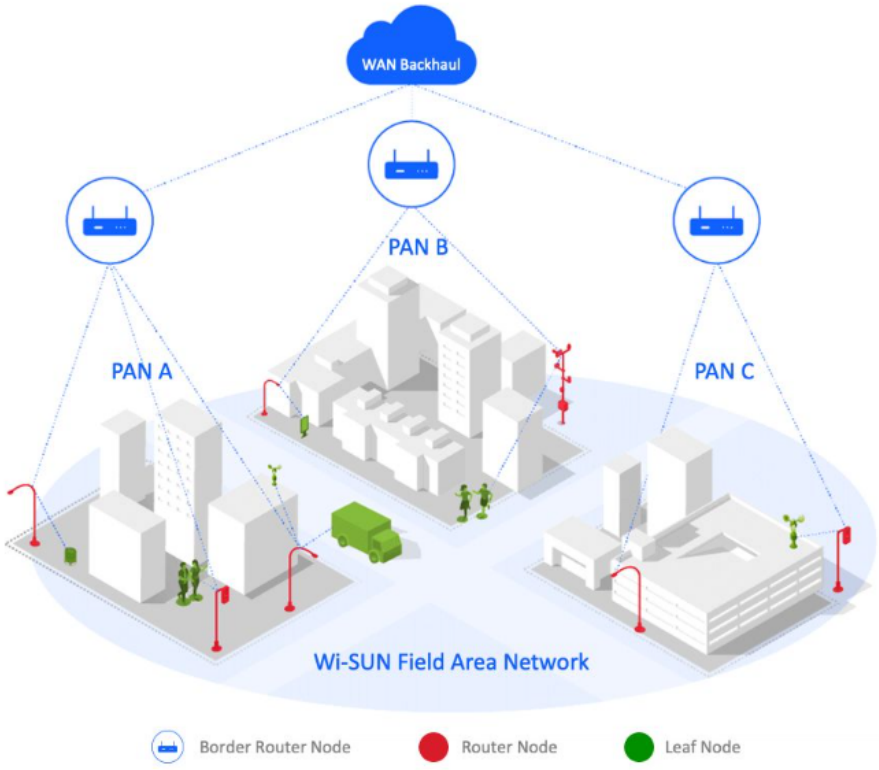
Wi-SUN

Developing with Wi-SUN

Developing with Silicon Labs Wi-SUN

Wireless Smart Ubiquitous Network (Wi-SUN) is the leading IPv6 sub-GHz mesh technology for smart city and smart utility applications. Silicon Labs' Wi-SUN SDK includes industry-leading software stacks and development tools for Wi-SUN end nodes and border routers. In conjunction with SoCs and reference applications for Wi-SUN, developers can use software and tools from Silicon Labs to quickly and reliably:

- Develop multi-node mesh networks
- Monitor and debug multiple nodes simultaneously
- Visually analyze system performance



The content on these pages is intended for those are developing a Wi-SUN application using the Silicon Labs Wi-SUN SDK.

For details about this release: Links to release notes are available on the silabs.com Gecko SDK page.

For Silicon Labs' Wi-SUN product information: See the product pages on silabs.com.

For background about the Wi-SUN protocol: The introductory materials on silabs.com is a good place to start.

To get started with development with the Wi-SUN stack: See the Getting Started page to get started working with sample applications.

If you are already in development with the stack: See the specific material on this page for details or go directly to the API Reference.

If you are primarily interested in working directly with the Wi-SUN PHY and system, without using the Silicon Labs stack, see the Wi-SUN PHY section.

Introduction

Introducing Wi-SUN Development

Wireless Smart Ubiquitous Network (Wi-SUN) is the leading IPv6 sub-GHz mesh technology for smart city and smart utility applications. Wi-SUN brings Smart Ubiquitous Networks to service providers, utilities, municipalities/local government, and other enterprises, by enabling interoperable, multi-service, and secure wireless mesh networks. Wi-SUN can be used for large-scale outdoor IoT wireless communication networks in a wide range of applications covering both line-powered and battery-powered nodes.

Silicon Labs provides a complete set of hardware and software solutions to help developers design their Wi-SUN wireless products:

- Certified hardware platforms
- Certified Wi-SUN FAN router stack
- Certified Wi-SUN FAN border router solution (GitHub)

Silicon Labs has enhanced Wi-SUN to work with Silicon Labs hardware. The Wi-SUN stack library is available as a software development kit (SDK) installed as part of the Gecko SDK Suite (GSDK) of Silicon Labs SDKs. The two primary elements to getting started with Wi-SUN development are the **Silicon Labs Wi-SUN SDK** and **Simplicity Studio 5**.

The Silicon Labs Wi-SUN SDK

The Silicon Labs Wi-SUN SDK is composed of the Wi-SUN FAN stack and sample applications as well as the addition of metadata to allow for the seamless integration into Simplicity Studio 5. The Silicon Labs Wi-SUN SDK contains the Wi-SUN stack in a library format.

The Silicon Labs Wi-SUN SDK is based on the Gecko Platform component-based design, where each component provides a specific function. Components are made up of a collection of source files and properties. The component-based design enables customization by adding, configuring, and removing components. The application developer can use SSv5's Project Configurator and Component Editor to easily assemble the desired features by including those components that match the required functionality and by configuring the various properties associated with those components.

For details on the Wi-SUN stack version included within the Silicon Labs Wi-SUN SDK, refer to the Wi-SUN SDK release notes.

Simplicity Studio 5 (SSv5)

The Silicon Labs Wi-SUN SDK is downloaded through SSv5. SSv5 is the core development environment designed to support the Silicon Labs IoT portfolio of system-on-chips (SoCs) and modules. It provides access to target device-specific web and SDK resources; software and hardware configuration tools; an integrated development environment (IDE) featuring industry-standard code editors, compilers and debuggers; and advanced, value-add tools for network analysis and code-correlated energy profiling.

SSv5 is designed to simplify developer workflow. It intelligently recognizes all Silicon Labs evaluation and development kit parts and, based on the selected development target, presents appropriate software development kits (SDKs) and other development resources.

The GNU Compiler Collection (GCC) is provided with SSv5. Other important development tools provided with SSv5 are reviewed in Development Tools.

Starting Development

Prerequisites

Before following the procedures in this document you must have:

- Purchased one of the Wireless Gecko (EFR32) Portfolio Wireless Kits with compatible radio boards, listed on https://www.silabs.com/wireless/wi-sun.
- Downloaded SSv5 and the Gecko SDK and be generally familiar with the SSv5 Launcher perspective. SSv5 installation and getting started instructions along with a set of detailed references can be found in the online Simplicity Studio 5 User's Guide.
- Obtained a compatible compiler (See the Wi-SUN SDK's release notes for the compatible versions):
  - Simplicity Studio comes with a free GCC C-compiler.
  - IAR Embedded Workbench for ARM (IAR-EWARM) can also be used as the compiler for Silicon Labs Wi-SUN projects. Once IAR-EWARM is installed, the next time Simplicity Studio starts it will automatically detect and configure the IDE to use IAR-EWARM. Refer to the IAR-IDE License section in the Development Tools page to get a 30-day IAR IDE trial.

Setup

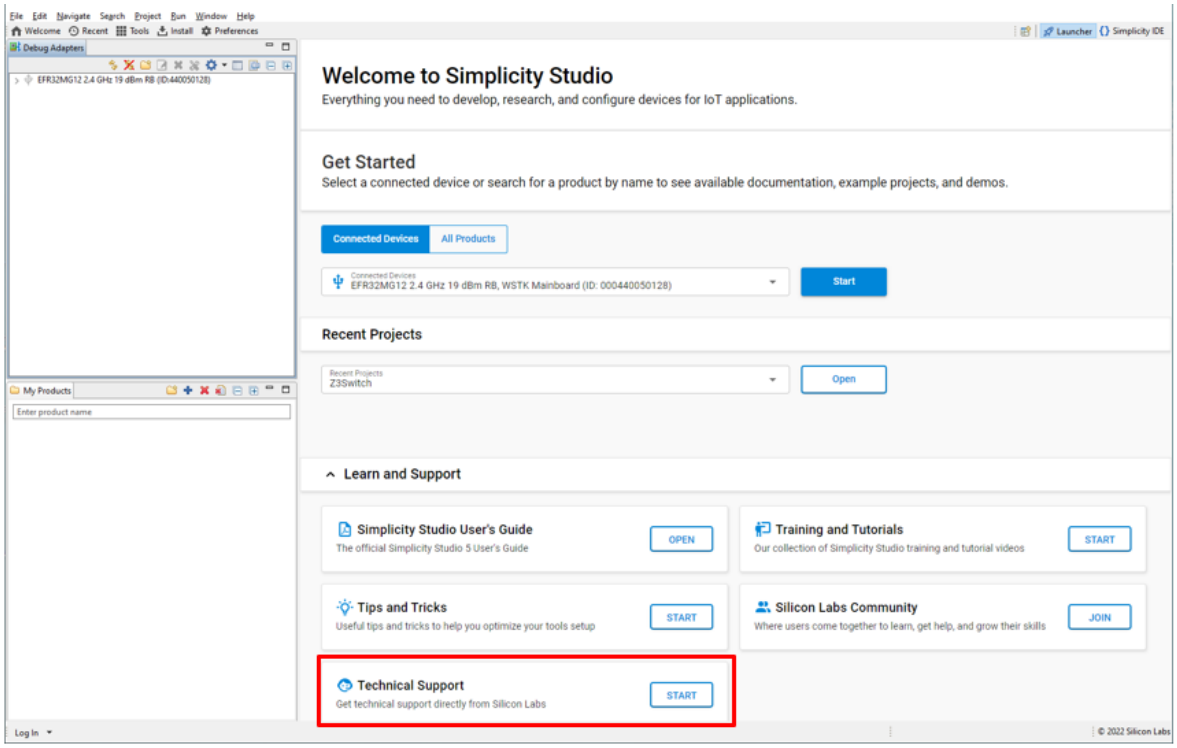
Install Simplicity Studio 5 (SSv5), which will set up your development environment and walk you through GSDK installation. Installation instructions are provided in the Simplicity Studio 5 online User's Guide. This is the recommended method for getting started with development.

Alternatively, Gecko SDK may be installed manually by downloading or cloning the latest from GitHub. See https://github.com/siliconlabs/gecko_sdk for more information.

Using Simplicity Studio, you have easy access to the sample applications and demos. Instructions for quickly beginning to use these to create a Wi-SUN network is included in Getting Started with Application Development.

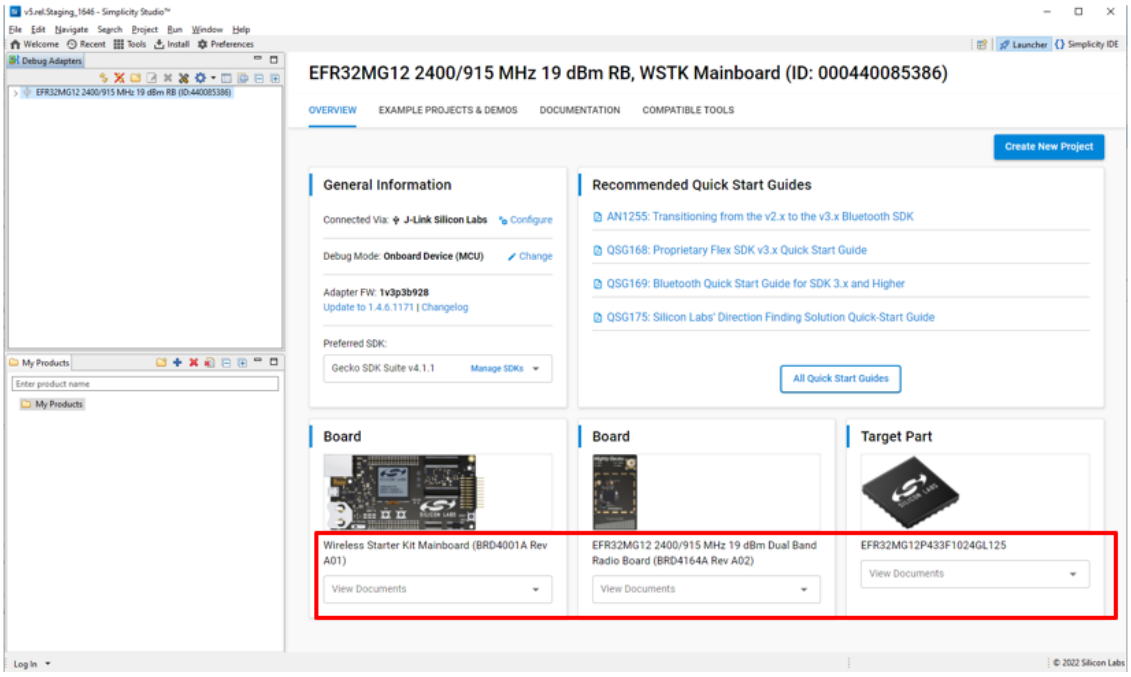
Support

Access the Silicon Labs support portal at https://www.silabs.com/support through SSv5's Welcome view under **Learn and Support**. Use the support portal to contact Customer Support for any questions you might have during the development process.

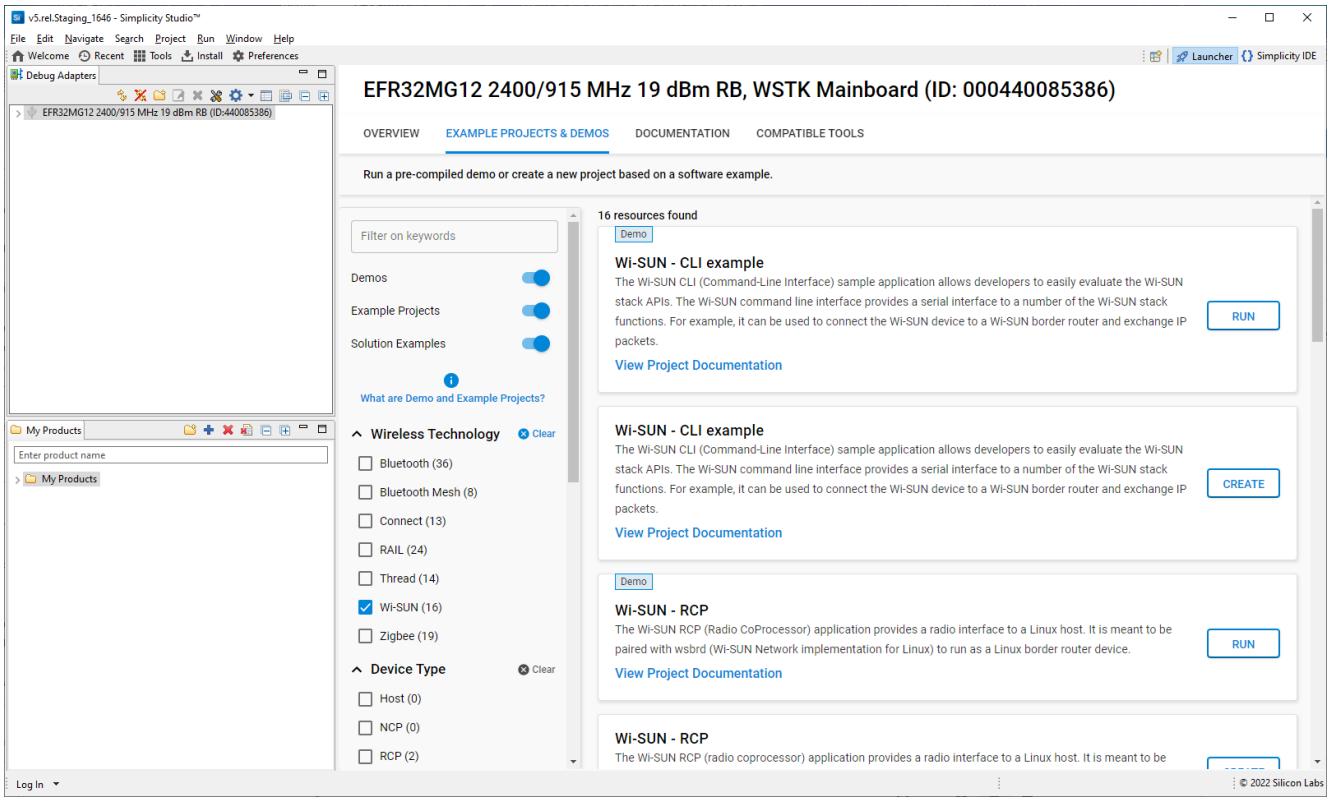


Documentation

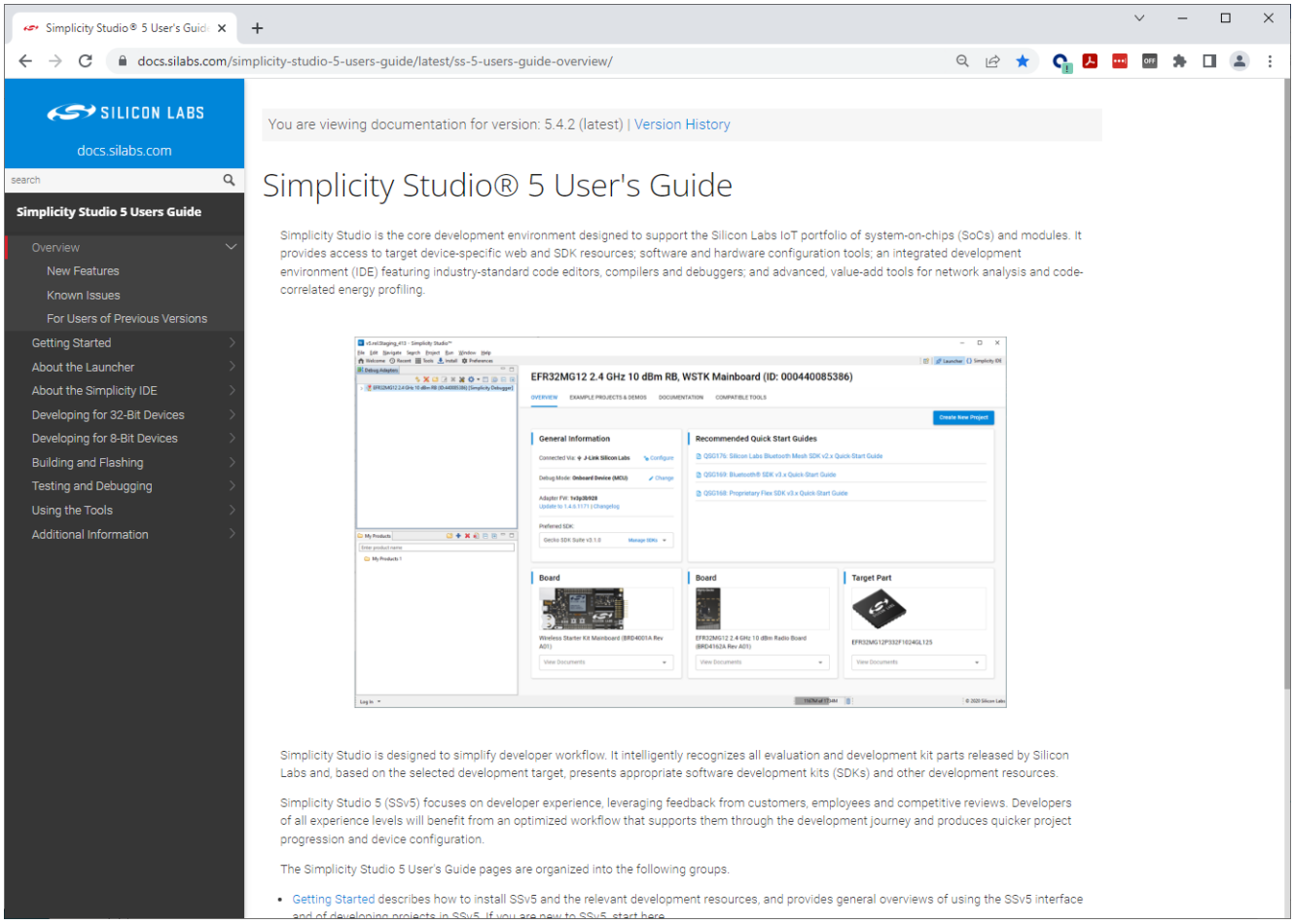
Relevant documentation is available through SSv5. It is filtered based on the device selected in either the Debug Adapters view or the My Product view. Hardware-specific documentation for the device can be accessed through links on the **OVERVIEW** tab.



SDK documentation and other references are available through the **DOCUMENTATION** tab. Filter with the **Wi-SUN Technology Type** checkbox to see documentation most closely related to the Wi-SUN SDK. To see documents specific to Wi-SUN, select **Wi-SUN** under **Wireless Technology**.



SSv5 and its tools are documented in the online Simplicity Studio 5 User's Guide.



Getting Started

Getting Started with Wi-SUN Application Development

This section assumes that you have purchased your hardware and have downloaded SSv5 and the Silicon Labs Wi-SUN SDK as described in Prerequisites. You should also be familiar with the features of the SSv5 Launcher perspective as documented in the Simplicity Studio 5 online User's Guide.

Content in this section includes:

- **Wi-SUN Sample Applications**: Introduces the list of Wi-SUN Demos and Sample applications to start creating a Wi-SUN Network and evaluate the Silicon Labs Wi-SUN solution.
- **Creating a Wi-SUN Network**: Provides instructions to compile and load a simple **Wi-SUN Ping** application on a node and a **Wi-SUN Border Router** demo image on another node. You can then use these to create a network.

Wi-SUN Sample Applications

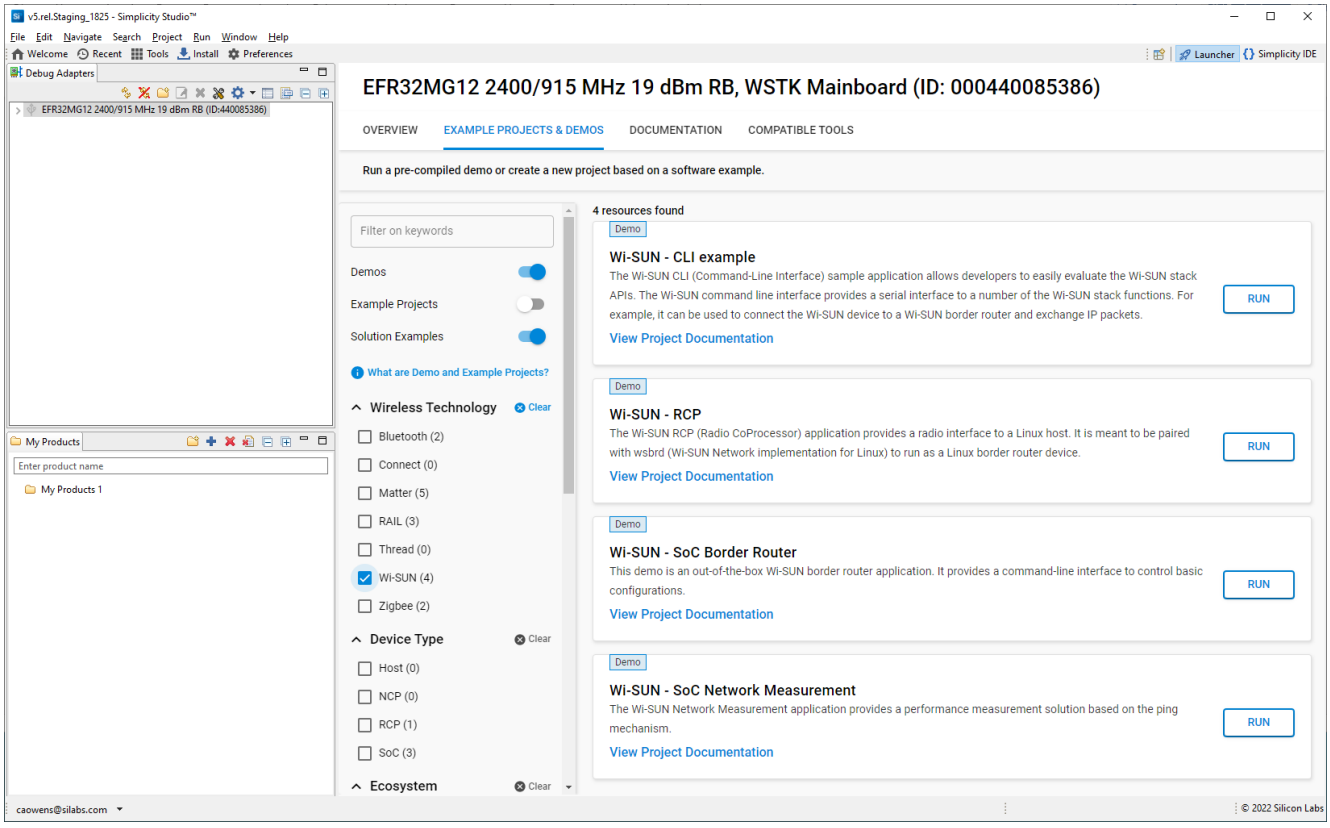
About Sample Applications and Demos

Because starting application development from scratch is difficult, the Silicon Labs Wi-SUN SDK comes with a number of built-in sample applications and demos covering the most frequent use cases designed to illustrate common application functions. Silicon Labs strongly recommends starting development from one of the sample applications.

Like everything in SSv5, the examples and the demos shown on the **EXAMPLE PROJECTS & DEMOS** tab are filtered based on the part you have connected or selected.

Demos

Demos are prebuilt firmware images that are ready to download to a compatible device. The quickest way to find if a demo is available for your part is by adding the part or board information in the My Products view and then navigating to the **EXAMPLE PROJECTS & DEMOS** tab in the **Launcher** perspective. Disable the Example Projects filter. The Solution Examples filter is provided for future use.



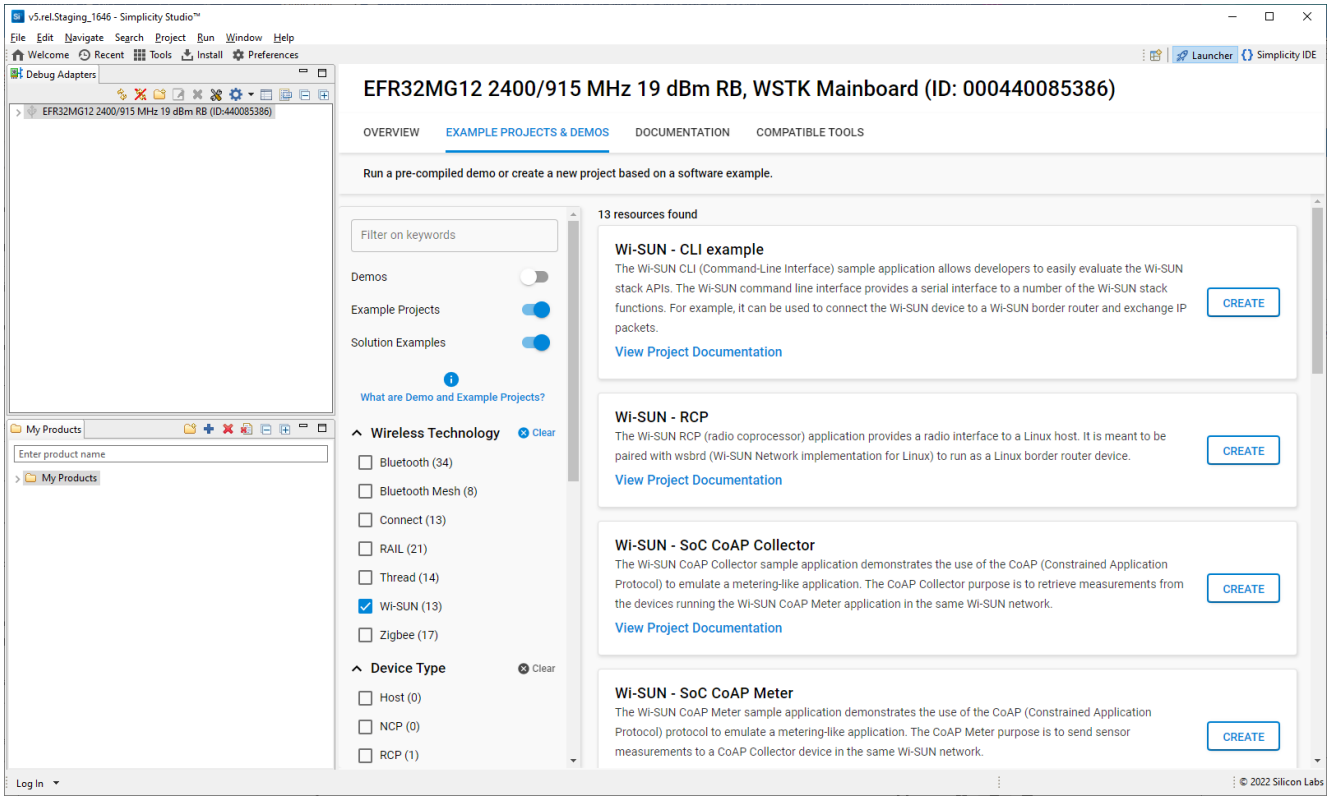
Precompiled demo application images provided with the Wi-SUN SDK are compatible with the Wi-SUN Wireless SoCs listed Silicon labs web page.

Software Examples

When you work with examples, the process is:

- Select an example
- Change the example configuration (if needed)
- Build the example
- Load the example to the target device

Since typically you will finish by flashing a compiled application image to a device, connect a device to your computer and select it in the Debug Adapters view. In the **EXAMPLE PROJECTS & DEMOS** tab on the **Launcher** perspective, enter 'wi-sun' as a keyword. A number of other filters are provided. To see the examples only, turn off **Demos**.



Each example has its own documentation accessible by clicking **View Project Documentation** below the example description. The HTML document covers the steps to set up the example and run the associated demonstration.

The sample applications provided with the Silicon Labs Wi-SUN SDK are as follows.

Wi-SUN – CLI example: Acts as a Wi-SUN router node in a network, and provides an interface device functionality.

Wi-SUN – SoC CoAP Collector: Collects data from other devices configured as meters using CoAP.

Wi-SUN – SoC CoAP Meter: Provides basic meter functionality to communicate with a collector using CoAP.

Wi-SUN – SoC Empty: Provides a basic framework to begin adding custom functionality.

Wi-SUN – SoC Collector: Collects data from other devices configured as meters.

Wi-SUN – SoC Network Measurement: Provides a tool to measure the Wi-SUN solution performance.

Wi-SUN – SoC Meter: Provides basic meter functionality to communicate with a collector.

Wi-SUN – SoC Ping: Provides simple connectivity testing.

Wi-SUN – SoC TCP Client: Works with the TCP server example using the TCP protocol.

Wi-SUN – SoC TCP Server: Works with the TCP client example using the TCP protocol.

Wi-SUN – SoC UDP Client: Works with the UDP server example using the UDP protocol.

Wi-SUN – SoC UDP Server: Works with the UDP client example using the UDP protocol.

Wi-SUN – RCP: Radio co-processor border router implementation that pairs with a Linux host running the Wi-SUN stack upper layers.

Create a Wi-SUN Network

Creating a Wi-SUN Network

In these instructions, you will compile and load a simple **Wi-SUN Ping** application on a node and a **Wi-SUN Border Router** demo image on another node. The Creating a Network section describes how to use the examples to create a network. The Network Analyzer section on the Going Further page describes how to use Network Analyzer to observe traffic across the network.

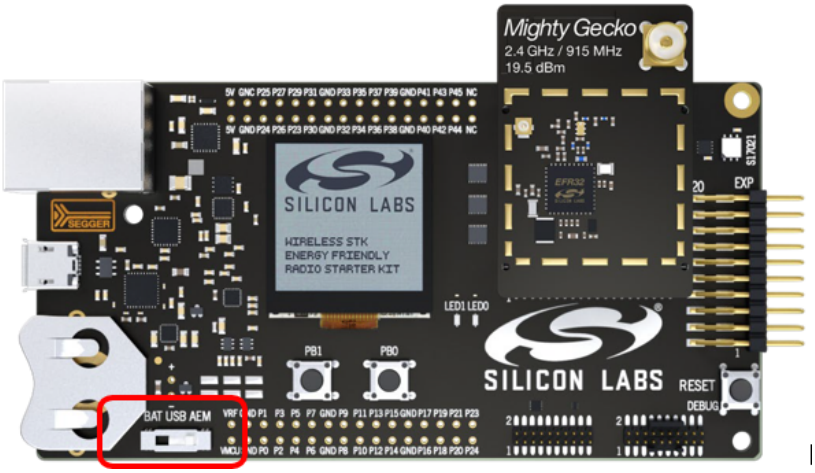
When working with an example application in Simplicity Studio, you will be executing the steps in the following order:

1. Create a project based on an example.
2. Configure the project.
3. Build the application image and flash it to your device.

These steps are described in detail in the following sections. These procedures are illustrated for a mainboard with an EFR32MG12. Note: Your SDK version may be later than the version shown in the figures.

You should have your mainboard connected.

Note: For best performance in Simplicity Studio 5, be sure that the power switch on your mainboard is in the Advanced Energy Monitoring or "AEM" position, as shown in the following figure.



Flashing the Wi-SUN Border Router

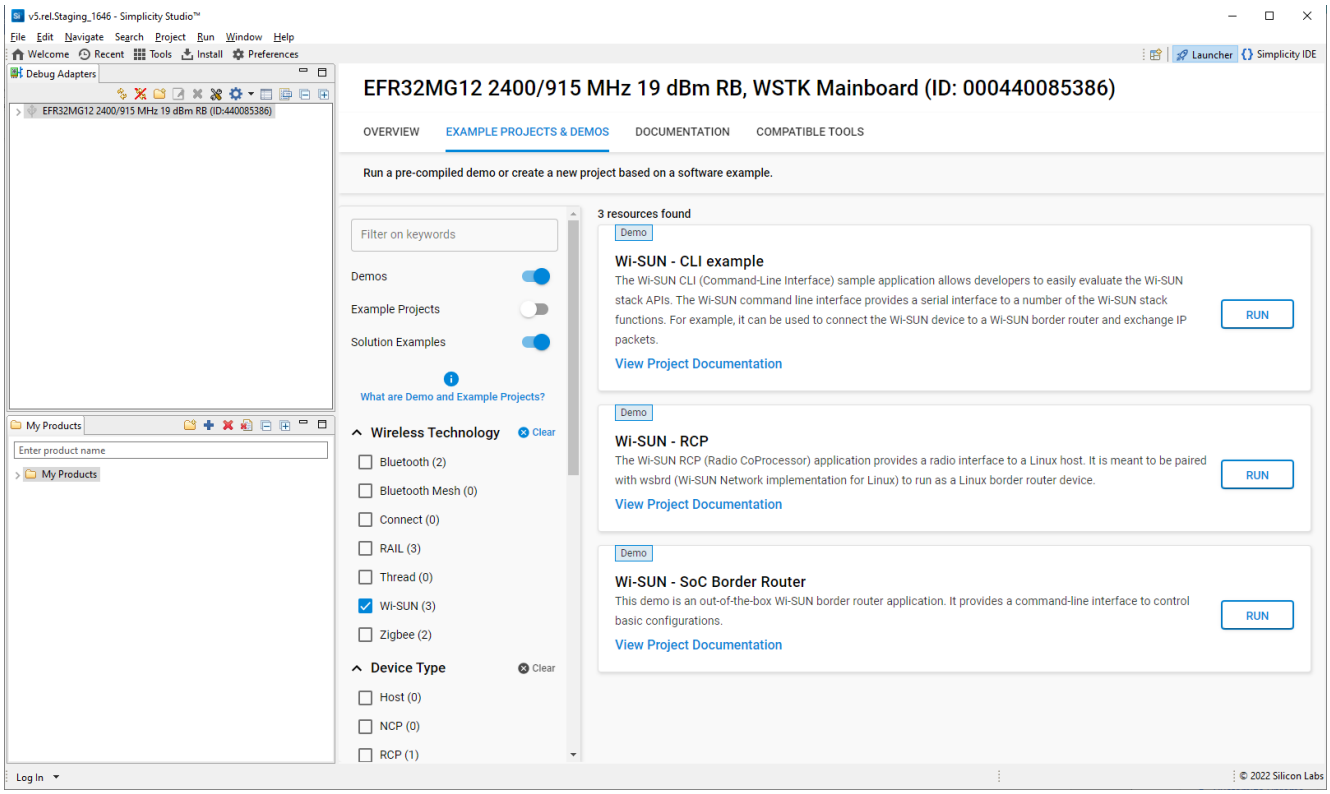
Every Wi-SUN example requires a Wi-SUN Border Router to create and manage a Wi-SUN network for the Wi-SUN devices to join. It provides an easy and quick medium to evaluate the Silicon Labs Wi-SUN stack solution without deploying an expensive and cumbersome production-grade Wi-SUN Border Router. A CLI (Command-Line Interface) is exposed to facilitate the configuration.

Note: Make sure your mainboard has the latest "Adapter FW" to avoid any issue when using the example CLI. To do so:

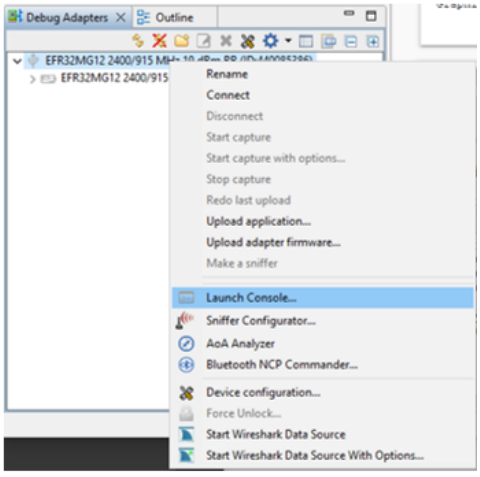
1. In the **Launcher** perspective, click the radio board listed in the Debug Adapters view.
2. In the **OVERVIEW** tab, verify the Adapter FW version in the **General Information** card.
3. If Simplicity Studio 5 proposes to update the firmware, do so.

The Wi-SUN Border Router demonstration is delivered only in a binary format. The implementation does not scale for a production-grade Border Router maintaining several thousand Wi-SUN nodes. For a production-grade border router solution,

refer to the Wi-SUN Border Router section to get started.



1. In the **Debug Adapters** view, select the device that will be the Wi-SUN border router. Note: To rename the device so that you know which device is the Border Router, right-click it and select **Rename** on the context menu.
2. Navigate to the **EXAMPLE PROJECTS & DEMOS** tab and turn off the **Example Projects** filter. Click **RUN** next to **Wi-SUN – SoC Border Router** demo.
3. Start the Wi-SUN Border Router with the CLI interface. In the Debug Adapters view, right-click the Border Router device and click Launch Console, as shown. Alternatively, click Tools in the Simplicity IDE menu and select 'Device Console'.



4. To get a prompt on the Console, go to the Serial 1 tab and press **Enter**. To start the Wi-SUN Border Router with default FAN 1.1 PHY, enter:

```
> wisun start_fan11
```

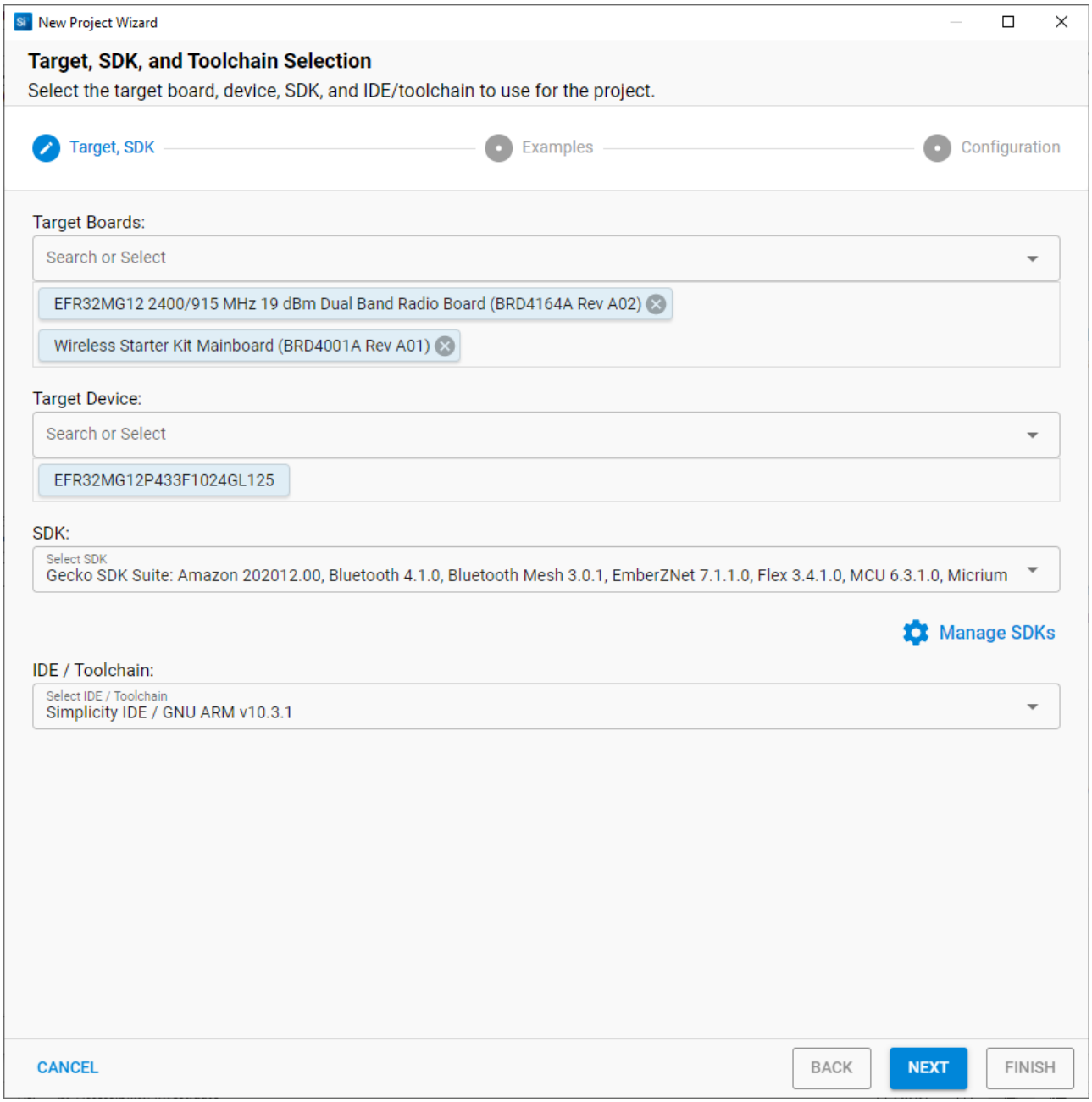
Or with default FAN 1.0 PHY using the following command:

```
> wisun start_fan10
```

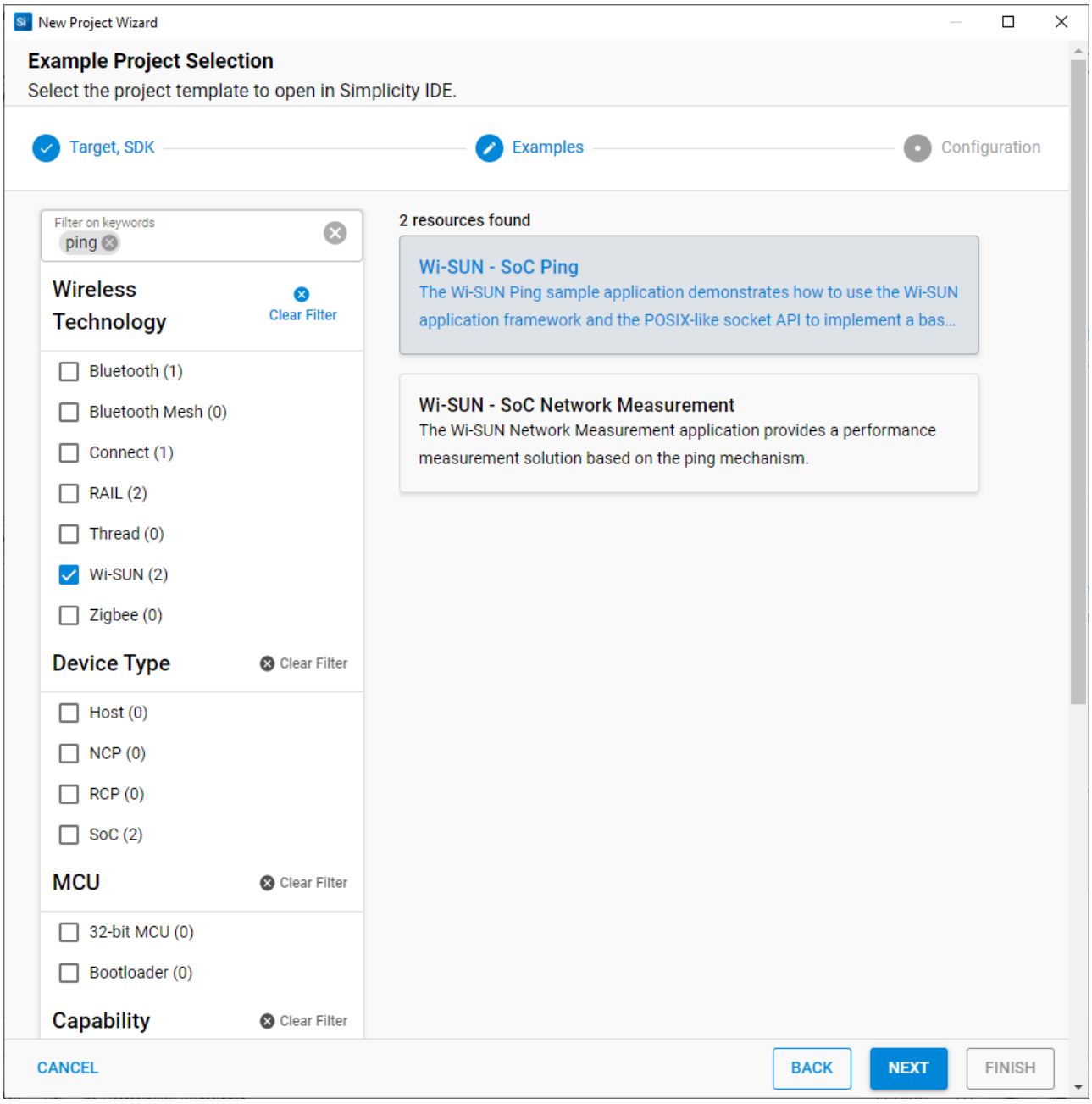
Creating a Project Based on an Example

Simplicity Studio 5 (SSv5) offers a variety of ways to begin a project using an example application. The online Simplicity Studio 5 User's Guide describes them all. This guide uses the **File > New > Silicon Labs Project Wizard** method because it takes you through all three of the Project Creation Dialogs. Details on each creation dialog option may be found in the *Simplicity Studio 5 User's Guide*.

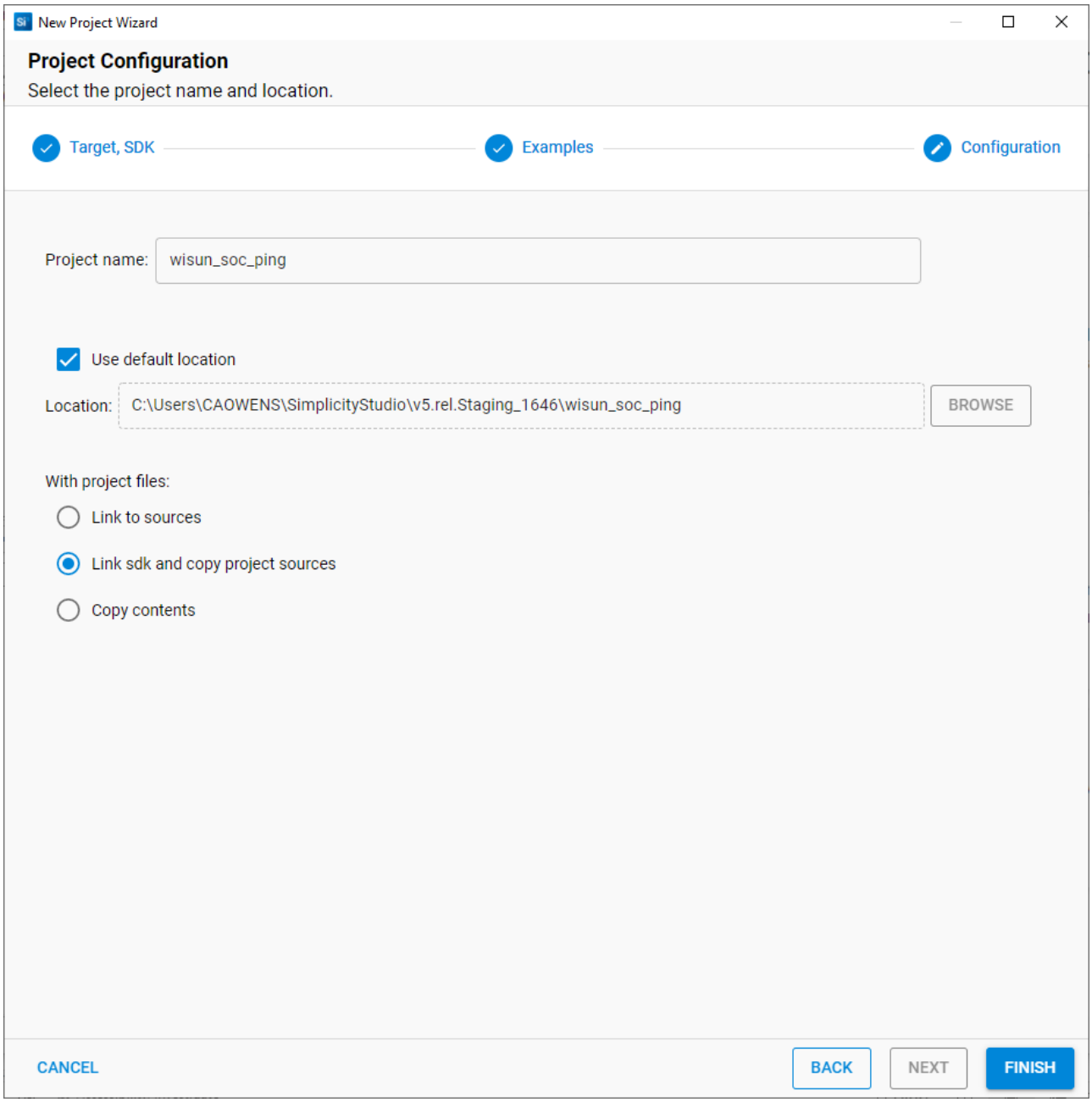
1. In the **Debug Adapters** view, select the target part for the application node. This should be a different part than the one used for the border router in the previous section.
2. Open SSv5's File menu and select **New > Silicon Labs Project Wizard**. The Target, SDK, and Toolchain Selection dialog opens. Do not change the default **Simplicity IDE / GNU ARM v<version>** toolchain supported by Wi-SUN. Click **NEXT**.



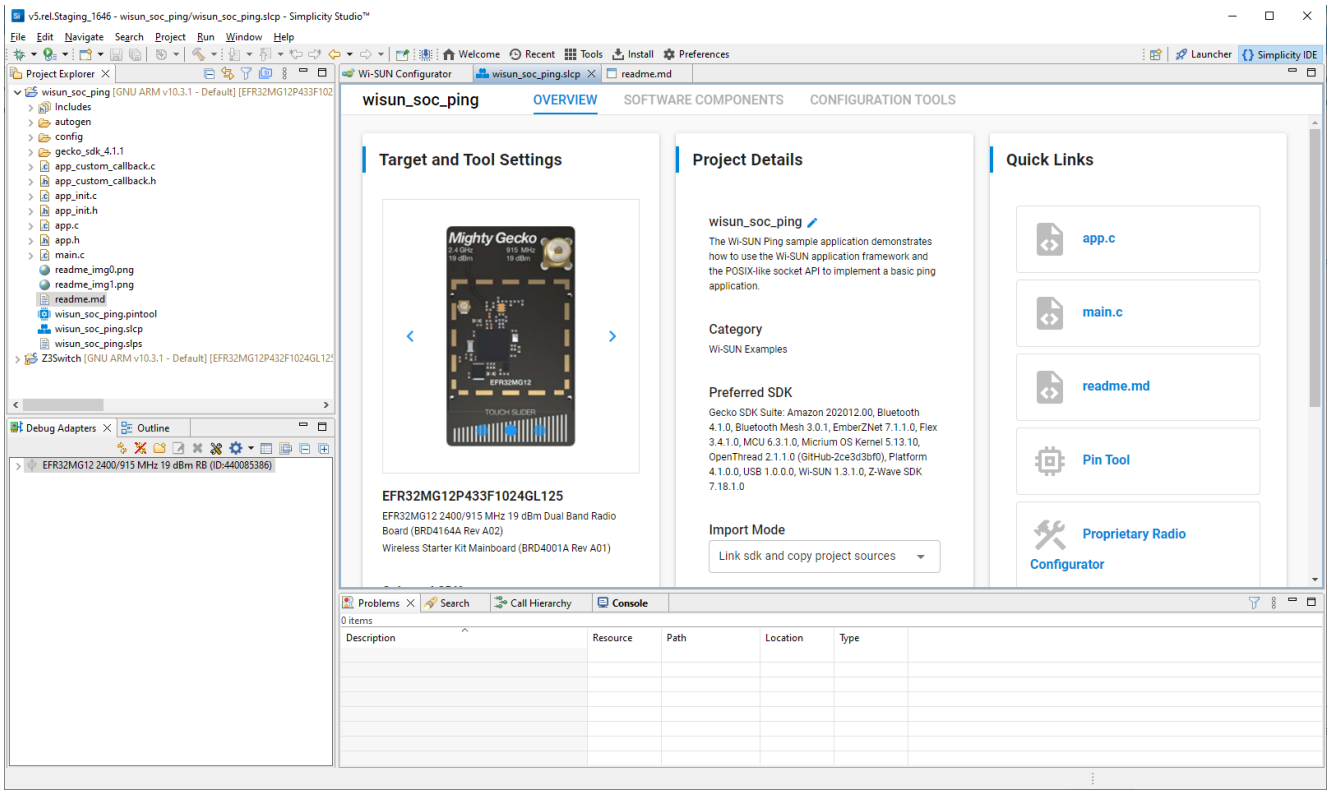
3. The Example Project Selection dialog opens. Use the 'Wi-SUN' Technology Type and Keyword filters to search for a specific example, in this case **Wi-SUN – SoC Ping**. Select it and click **NEXT**.



4. The Project Configuration dialog opens. Here you can rename your project, change the default project file location, and determine if you will link to or copy project files. Note that if you change any linked resource, it is changed for any other project that references it. Click **FINISH.**

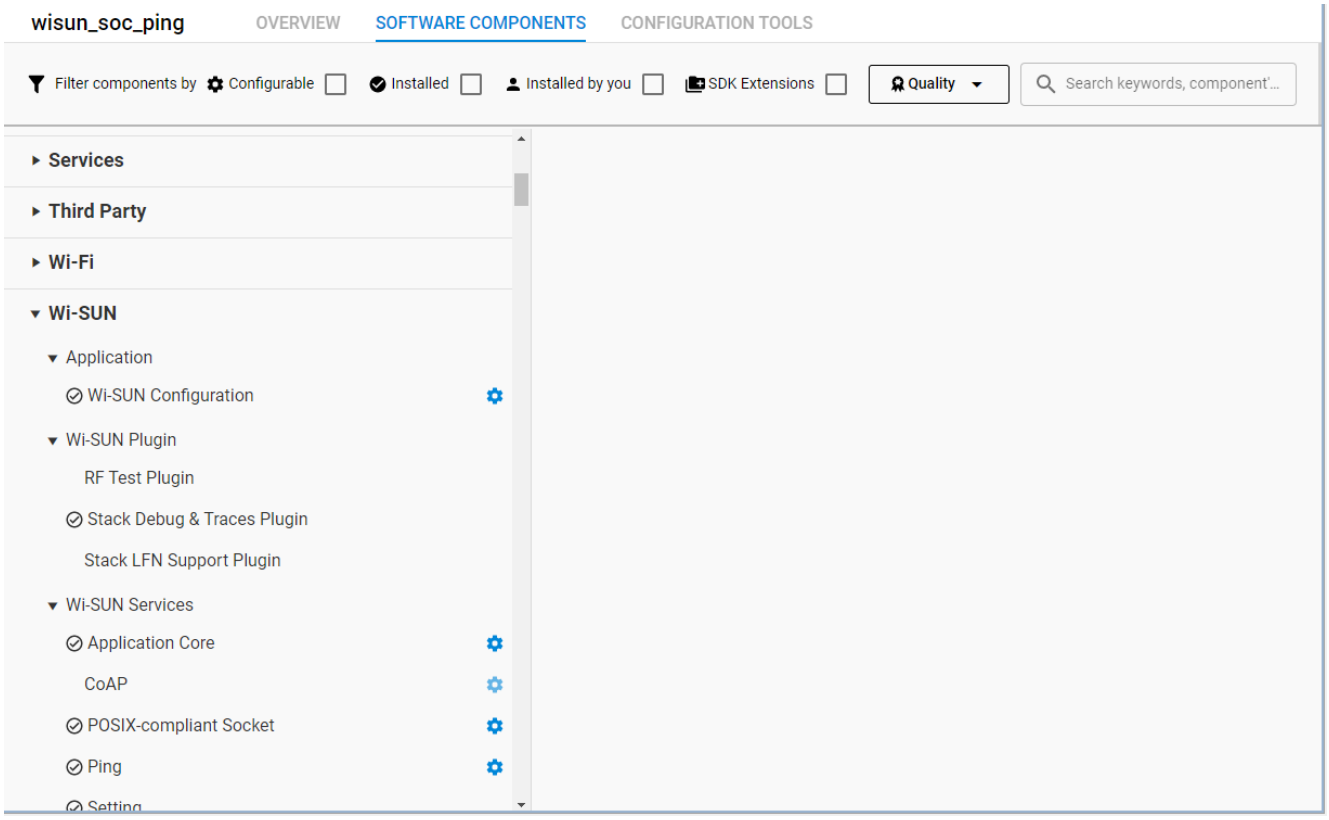


5. The Simplicity IDE Perspective opens with the project documentation (readme.md). Click the <project>.slcp tab to see the Project Configurator OVERVIEW tab. See the online Simplicity Studio 5 User's Guide for details about the functionality available through the Simplicity IDE perspective and the Project Configurator.



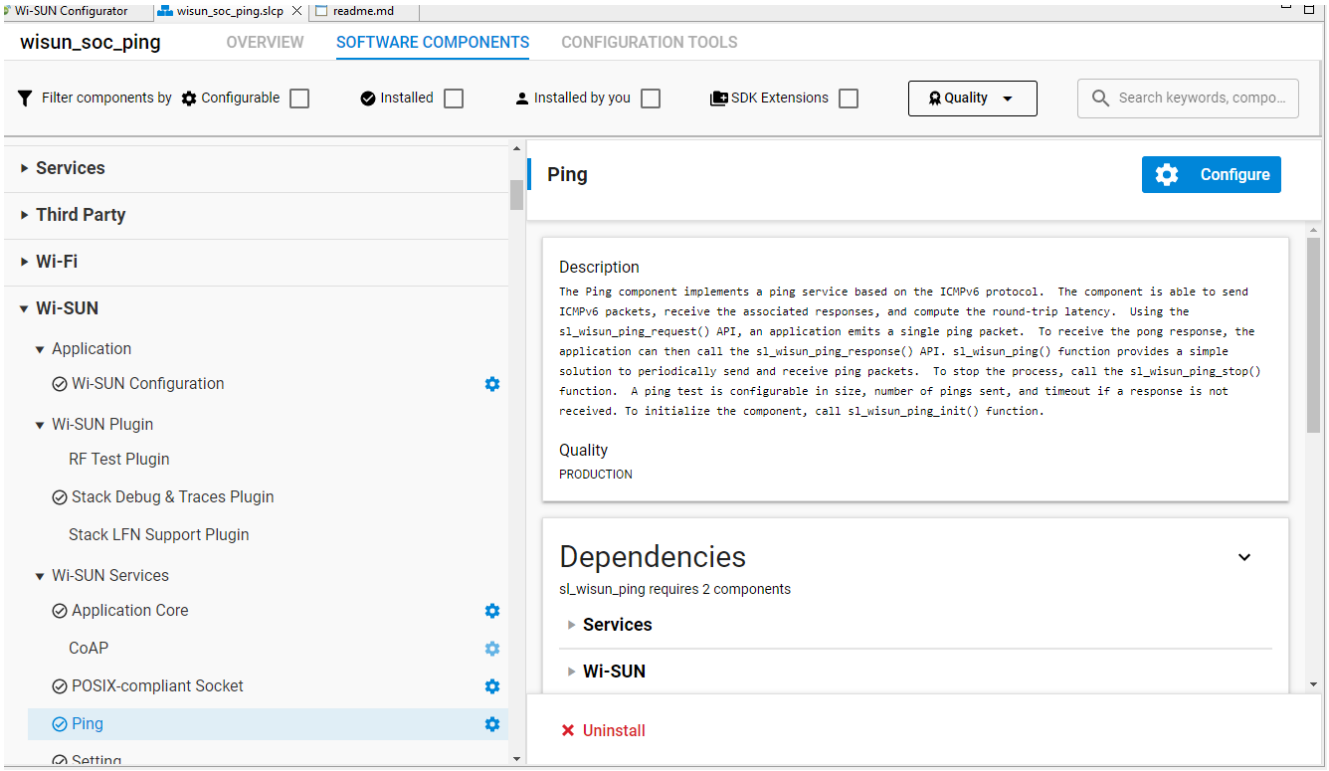
Configuring the Project

Silicon Labs Wi-SUN applications are built on a Gecko Platform component structure. Click the SOFTWARE COMPONENTS tab to see a complete list of component categories.



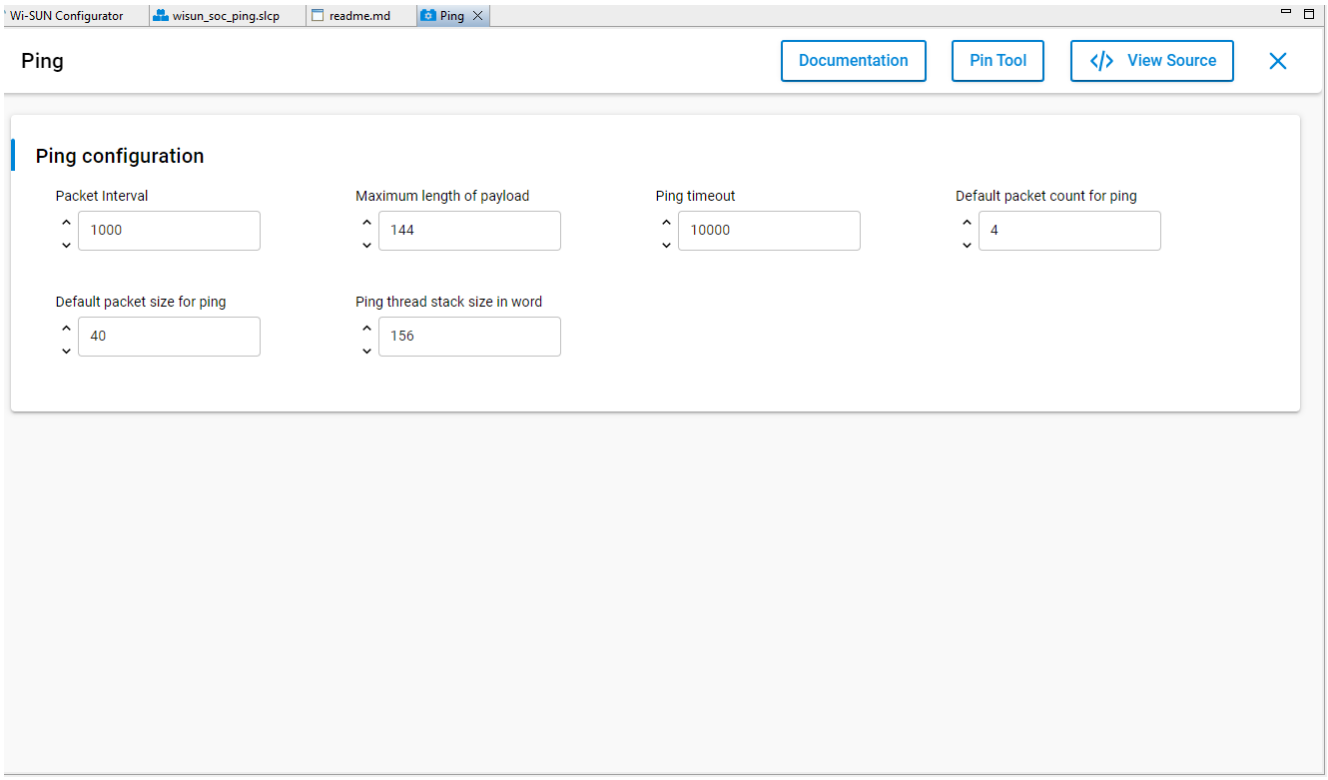
The project is configured by installing and uninstalling components and configuring installed components. Installed components are shown with a circled checkmark on their left. Click **Installed Components** to see a filtered list of components installed by the example application.

Configurable components have a gear symbol on their right. Select a component to see information about it.

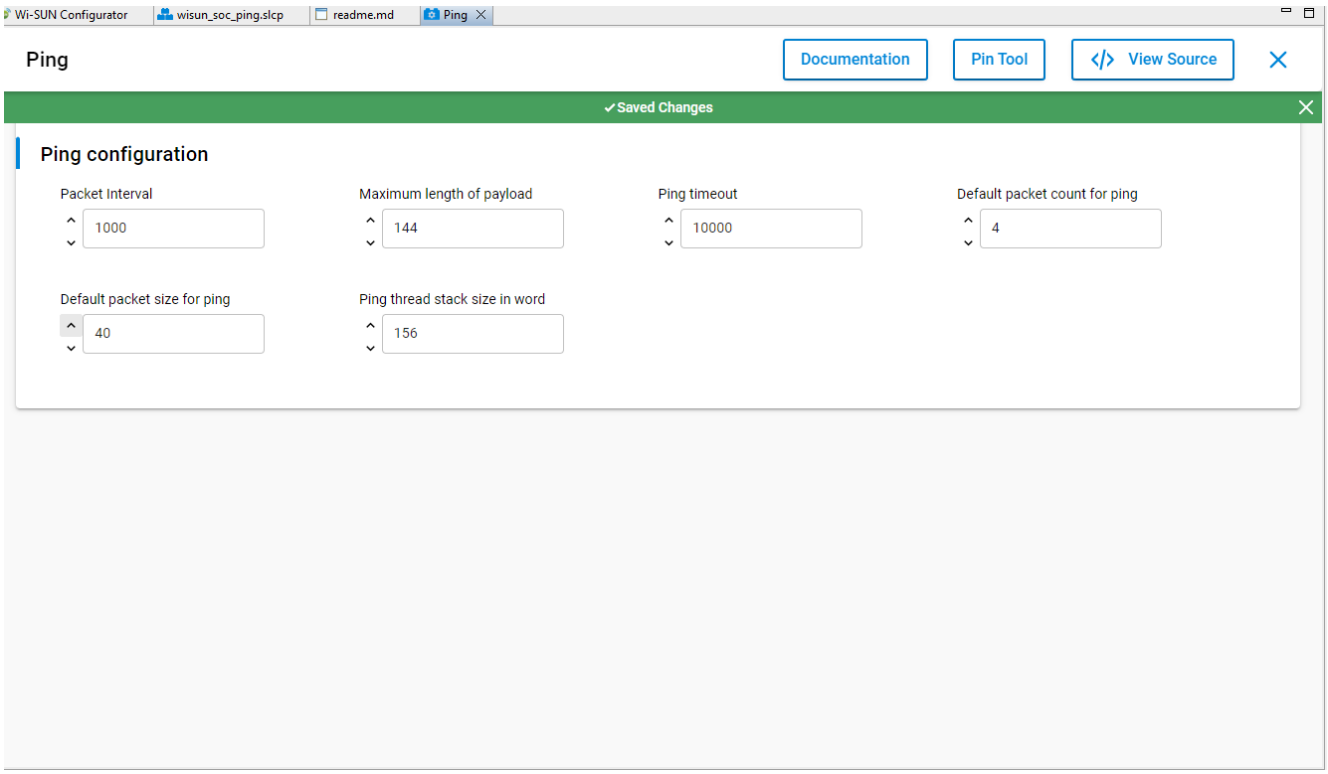


If the component is configurable, click **CONFIGURE** to open the Component Editor in a new tab.

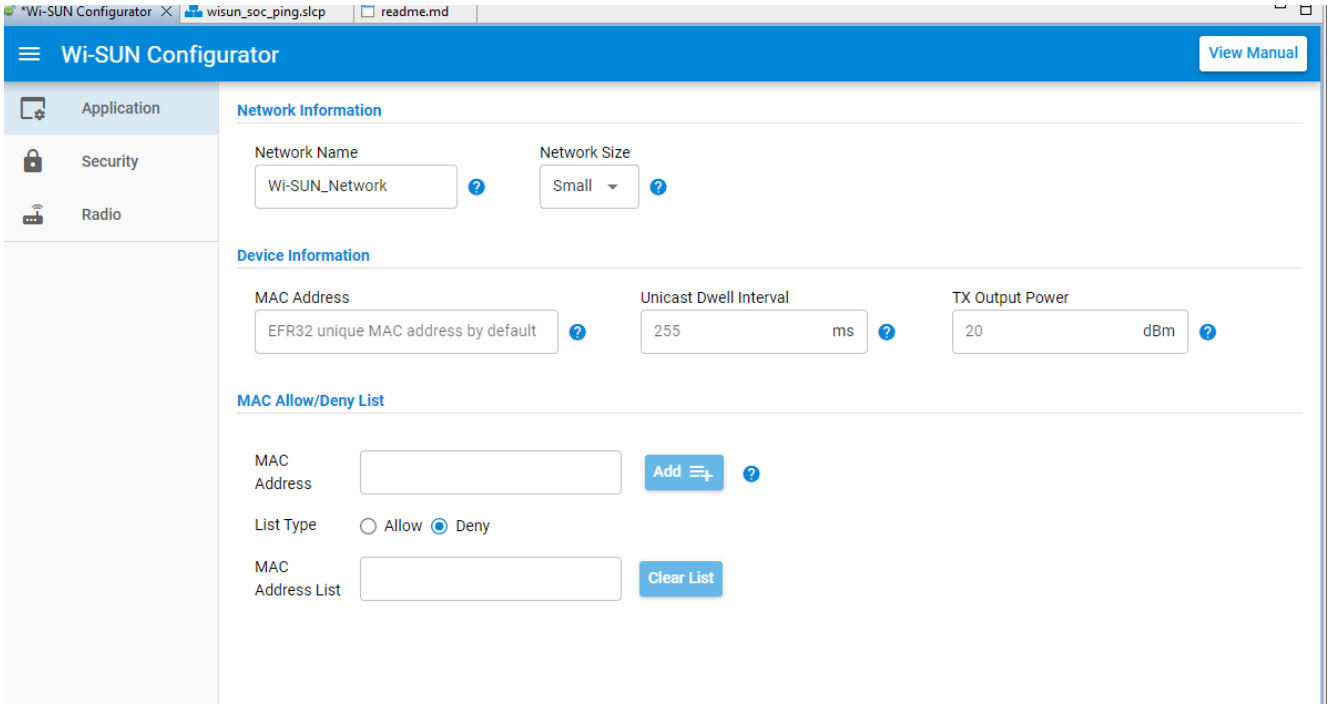
For example, in the Ping component you can configure various parameters that change the way the wisun-ping application behaves.



Any changes you make are autosaved, and project files are autogenerated.



In addition to the Project Configurator tab (<project>.slcp), a Wi-SUN Configurator tab is also available. For more information about using the Wi-SUN Configurator and how to change the default Wi-SUN PHY, see UG495: Silicon Labs Wi-SUN Developer's Guide.



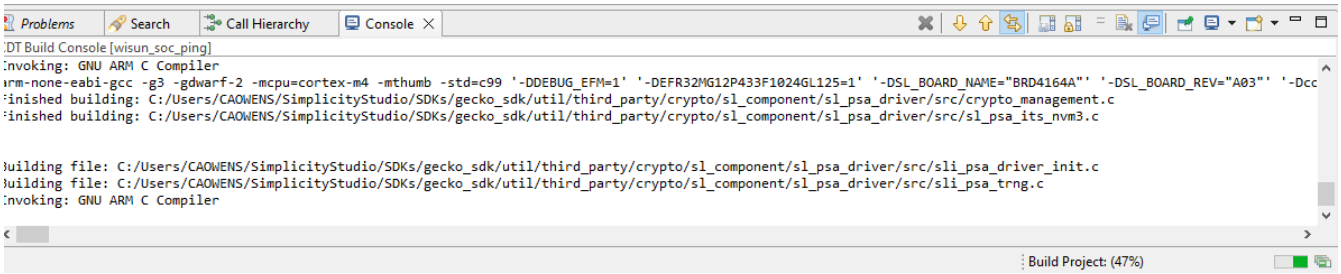
If you make changes, the Wi-SUN Configurator tab has an asterisk to the left. Save changes (CTRL-S) when you are finished updating Wi-SUN Configurator settings. If you build the project without saving changes, the changes are saved automatically.

Building the Project

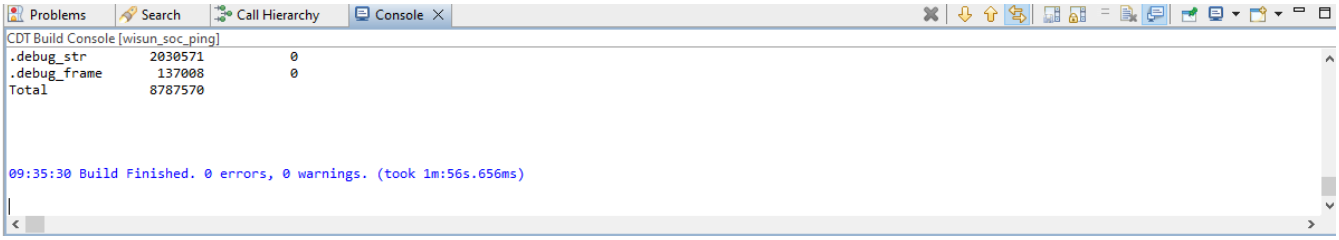
You can either compile and flash the application automatically, or manually compile it and then flash it.

Automatically Compile and Flash

1. You can automatically compile and flash the application to your connected development hardware in the Simplicity IDE, and open a Debug interface. Click **Debug** .
2. Progress is displayed in the console and a progress bar in the lower right.



The project should build without error.



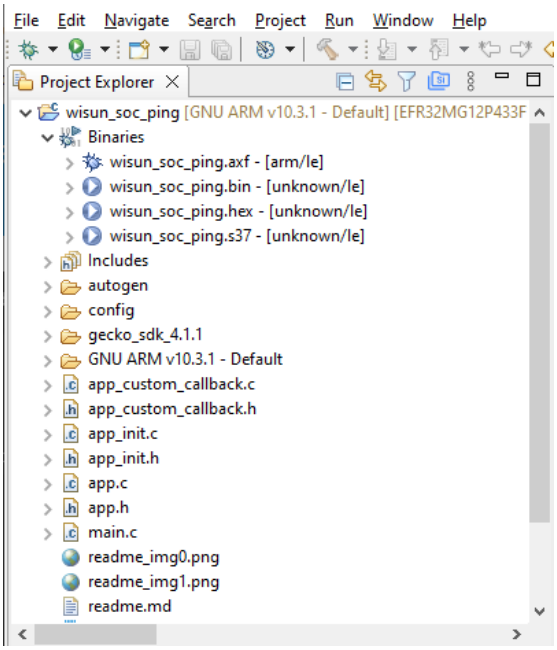
3. When building and flashing are complete a Debug perspective is displayed. Click **Resume** ( ) to start the application running on the WSTK.

Next to the Resume control are **Suspend**, **Terminate**, **Disconnect**, and **stepping** controls. Click **Disconnect** ( ) when you are ready to exit Debug mode.

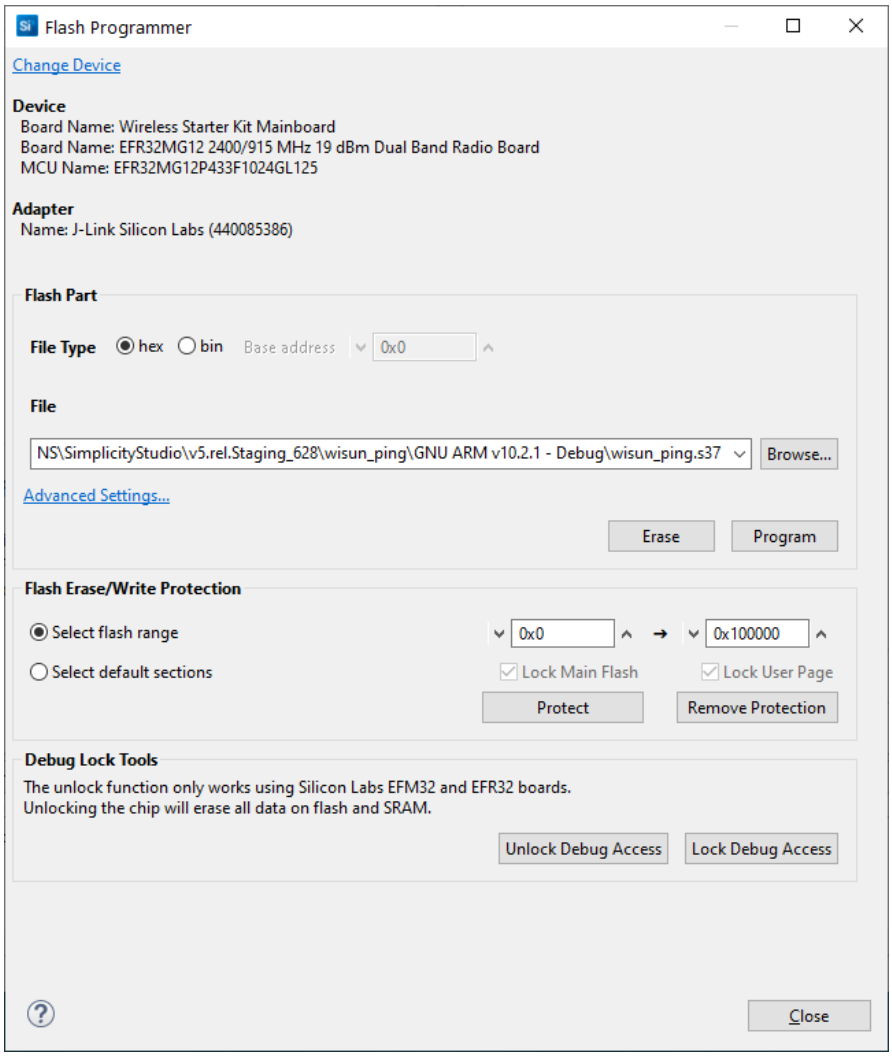
Manually Compile and Flash

1. After you generate your project files, instead of clicking Debug, click **Build** ( ) in the top tool bar.
2. You can load the binary image through Project Explorer view.

Locate the <project>.bin, .hex, or .s37 file in the Binaries subdirectory.



Right-click the file and select **Flash to Device...** If you have more than one device connected, select a device to program. The Flash Programmer opens with the file path populated. Click **PROGRAM** to flash the image to the device.



Flashing a Bootloader

All Silicon Labs examples require that a bootloader be installed. A bootloader is a program stored in reserved flash memory that can initialize a device, update firmware images, and possibly perform some integrity checks. Silicon Labs networking devices use bootloaders that perform firmware updates in two different modes: standalone (also called standalone bootloaders) and application (also called application bootloaders). An application bootloader performs a firmware image update by reprogramming the flash with an update image stored in internal or external memory. By default, a new device is factory-programmed with a bootloader, which remains installed until you erase the device. The Gecko Bootloader is a code library configurable through Simplicity Studio's IDE to generate bootloaders that can be used with a variety of Silicon Labs protocol stacks. The Gecko Bootloader is used with all EFR32xG parts. For more information about bootloaders see *UG103.6: Bootloader Fundamentals*.

By default, a new device is factory-programmed with a bootloader. If you have a new device, haven't cleared the bootloader region for your part or have a supported bootloader image already flashed on your device, skip this step and continue with the next section.

With Silicon Labs Wi-SUN, the bootloader serves to start the application code within the image you created and flashed in the previous procedure. Once you have installed a bootloader image, it remains installed until you erase the device.

To flash a bootloader, first select one of the bootloader examples, such as **SPI Flash Storage Bootloader (single image)**, and build and flash it as described above. For more information see UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher.

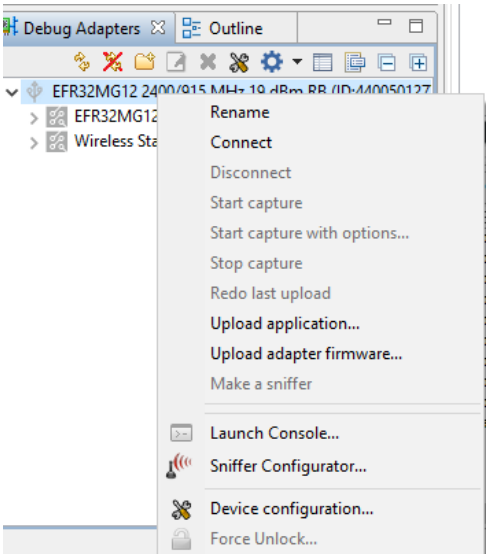
If you are working with the Gecko Bootloader, bootloader images must be formatted as GBL files. To create a GBL file from an .s37 or binary, follow the instructions in UG162: Simplicity Commander Reference Guide, section 6.7.1, GBL File Creation.

The exact format of the GBL file depends on the hardware you selected.

Creating a Network

Depending on the example application, you may be able to interact with it through your development environment's Console interface using a CLI (command-line interface). The console interface allows you to form a network and send data using the border router device created in section Flashing the Wi-SUN Border Router and the application node.

To launch the Console interface, in the Simplicity IDE perspective right-click the application node in the Debug Adapters View. Select **Launch Console**. Alternatively, click Tools in the Simplicity IDE menu and select Device Console.



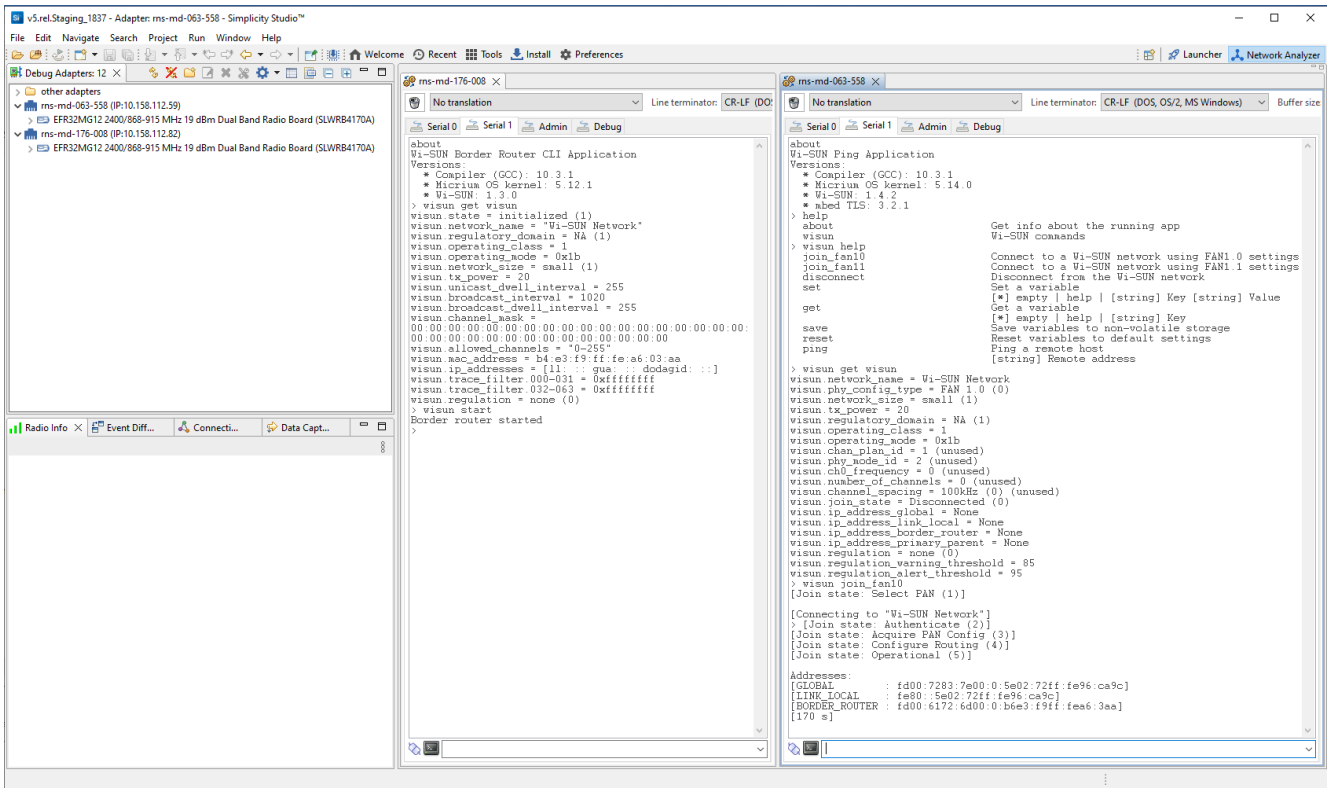
To get a prompt on the Console, go to the Serial 1 tab and press Enter.

Connect the Wi-SUN Ping WSTK

The Wi-SUN Ping application automatically starts by connecting to the Border Router. If the connection is successful, the application should output the traces below in the console.

```
[Connecting to "Wi-SUN Network"]
> [Join state: Authenticate (2)]
[Join state: Acquire PAN Config (3)]
[Join state: Configure Routing (4)]
[Join state: Operational (5)]

Addresses:

[GLOBAL      : fd00:7283:7e00:0:5e02:72ff:fe96:ca9c]
[LINK_LOCAL  : fe80::5e02:72ff:fe96:ca9c]
[BORDER_ROUTER : fd00:6172:6d00:0:b6e3:f9ff:fea6:3aa]
[170 s]
```

The following is an illustration of connecting the Wi-SUN SoC Border Router application to the Wi-SUN SoC Ping Example Application. This setup allows showing the Border Router and the device consoles side by side, where Wi-SUN settings can be compared. If these do not match, the connection will fail. When using a Linux Border Router, use 'wsbrd_cli status' to check the Border Router settings.



The two Wi-SUN devices (Border Router and Wi-SUN SoC Ping) are now part of the same Wi-SUN network.

Ping the Wi-SUN Border Router

To check the commands exposed in the Wi-SUN Ping application, enter:

```
wisun help
```

To retrieve the Border Router IPv6 address, enter:

```
wisun get wisun.ip_address_border_router
```

The Wi-SUN Ping application has a specific command: `wisun ping [IPv6 address]`. Use the command to ping the Border Router.

```
wisun ping [Border Router Global IPv6 address]
```

If the ping command is successful, the pong message size and latency are output on the console.

```
> wisun ping fd00:6172:6d00:0:20d:6fff:fe20:bd95`
PING fd00:6172:6d00:0:20d:6fff:fe20:bd95: 40 data bytes
> [40 bytes from fd00:6172:6d00:0:20d:6fff:fe20:bd95: icmp_seq=1 time=196.231 ms]
```

In this case, the ping took 196 milliseconds to come back to the Wi-SUN device. The ping command can be used to communicate with other Wi-SUN devices in the same Wi-SUN network.

Disconnect and Reconnect the Ping WSTK

You need to disconnect the WSTK and reconnect it to apply new network settings, if you modified them on the Border Router (check these on the Linux Border Router using 'wsbrd_cli status').

Note: It may be worth going through this in situations where the Wi-SUN device does not connect to the Border Router.

To disconnect the WSTK, enter:

`wisun disconnect`

Check the Wi-SUN settings using:

`wisun get wisun````

Set the new Wi-SUN settings using:

`wisun set wisun.<parameter> <value>`

Use the online help or refer to the readme.md file (at the root of you project) to check what parameters are required for your FAN configuration, using:

`wisun set wisun help`

If you want to preserve your settings following a power cycle or reset, use:

`wisun save`

Depending on the FAN configuration, use one of the commands below to connect with your new settings:

`wisun join_fan10`

`wisun join_fan11`

`wisun join_explicit` (only for the Wi-SUN SoC CLI application)

Next Steps

Some next steps:

- Explore the functions available through the Wi-SUN CLI example. In the Serial 1 console connected to a device running the example, enter wisun help to see a list of commands. Enter wisun get wisun to see a list of Wi-SUN network parameters.
- Compile and flash different sample applications and explore the functionality they provide.
- Explore configuring one of the sample applications to meet your needs.
- Explore the documentation provided in this site.

Going Further provides other suggestions once you have created a network.

Going Further

Going Further

After creating a Wi-SUN Network, this section is a guide toward the next steps in exploring the Silicon Labs Wi-SUN solution.

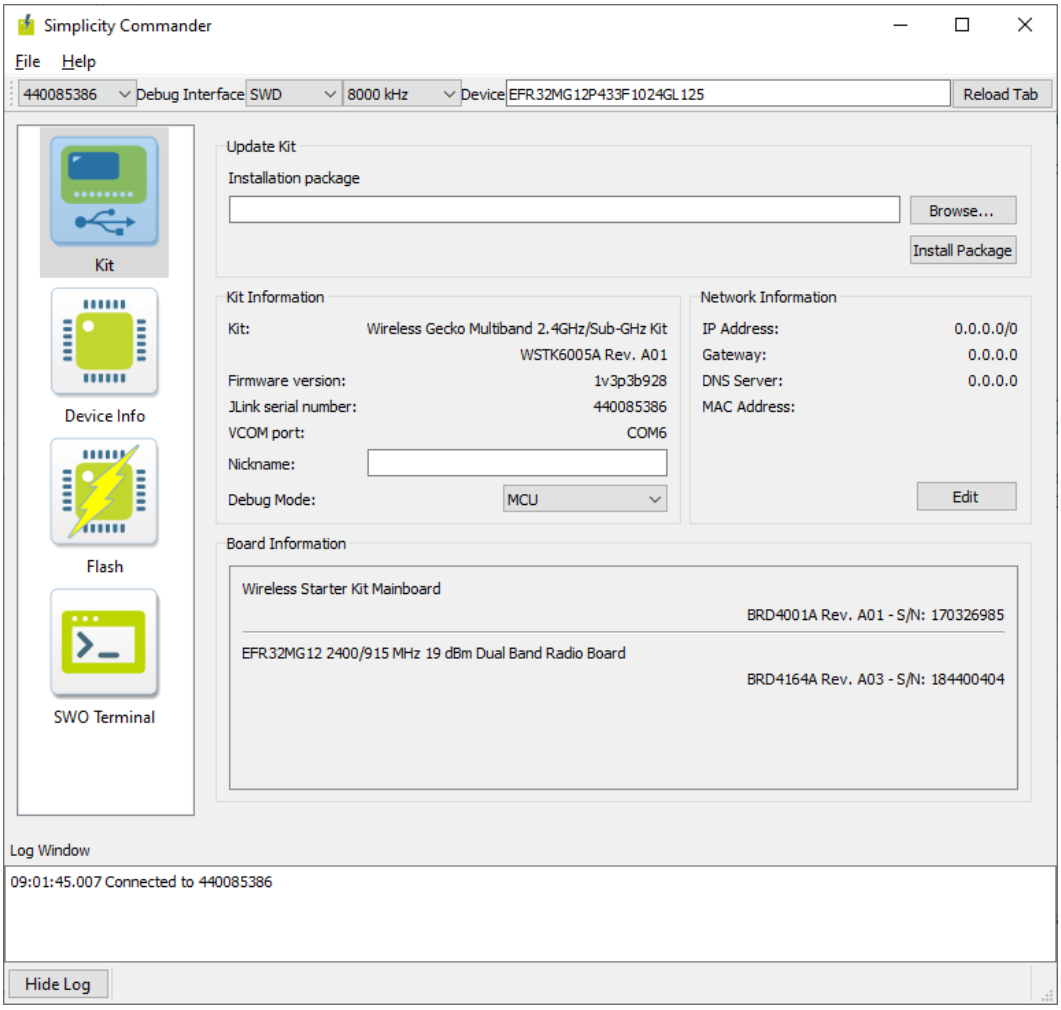
- Wi-SUN Border Router GUI: Under Wi-SUN Border Router, this section presents the Wi-SUN Border Router Dashboard that can be used to configure the Wi-SUN Linux Border router and visualize the Wi-SUN Network.
- **Development Tools**: The development tools below are provided within Simplicity Studio as well as third party.

Gecko Platform

The Gecko Platform is a set of drivers and other lower layer features that interact directly with Silicon Labs chips and modules. Gecko Platform components include EMLIB, EMDRV, RAIL Library, NVM3, and MbedTLS. For more information about Gecko Platform, see release notes that can be found in SSv5's Documentation tab, as well as online API documentation at https://docs.silabs.com/.

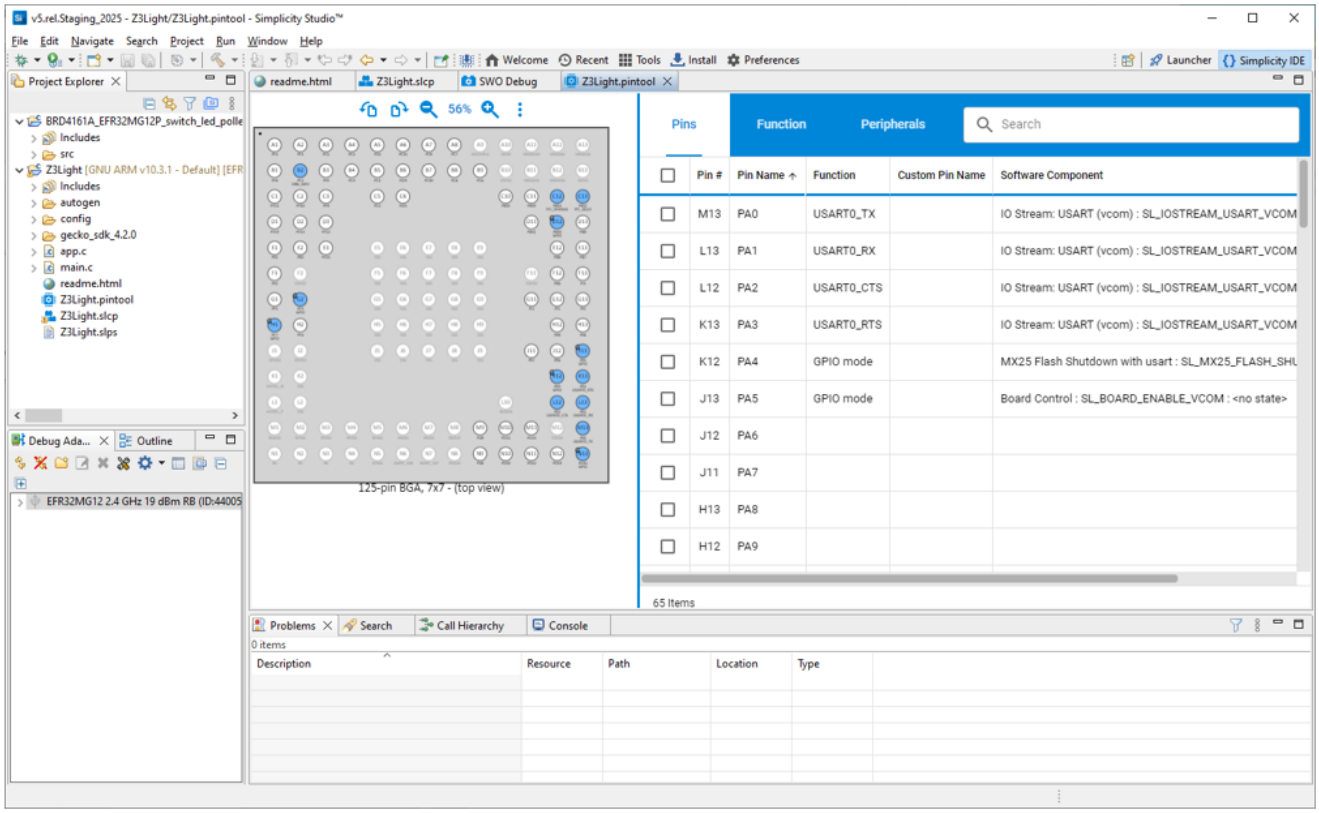
Simplicity Commander

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It is invoked using a simple Command Line Interface (CLI) that is also scriptable. Simplicity Commander enables customers to complete essential tasks such as configuring and building applications and bootloaders and flashing images to their devices. Simplicity Commander is available through Simplicity Studio or can be downloaded through system-specific installers. The **Simplicity Commander User's Guide (PDF)** provides more information.



Pin Tool

Simplicity Studio 5 offers a Pin Tool that allows you to easily configure new peripherals or change the properties of existing ones. In the Project Configurator **SOFTWARE COMPONENTS** tab, expand the **Advanced Configurators** group and open the Pin Tool. The graphical view differs based on the chip.



The pin, function, and peripheral tabs in the configuration pane provide different modes of access. A search function also provided.

Use the Pin Tool to modify the pin configuration of the device. Software components control behavior in the project but must be associated with a peripheral, and generally need pin or function assignments. These pin or function assignments are most easily edited in the Component Editor for that component. The Pin Tool allows you to assign functions to pins. On all three dialogs, click **EDIT** next to a software component to go directly to the Component Editor for that component. Click **NEW** to go to the Project Configurator's SOFTWARE COMPONENTS tab, where you can install a component in the project so that it can be selected in the dialog.

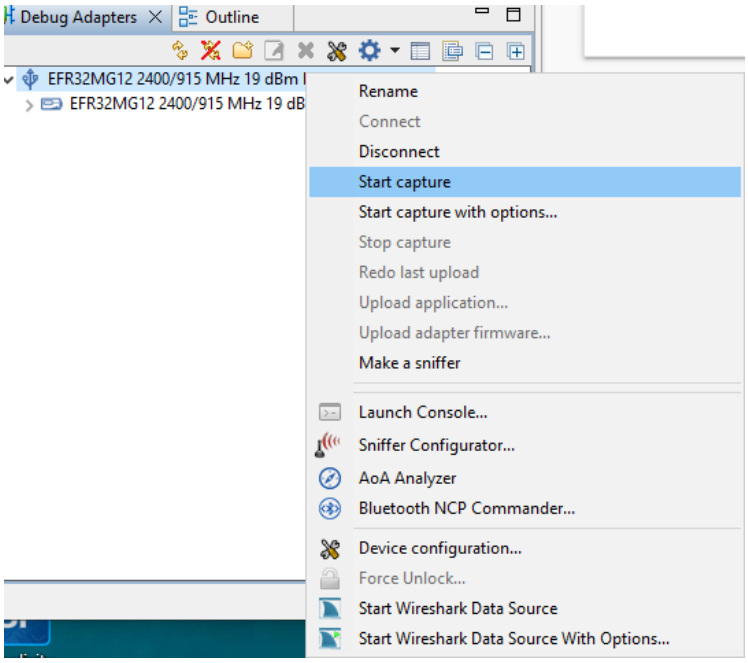
For more information see the Simplicity Studio 5 User's Guide Pin Tool section.

Network Analyzer

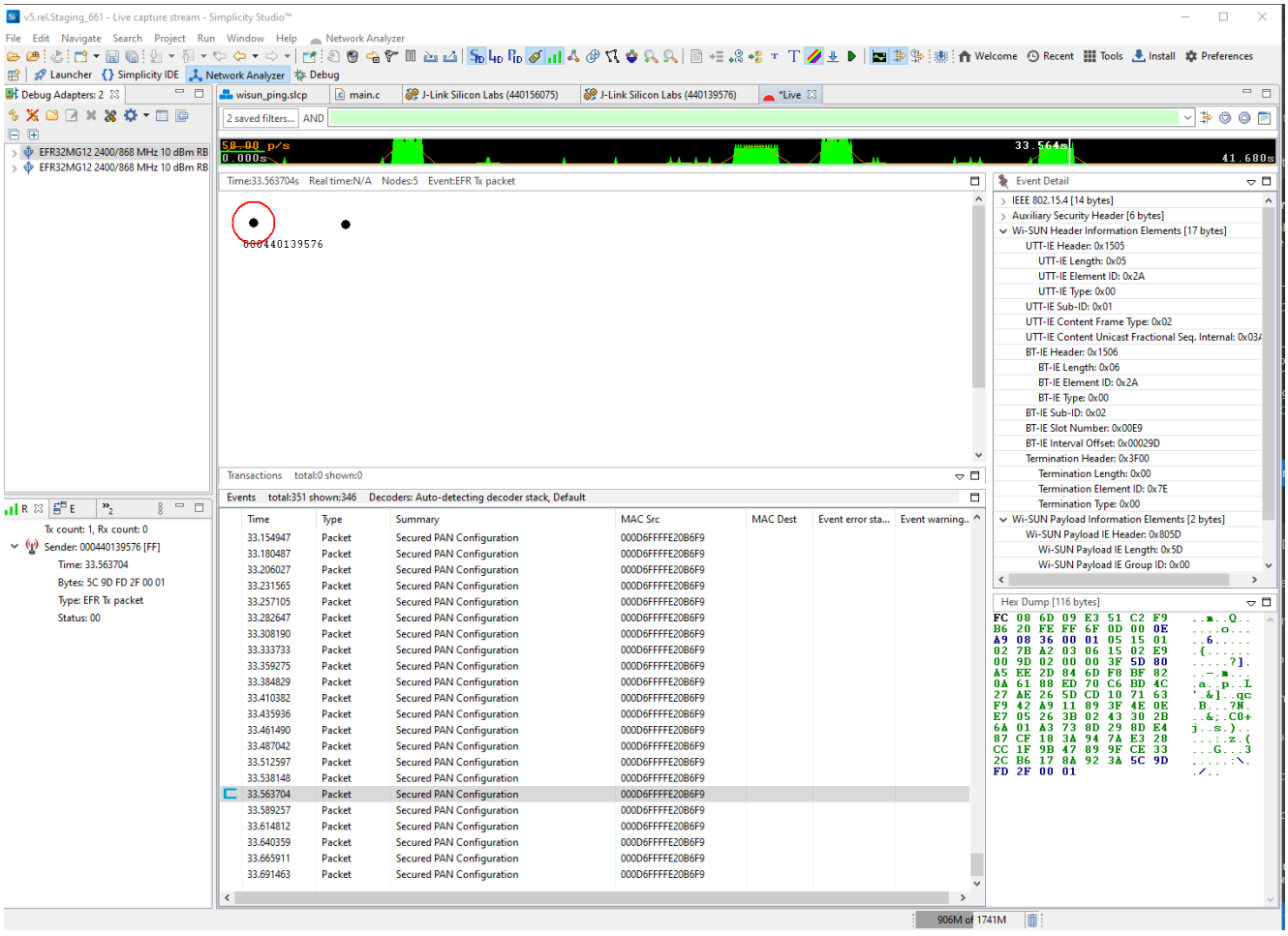
Silicon Labs Network Analyzer is a packet capture and debugging tool that can be used to debug connectivity between wireless nodes running Wi-SUN stack on EFR32 platform. It significantly accelerates the network and application development process with graphical views of network traffic, activity, and duration.

The Packet Trace application captures the packets directly from the Packet Trace Interface (PTI) available on the Wireless Gecko SoCs and modules. It therefore provides a more accurate capture of the packets compared to air-based capture.

To capture Wi-SUN packets using the Silicon Labs Network Analyzer, in the Debug Adapters view, first make sure you are connected and then right-click the device that is running on a Wi-SUN network and select **Start capture**.



You should now be able to see the Wi-SUN traffic as shown below. Click the packets to see more details about its contents in the **Event Detail** view (on the right).



In the current state, the Network Analyzer does not decrypt the Wi-SUN Upper Layer Application Data. To decrypt the packet and analyze the complete payload including the data, refer to UG495: Silicon Labs Wi-SUN Developer's Guide.

Wireshark

Wireshark is the recommended network protocol analyzer for the use with Wi-SUN networks. Download instructions are provided for Windows/Mac users or Linux users. Simplicity Studio® 5 supports live interaction between the application running on a Silicon Labs device and Wireshark.

Silicon Labs Configurator (SLC)

SLC offers command-line access to application configuration and generation functions. **Software Project Generation and Configuration with SLC-CLI** provides instructions on downloading and using the SLC-CLI tool.

IAR IDE

To get a 30-day evaluation license for IAR-EWARM:

1. Go to the Silicon Labs support portal at https://www.silabs.com/support.
2. Scroll down to the bottom of the page, and click **Contact Support**.
3. If you are not already signed in, sign in.
4. Click the Software Releases tab. In the View list select **Development Tools**. Click **Go**. In the results is a link to the IAR-EWARM version named in the release notes.
5. Download the IAR package (takes approximately 1 hour).
6. Install IAR.
7. In the IAR License Wizard, click **Register with IAR Systems to get an evaluation license**.
8. Complete the registration and IAR will provide a 30-day evaluation license.
9. Once IAR-EWARM is installed, the next time Simplicity Studio starts it will automatically detect and configure the IDE to use IAR-EWARM.

Overview

Wi-SUN Development Walkthrough

The purpose of this Wi-SUN Development Walkthrough is to help Wi-SUN application developers start their application development based on **Wi-SUN SoC Empty**, and to build a final production-ready application, after initial evaluation of the Wi-SUN example applications.

The Development Walkthrough is organized in the following sections:

- Build an Application and Connect to a Wi-SUN Network
- Wi-SUN API Calls to Connect to a Wi-SUN Network
- Add a Custom Application to a Wi-SUN Network
- Add OTA DFU (Over-The-Air Device Firmware Upgrade) Capability

Silicon Labs expects that developers have evaluated some of the Wi-SUN example applications prior to beginning this walkthrough to get a basic understanding of Wi-SUN. These applications are available in Simplicity Studio through the Launcher perspective on the **EXAMPLE PROJECTS & DEMOS** tab. The intended uses of these Wi-SUN example applications are:

- An application to build a stand-alone Wi-SUN Border Router on an evaluation kit
  - **Wi-SUN - SoC Border Router**
    - Also available as a 'demo' in binary format for easy evaluation. It is convenient for quick testing, while being limited in terms of features. Moving to the Linux Border Router (below) is recommended to start communicating with the Wi-SUN network from the outside.
- An application to build a Linux Wi-SUN Border Router using an evaluation kit as the RCP (Radio Co-Processor) and a Linux platform as the host (more details on the Linux Border Router on GitHub for `wsbrd` and the corresponding GUI).
  - **Wi-SUN - RCP**
    - Also available as a 'demo' in binary format for easy evaluation
- An application to test most Wi-SUN Stack API commands: **Wi-SUN - CLI example**
  - Also available as a 'demo' in binary format for easy evaluation
    - Each Radio Board has a matching 'demo' binary, depending on its Frequency Band. This means all PHYs will not be available for a given Radio Board, only the ones compatible with the Radio Board Frequency Band.
- Applications to evaluate Wi-SUN
  - ping
    - **Wi-SUN - SoC Ping**
  - TCP
    - **Wi-SUN - SoC TCP CLient**
    - **Wi-SUN - SoC TCP Server**
  - UDP
    - **Wi-SUN - SoC UDP CLient**
    - **Wi-SUN - SoC UDP Server**
    - **Wi-SUN - SoC UDP Meter**
    - **Wi-SUN - SoC UDP Collector**
  - CoAP
    - **Wi-SUN - SoC CoAP Meter**
    - **Wi-SUN - SoC CoAP Collector**
  - Network Performance (ping/iperf)
    - **Wi-SUN SoC Network Measurement**
      - Also available as a 'demo' in binary format for easy evaluation
- An application to start final development, after the evaluation phase is complete
  - **Wi-SUN Soc Empty**

These applications allow quick evaluation of Wi-SUN without the need to compile anything for the first level of testing, thanks to the pre-compiled 'demo' binaries. Applications not delivered as binaries can be created easily, built, and flashed to

the evaluation kits.

After this evaluation phase is achieved, actual application development starts, based on the minimal **Wi-SUN SoC Empty** example application. It is minimal is the sense that it allows connecting to the Wi-SUN network and doesn't contain application code, which will be added by the customer.

Build and Connect

Build Wi-SUN - SoC Empty and Connect to any Wi-SUN Network

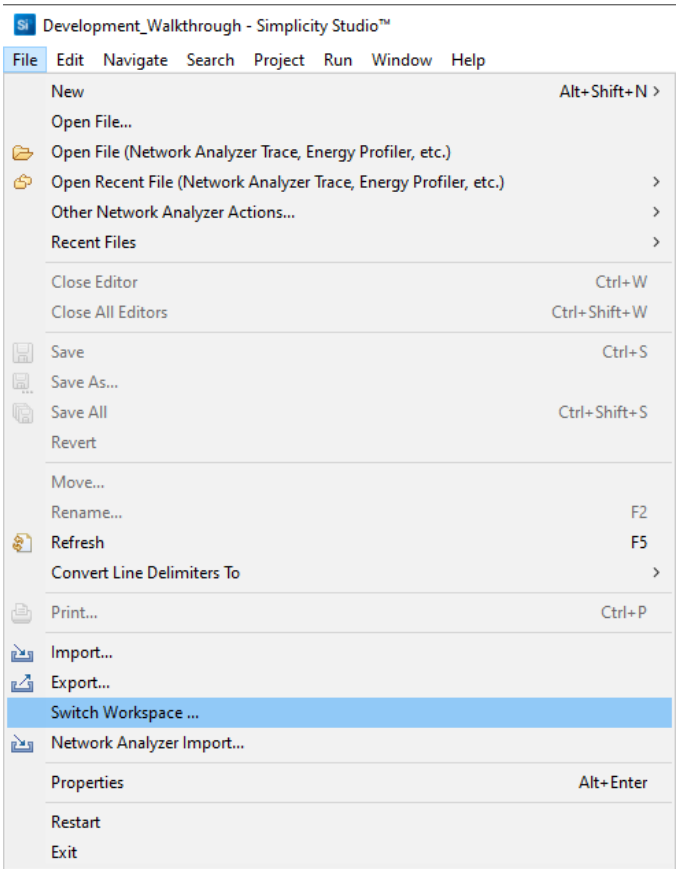
The Development Walkthrough begins with Simplicity Studio.

Starting Simplicity Studio

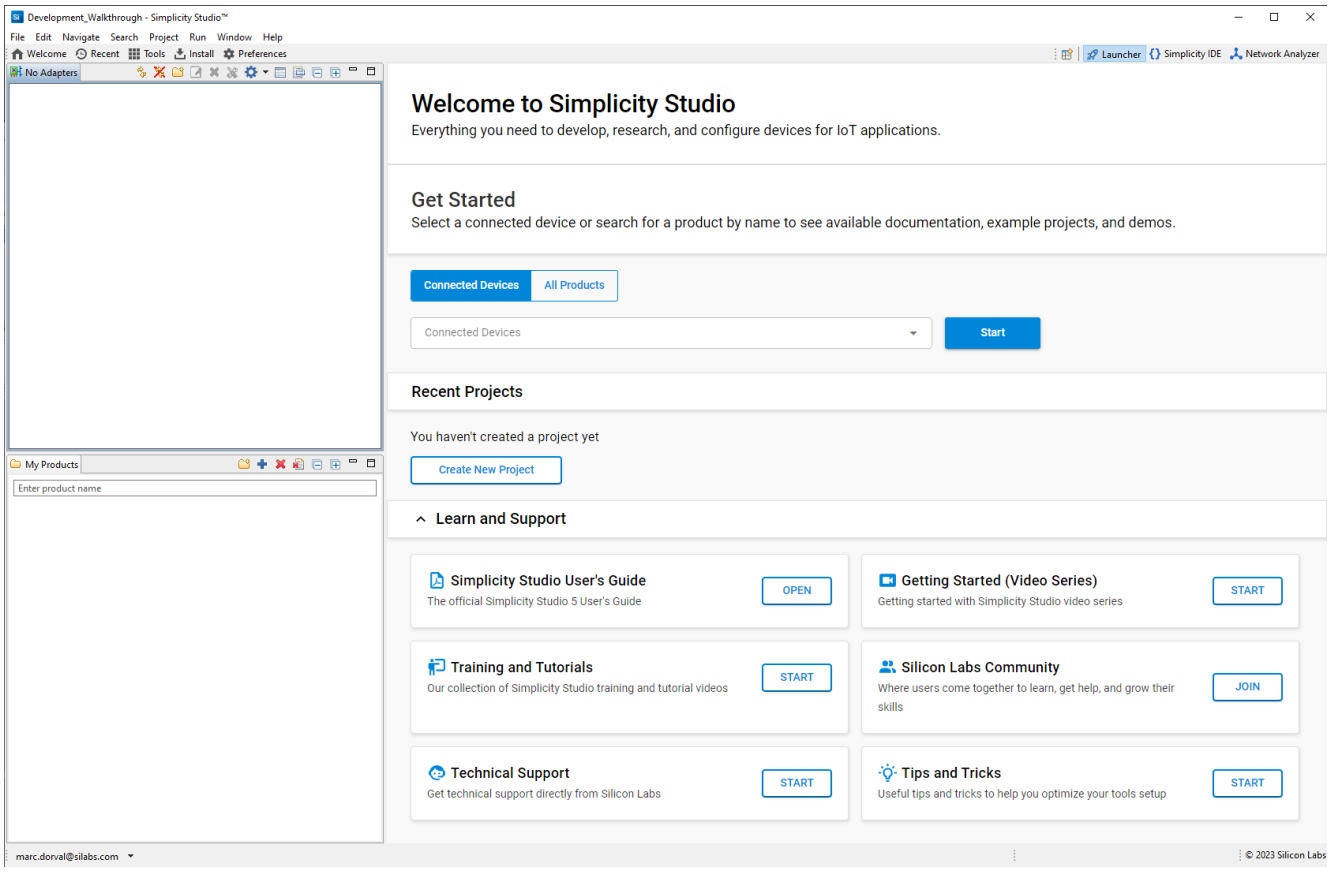
It is best to keep the code separate from other projects for the sake of clarity. For this reason, start by creating a `Development_Walkthrough` folder to serve as a workspace folder for Simplicity Studio.

In Windows, the default path for Simplicity Studio workspaces is `C:\Users\username\SimplicityStudio`, so create `C:\Users\username\SimplicityStudio\Development_Walkthrough`.

Now, start Simplicity Studio and switch to the workspace, if not already selected, by selecting **File > Switch Workspace**.

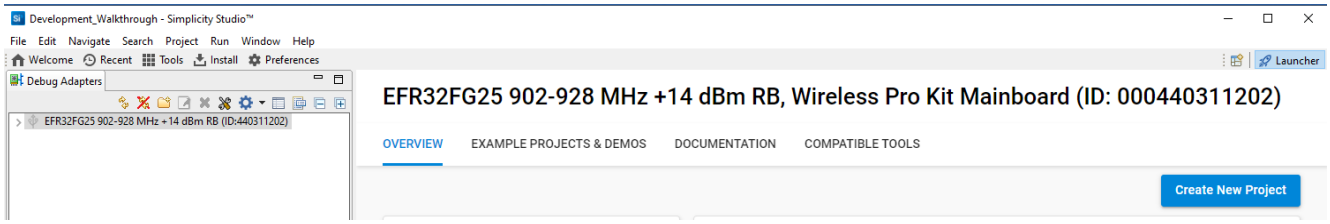


Once in the new workspace, you see the Simplicity Studio Welcome Screen.

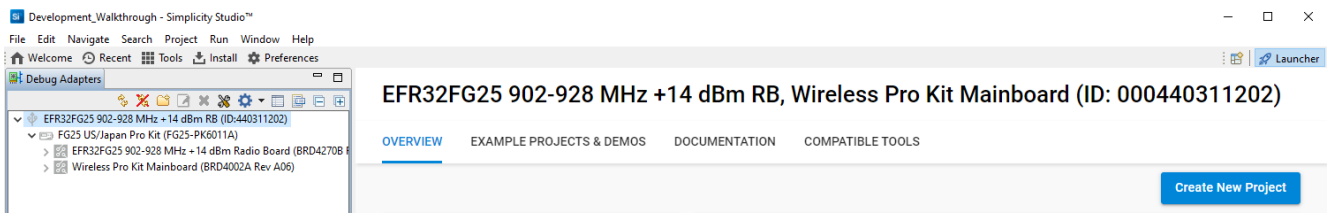


Connecting a Wi-SUN Development Kit

At this point, you have no Adapter Board connected, so fit a Wi-SUN compatible Radio Board to a Silicon Labs WSTK/WPK from a Wi-SUN Development Kit and plug it into a USB port on the PC to get it displayed in the **Debug Adapters** frame.



You can expand the Adapter information to check the kit's information, including the Radio Board reference, the exact part number, and the Main board information.



> TIP: Additional information on the Wi-SUN kits is available on the Silicon Labs Wi-SUN webpage.

Creating the Wi-SUN - SoC Empty Application

Wi-SUN - SoC Empty is the recommended application to start development from, since it contains the necessary resources to easily connect Wi-SUN devices to any Wi-SUN network.

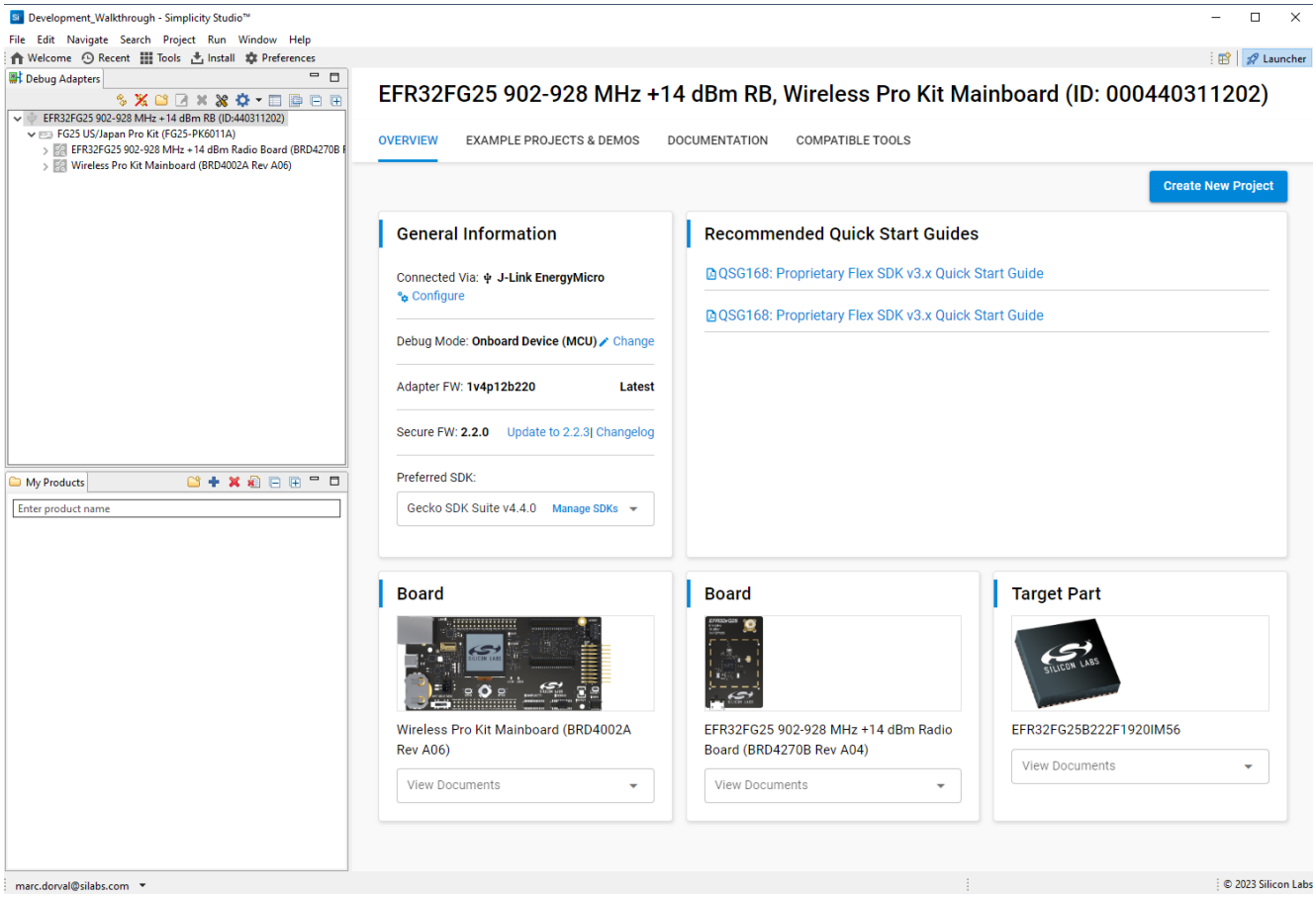
In this example:

- Use console `printf` calls to help follow the application progress. It makes application development easier to get console traces. Any C developer will know how to make these optional in a 'release' build, if required, although minimal traces are always helpful for maintenance reasons, if only to display the application and version information at startup.
- Don't add a CLI (Command Line Interface) through the console, because:
  - In a Wi-SUN network, the only access to the devices is through the Wi-SUN network itself.
  - Generally, access to a device console will not be possible. It requires physical access to the device.
  - Not adding a CLI makes the console impossible to use to take control of the device, making the application more secure.

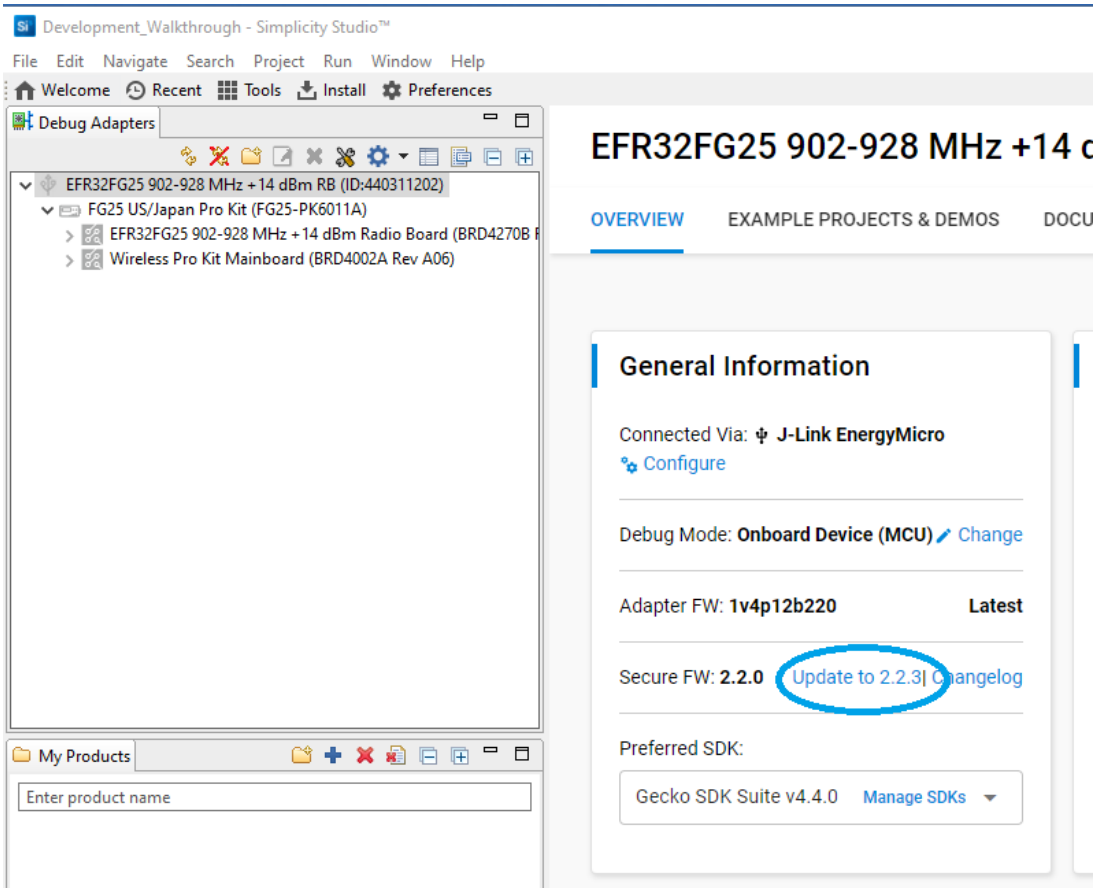
> TIP: A typical WI-SUN device also requires OTA DFU capability for updates/upgrades. Silicon Labs recommends adding OTA DFU once the Wi-SUN SoC Empty application is working as expected.

After connecting a Wi-SUN Development Kit and selecting it, Simplicity Studio opens the Launcher perspective, where information on the selected Adapter is displayed, and Simplicity Studio is ready to help create matching projects.

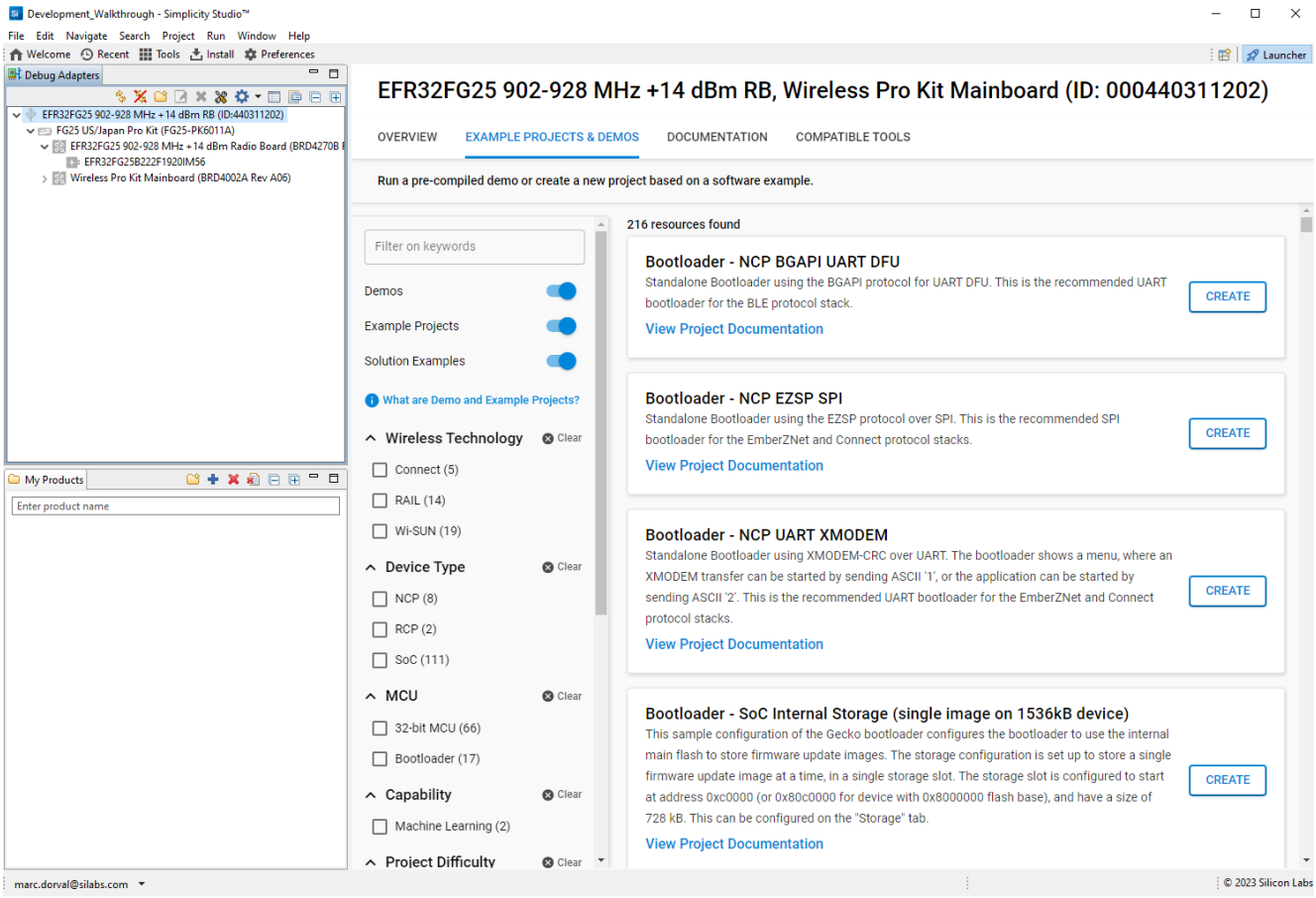
Simplicity Studio only displays information relevant to the context it is in, so here, because the target Debug Adapter is selected, it shows the Launcher perspective and the **OVERVIEW** tab for the evaluation kit. You also have easy access to the corresponding documentation and tools.



TIP: When the **General Information** box indicates the need to update the Adapter FW or the Secure FW to newer versions, it is recommended to do so. New releases generally improve stability.



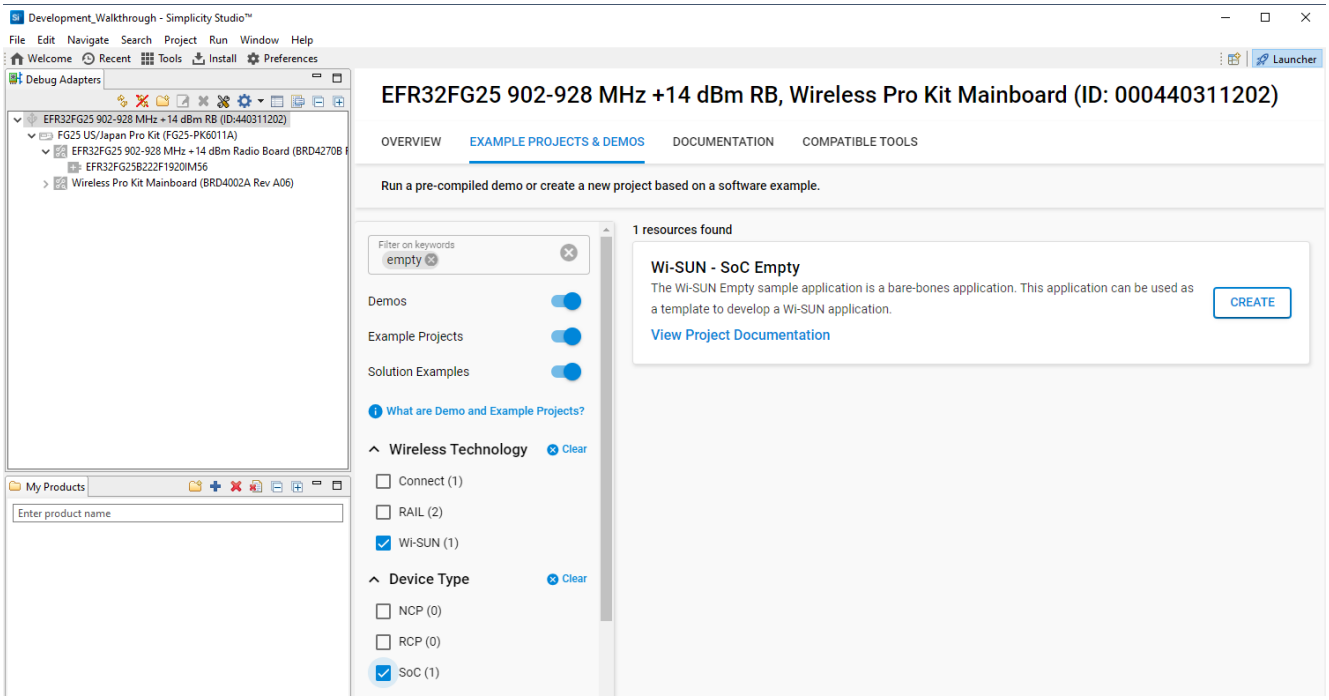
To get started with the Wi-SUN SoC Empty application, select the **EXAMPLE PROJECTS & DEMOS** tab, still in the **Launcher** perspective, and wait for Simplicity Studio to display the examples. It can take a couple seconds.



Filter the projects by clicking the following items:

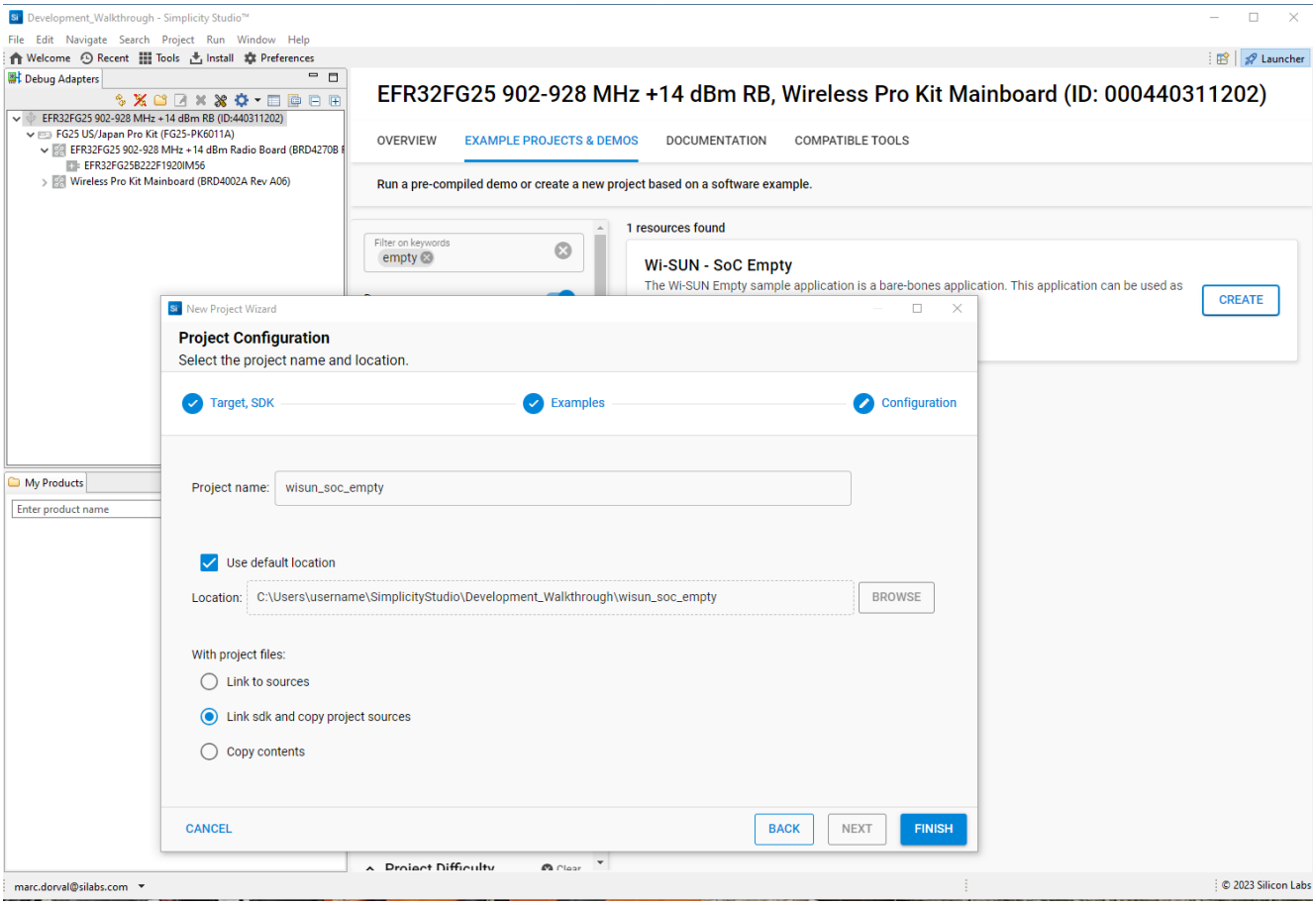
- Under **Wireless Technology**: **Wi-SUN**
- Under **Device Type**: **SoC**

To further limit to items containing 'empty', type 'empty' in the **Filter on keywords** box, and press **Enter**.



This makes it easy to locate the **Wi-SUN - SoC Empty** project.

Now click **CREATE** in the **Wi-SUN - SoC Empty** box to start creating your project.

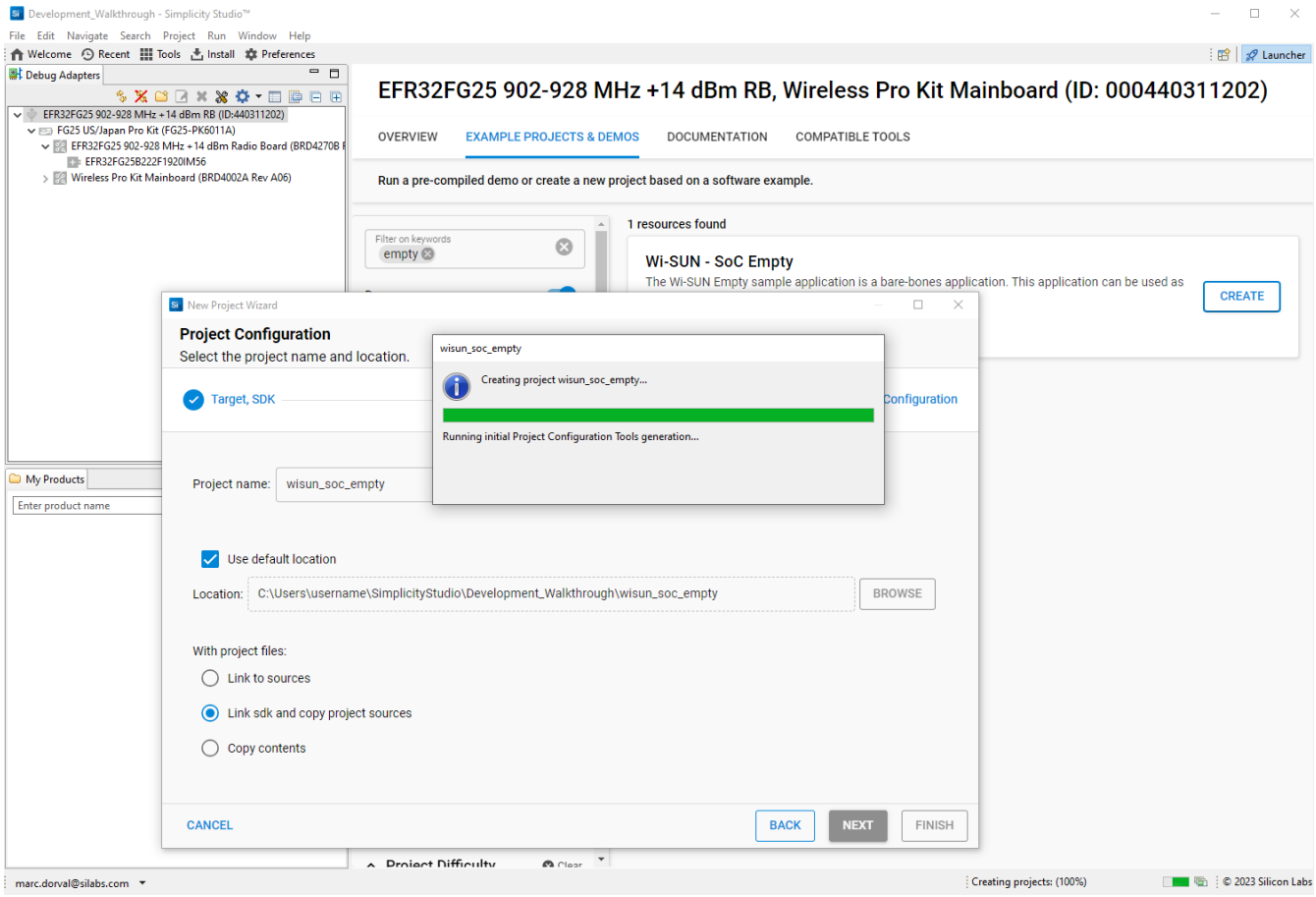


Since you are happy with the default project name and location, click **FINISH** to create the project.

> TIP: An alternate path to project creation is **File > New > Silicon Labs Project Wizard**. If you use this path after selecting your development kit in the **Debug Adapter** window, the settings will be pre-filled for it as well. You will be at the **Target/SDK** selection step (Target, SDK) in this case, two steps higher in the process than the **Configuration** step above (Configuration). When using the Launcher path just described, you can still click the **Back** button twice to go back to the **Target/SDK** selection step, which allows you to select an IDE (to edit your code) and a toolchain (to compile your code) if the default IDE (Simplicity IDE) and toolchain (GNU ARM) are not what you want to use.

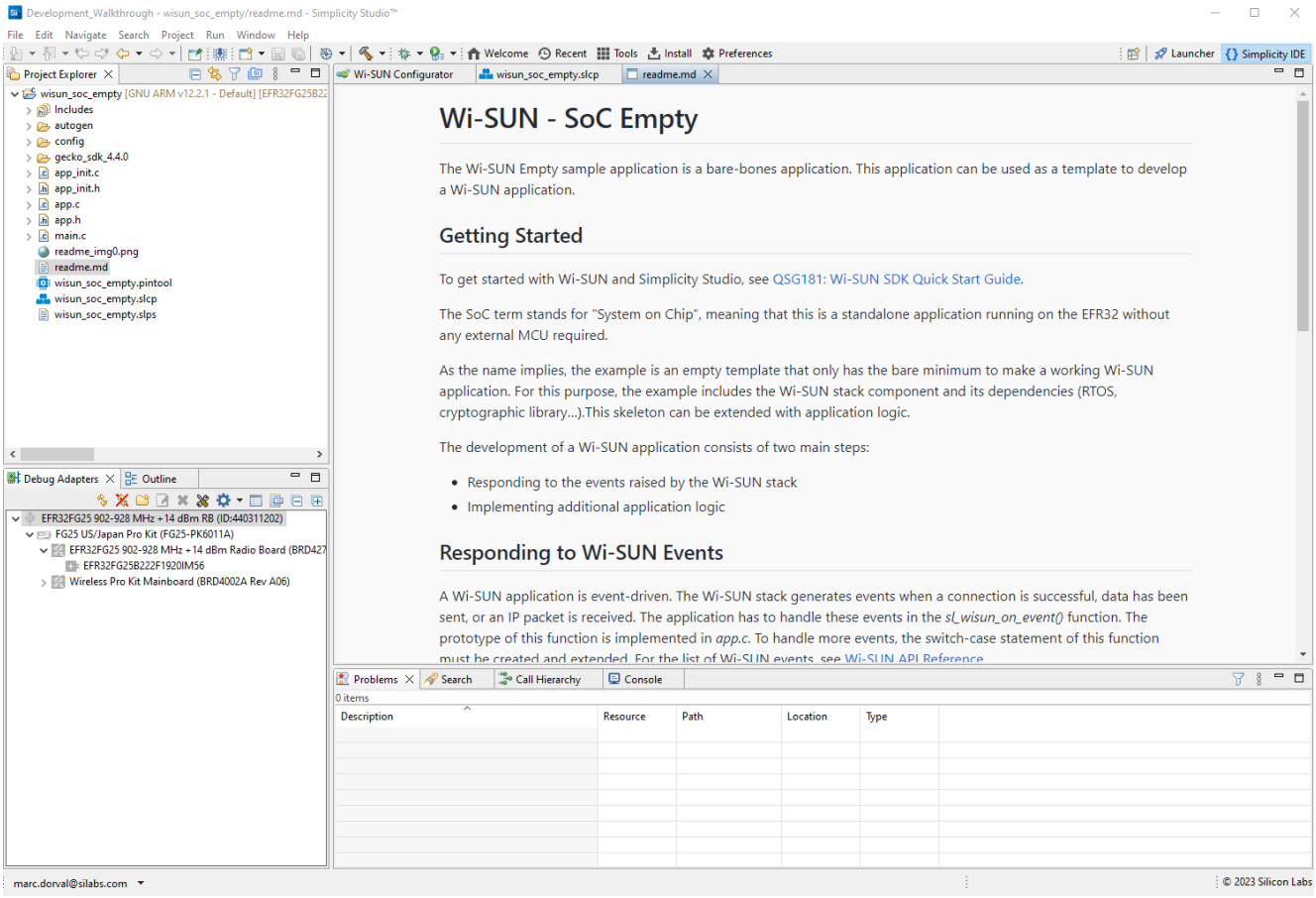
Notes on the 'With project files' Options

- Part of the code in the application is the Wi-SUN Stack. This is delivered as pre-compiled libraries matching each Stack_application/Part/Compiler/Configuration combination. Customers don't have access to the source code for this part, to keep it under control by Silicon Labs. It's complex, and the Wi-SUN stack developers are responsible for keeping it working and up to date, and they add new features over time. These will be linked with the rest of the object files in the project. You will check this after compiling your first project.
- Another part of the code is the GSDK (Gecko Software Development Kit) code which is common to all projects. Customers are given access to this code. When a new project is created in your workspace, the preferred method consists of creating symbolic links to those 'sdk' sources, to save space and allow updating the GSDK version in the future. It ensures that all the projects use the same Wi-SUN Stack code, and it can be easily updated.
- The final part of the code is the application code. Customers need to add their own application code there. The preferred method here is to copy the 'project sources' code from the GSDK to your project's folder, such that changing it will keep changes local to your project. This is why the default option is **Link sdk and copy project sources**, which is fine for most cases.



Created Project

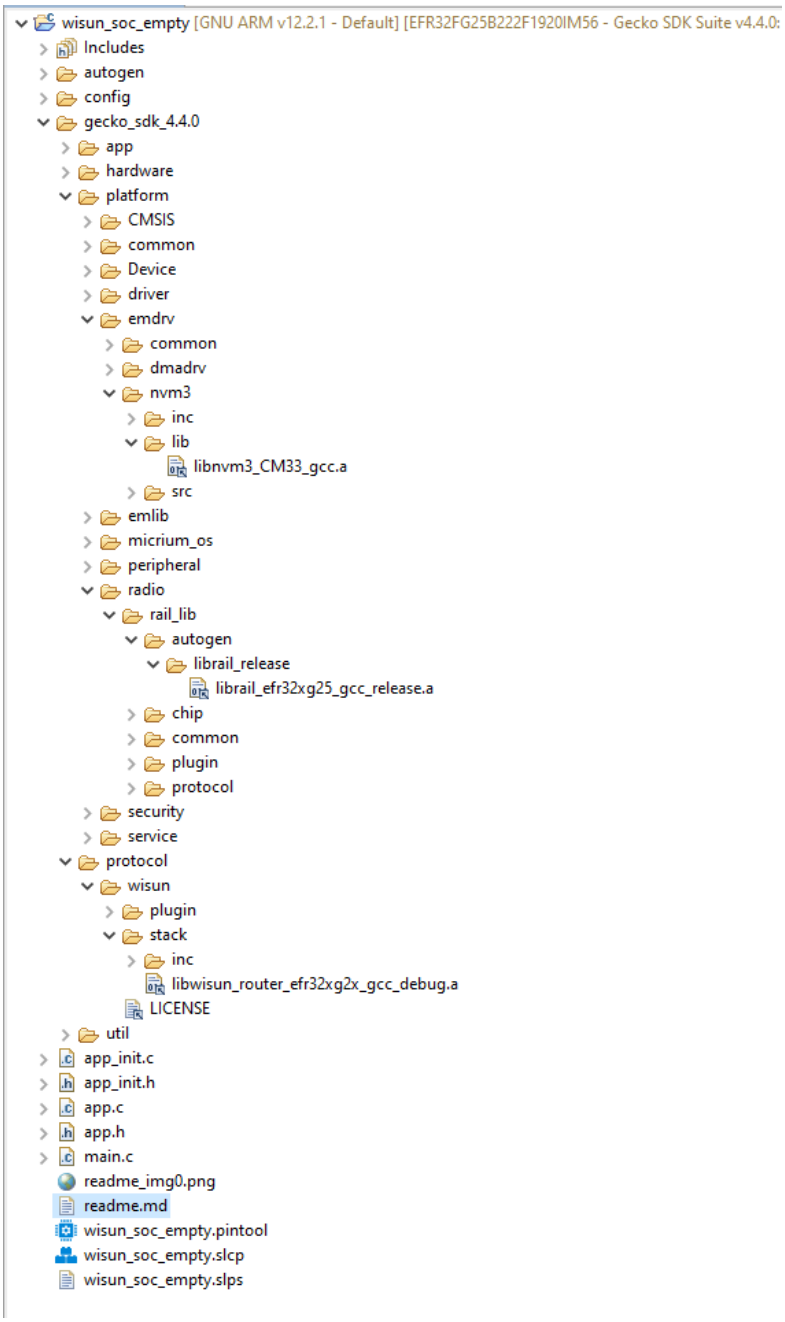
Once project creation is complete, Simplicity Studio switches to the {} Simplicity IDE perspective (see the top-right corner), and is ready for project configuration and compilation.



Three tabs are present in this perspective:

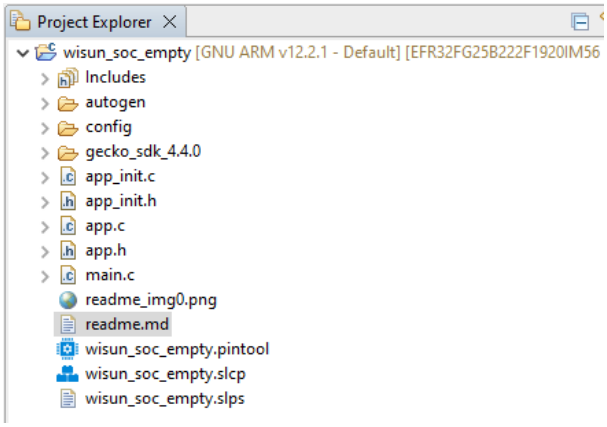
- **Wi-SUN Configurator**: Used to set your Network / Security / Radio options via three tabs.
  - TIP: You can always get access to the Wi-SUN Configurator by double-clicking on `config/wisun/wisun_settings.wisunconf`, if you ever close it.
- **wisun_soc_empty.slcp**: Your access to the previous view with the **Overview** / **Software Components** / **Configuration Tools** tabs.
  - TIP: Similarly, you can always access this tab by double-clicking on the `wisun_soc_empty.slcp` file in the **Project Explorer** tab.
- **readme.md**: The project's documentation, with links to the Quick Start guide and additional details on docs.silabs.com.
  - TIP: Open the `readme.md` file in the **Project Explorer** tab to re-open it.

You can see the pre-compiled libraries, which will be linked with your object files.

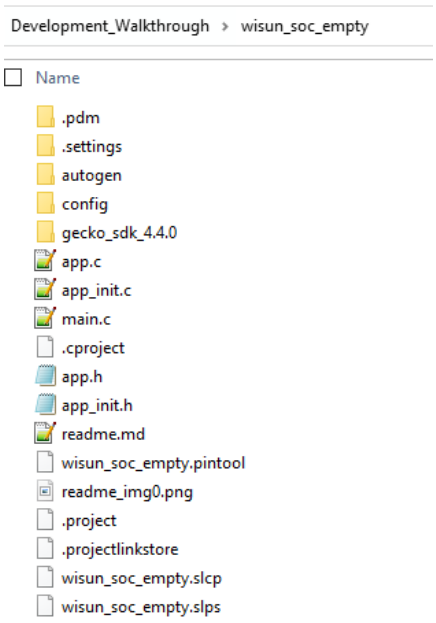


Project Explorer

Now look at what you have in your **Project Explorer** view when selecting the project.



This matches the files in your workspace folder, since it's a file view. Note that, if using the **Show hidden files, folders, and drives** folder option in Windows, you also see the .* folders and files below.



Project Sub-Folders

- `includes` contains links to the sdk and stack API header files relevant to your project.
- `autogen` contains all files copied from the GSDK which contain the application code you may want to customize, plus component configuration files.
- `config` contains the files giving access to the Wi-SUN Configurator and to the RAIL configuration, plus some low-level stack configuration files.
- `gecko_sdk_x.y.z` contains links to the GSDK files required by your project. Only links to files relevant to your use case are included, not the entire GSDK.

> TIP: If you want to modify one of the GSDK files, the IDE will ask you whether you want to edit the GSDK file or create a copy of the file (i.e. replacing the symbolic link to the file by a copy of the file). With the first option, changes you make will be shared with all other projects, and possibly lost with GSDK updates. The second option will keep your changes local to the current project.

Using Git to Track Changes

For the sake of documenting/understanding what happens when you act on your project and tracking changes, Silicon Labs recommends using Git to track changes.

The following actions are required for this:

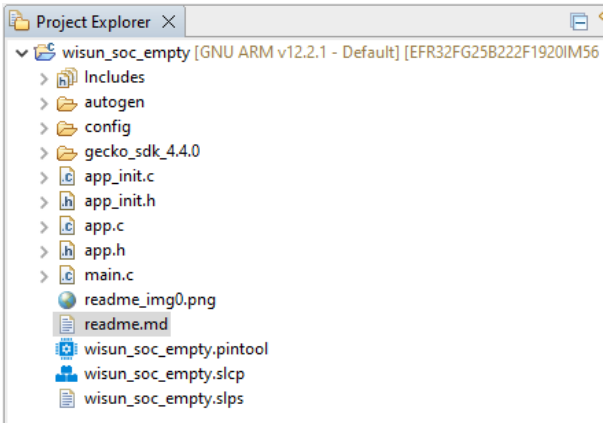
1. Open a Git Bash command window in the `Development_Walkthrough/wisun_soc_empty` folder.
2. Make it a Git-controlled folder using `git init`.
3. Copy the `gitignore` file to this folder and rename it to `.gitignore`. This is to avoid tracking object files, git files. Renaming the file is necessary to include this git-specific file to this documentation.
4. Use `git config core.autocrlf false` to avoid changing the line endings.
5. Use `git add --all` to add all selected files to git.
6. Use `git commit -m "Initial Commit"` to commit the initial files.

> TIP: Later on, opening the `Development_Walkthrough/wisun_soc_empty` folder with Visual Studio Code is a nice way to get access to a revision control interface, for those not familiar with git's command line.

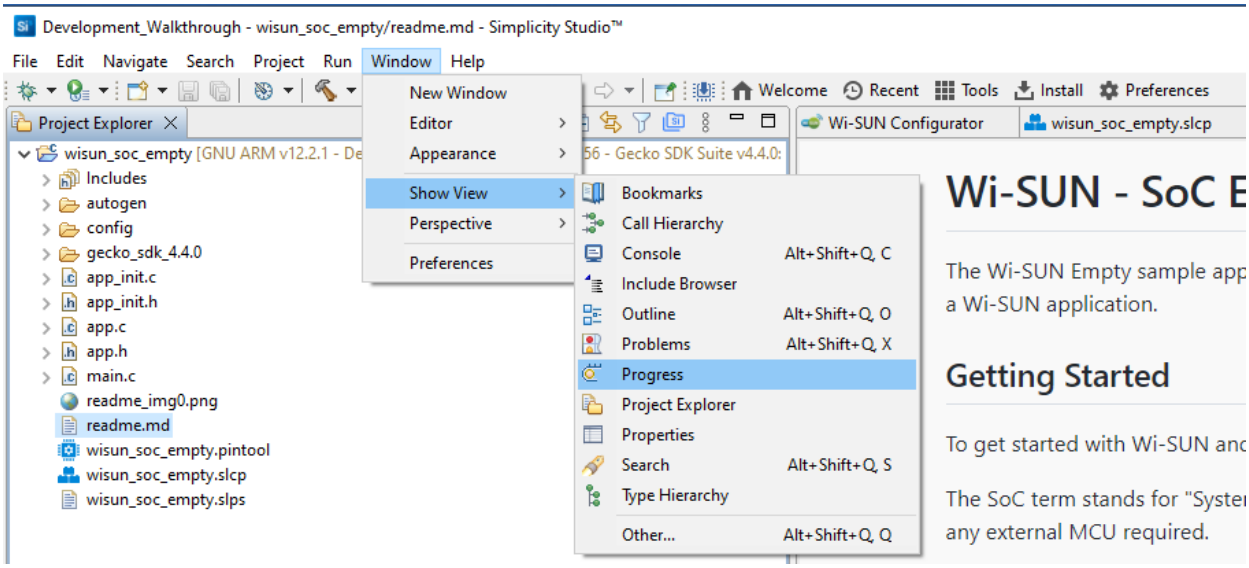
First Compilation

Back to Simplicity Studio's **Simplicity IDE** perspective, with your project selected, build your project with all settings by default.

> TIP: Click either on the project or in any of its opened files to select it. Check that it's the current project in the top bar.

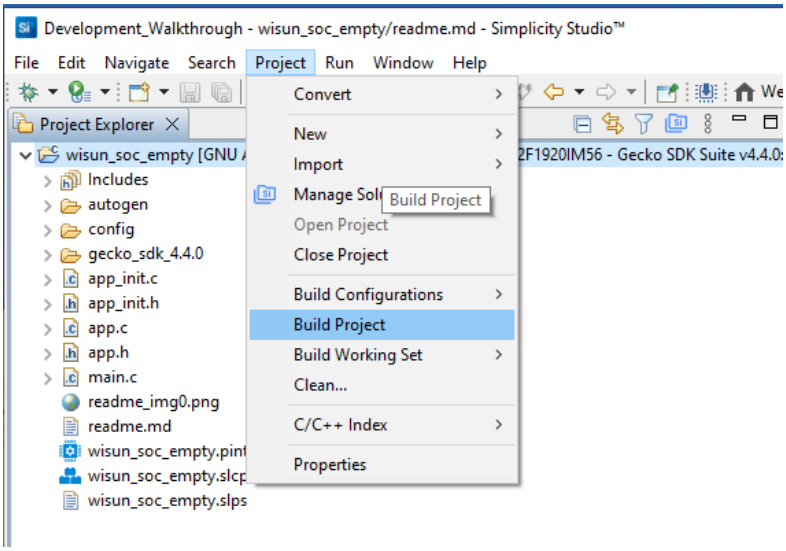


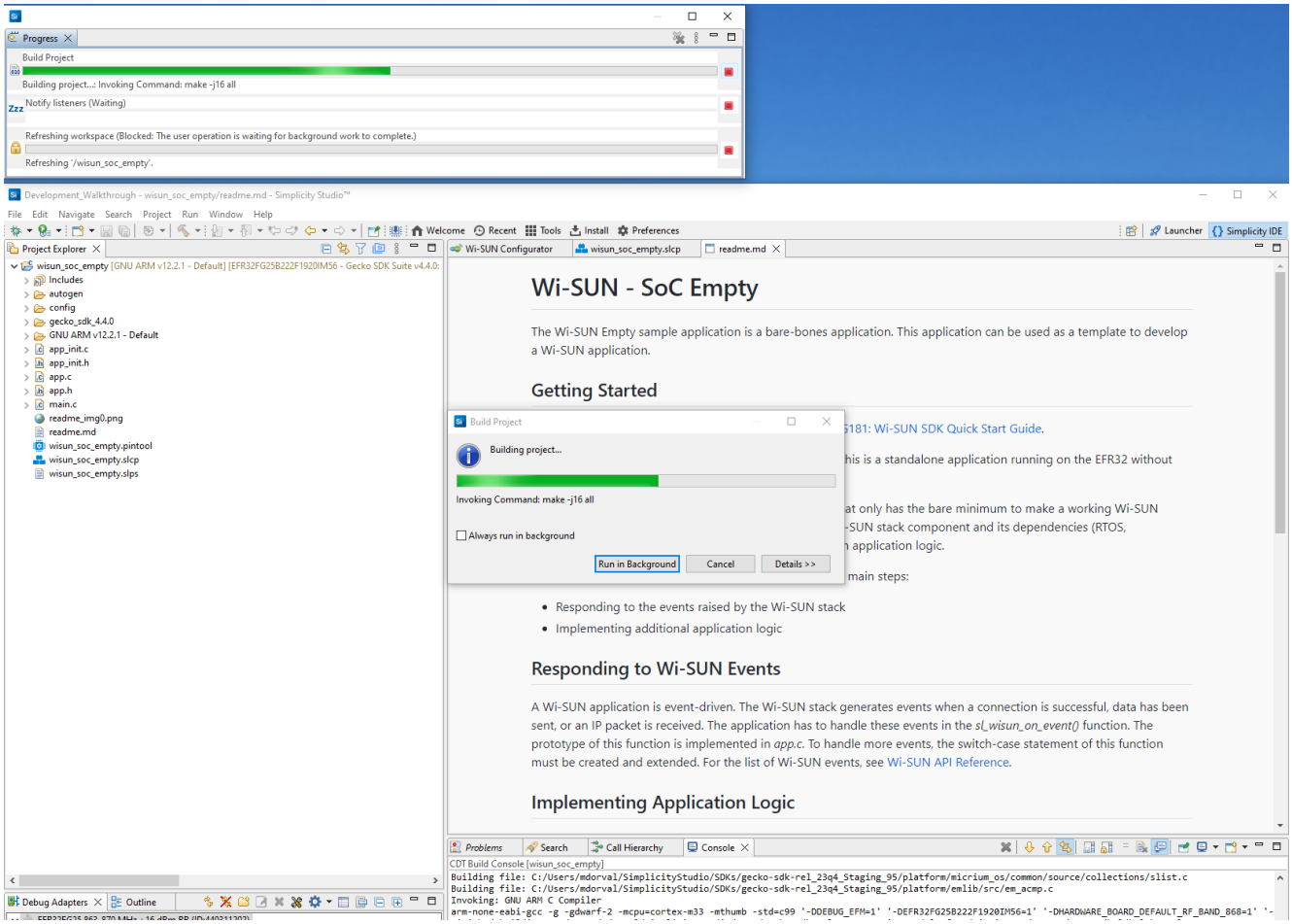
> TIP: Open the **Progress** view to check what Simplicity Studio is working on, and check when compilation is complete. Undock this window and move it next to Simplicity Studio to see progress at any time, while reducing the window dimensions. For multi-screen users, move it to a second screen.



Time to Build

Use **Project** > **Build Project** to launch your first build of the project.

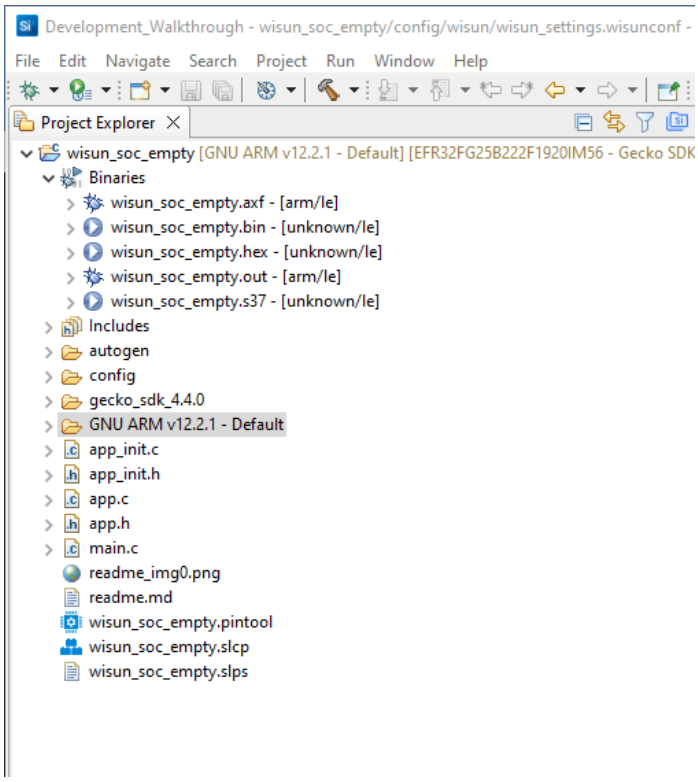




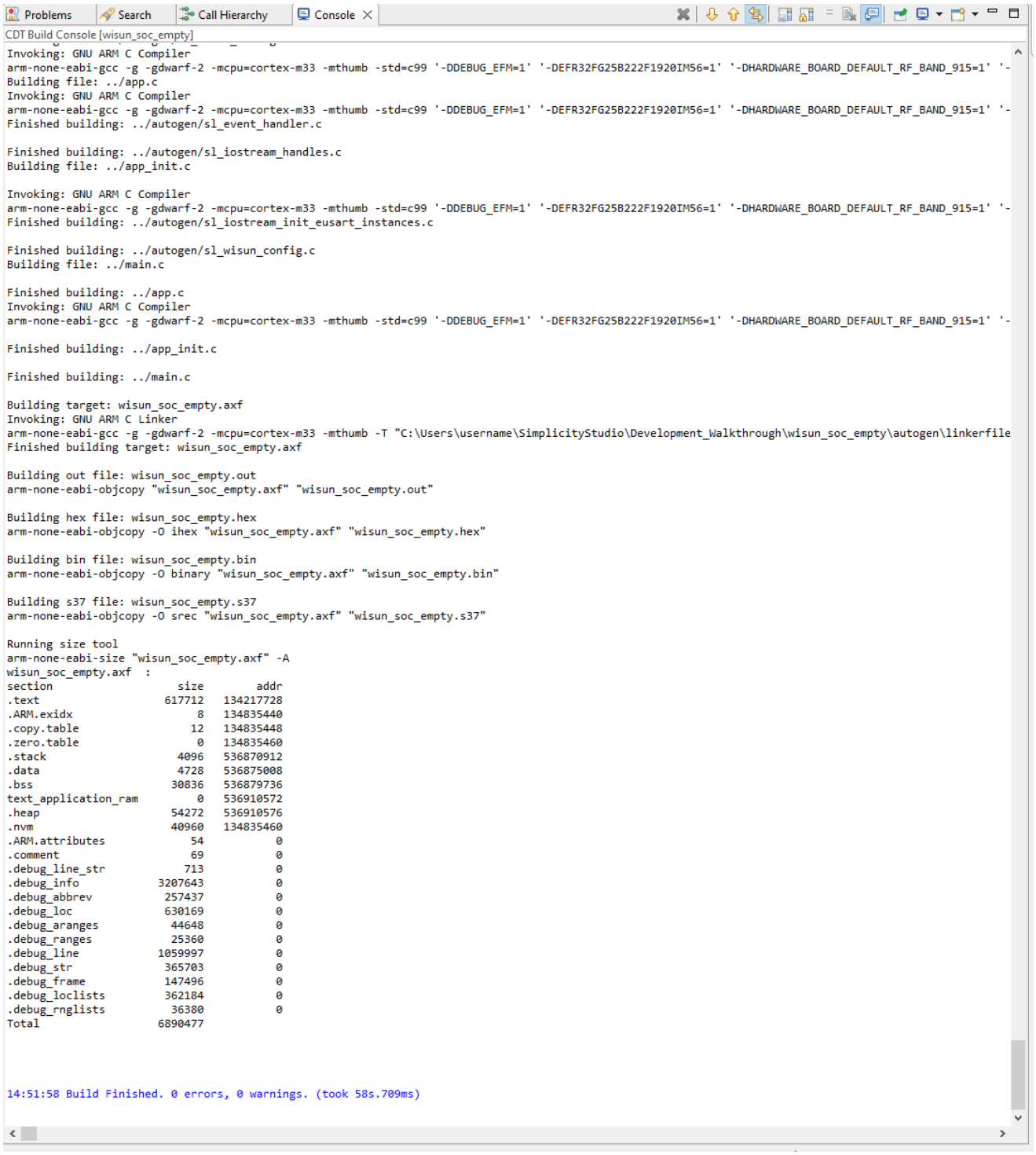
Hopefully, the project compiles without errors. Since this was a first-time full build, it takes approximately 1 min. Subsequent builds are generally smaller, if only adding minor changes to a couple files (only files impacted by the changes are recompiled).

```
14:51:58 Build Finished. 0 errors, 0 warnings. (took 58s.709ms)
```

What Happened



- Two new folders were added to your project folder:
  - `Binaries` contains the executable files in various formats.
  - `GNU ARM vx.y.z - Default` contains all the `.o` files corresponding to the `.c` files compiled. There are a lot of files in there, but you don't need to look at them. It's a storage area for the compiler.
- The **Console** window shows the results of the compilation.
  - This is where you will need to check for any compilation issue.



You can see all 'build' commands (there are many, one per .c files) starting with:

```
Invoking: GNU ARM C Compiler
arm-none-eabi-gcc ...
```

You can see the 'link' command (there is only one link command) starting with:

```
Invoking: GNU ARM C Linker
arm-none-eabi-gcc ...
```

You can see the creation of all binary files (those stored in `Binaries`):

```
Building target: wisun_soc_empty.axf
Invoking: GNU ARM C Linker
arm-none-eabi-gcc -g3 -gdwarf-2 -mcpu=cortex-m33 -mthumb ...
Finished building target: wisun_soc_empty.axf

Building out file: wisun_soc_empty.out
arm-none-eabi-objcopy "wisun_soc_empty.axf" "wisun_soc_empty.out"

Building hex file: wisun_soc_empty.hex
arm-none-eabi-objcopy -O ihex "wisun_soc_empty.axf" "wisun_soc_empty.hex"

Building bin file: wisun_soc_empty.bin
arm-none-eabi-objcopy -O binary "wisun_soc_empty.axf" "wisun_soc_empty.bin"

Building s37 file: wisun_soc_empty.s37
arm-none-eabi-objcopy -O srec "wisun_soc_empty.axf" "wisun_soc_empty.s37"
```

Last, a check is done on the code size using the `arm-none-eabi-size` tool:

```
Running size tool
arm-none-eabi-size "wisun_soc_empty.axf" -A
wisun_soc_empty.axf  :
section               size      addr
.text               617712  134217728
.ARM.exidx               8  134835440
.copy.table             12  134835448
.zero.table              0  134835460
.stack                4096  536870912
.data                 4728  536875008
.bss                 30836  536879736
text_application_ram      0  536910572
.heap                54272  536910576
.nvm                 40960  134835460
.ARM.attributes         54          0
.comment                69          0
.debug_line_str        713          0
.debug_info        3207643          0
.debug_abbrev       257437          0
.debug_loc          630169          0
.debug_aranges       44648          0
.debug_ranges        25360          0
.debug_line        1059997          0
.debug_str          365703          0
.debug_frame        147496          0
.debug_loclists     362184          0
.debug_rnglists      36380          0
Total              6890477
```

> TIP: To find the FLASH footprint of your application, add `.text` to `.data`. For the RAM usage, add `.bss` to `.data` (`.data` is used in both cases since it is copied from FLASH to RAM before execution).

You can run this same command on the `.axf` file whenever you want. `arm-none-eabi-size.exe` is normally installed with Simplicity Studio, under `C:\SiliconLabs\SimplicityStudio\v5\developer\toolchains\gnu_arm\12.2.rel1_2023.7\bin` (the `gnu_arm` release may vary).

> TIP: To get the values displayed in hex, use the '-x' flag. This can be convenient to understand where each piece of code will reside in Flash.

Checking the Linker Command

In the linker command, you can see several interesting things.

The command line is very long, so only an abstract is provided below, with line breaks for easier reading.

```
Invoking: GNU ARM C Linker
arm-none-eabi-gcc
 -g3
 -gdwarf-2
 -mcpu=cortex-m33
 -mthumb
 -T "C:\Users\username\SimplicityStudio\Development_Walkthrough\wisun_soc_empty\autogen\linkerfile.ld"
-Xlinker
 --gc-sections
  -Xlinker
  -Map="wisun_soc_empty.map"
  -mfpu=fpv5-sp-d16
  -mfloat-abi=hard
  --specs=nano.specs
-o wisun_soc_empty.axf
-Wl,--start-group "./app.o" "./app_init.o" "./main.o"... <all `.o` files having been previously compiled from `.c` files>
 "C:/Users/username/SimplicityStudio/SDKs/gecko_sdk/platform/emdrv/nvm3/lib/libnvm3_CM33_gcc.a"
 "C:/Users/username/SimplicityStudio/SDKs/gecko_sdk/platform/radio/rail_lib/autogen/librail_release/librail_efr32xg25_gcc_release.a"
 "C:/Users/username/SimplicityStudio/SDKs/gecko_sdk/protocol/wisun/stack/libwisun_router_efr32xg2x_micriumos_gcc_debug.a"
  -lgcc -lc -lm -lnosys -Wl,--end-group -Wl,--start-group -lgcc -lc -lnosys -Wl,--end-group
```

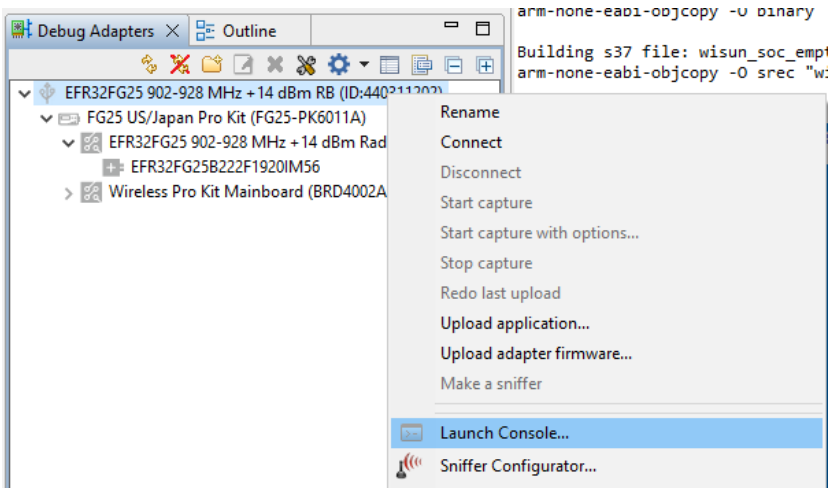
- All linker configuration flags (refer to GCC GNU DOC for details on these flags).
  - Among these, you see that, by default, Simplicity Studio compiles with `-g3`, which means 'optimize for debugging'.
- The name of the linked file, here `-o wisun_soc_empty.axf`.
  - You see in the Console output that the other binary files are copied from this file using various `arm-none-eabi-objcopy` options.
- A very long list of all `.o` files, compiled from your project's `.c` files.
- Three `.a` files matching your project's configuration. These correspond to pre-compiled libraries found in various libraries:
  - The NVM library `libnvm3_CM33_gcc.a`
  - The RAIL library for your hardware `librail_efr32xg25_gcc_release.a`
  - The Wi-SUN library managed by Silicon Labs R&D: `libwisun_router_efr32xg2x_micriumos_gcc_debug.a`
    - Looking in `C:/Users/username/SimplicityStudio/SDKs/gecko_sdk/protocol/wisun/stack/`, you see that there is a dedicated pre-compiled library per device_type/part/OS/Compiler/debug_or_release. Here, you linked with the library matching your project's default configuration.

Flashing the Binary to the Adapter Board

To run the binary you just compiled, you need to flash (copy) it to your Radio Board, with the following steps.

Open the Serial 1 UART Console

1. In the **Debug Adapters** frame, select the Radio Board.
2. Select **Launch Console** to open a communication console with the Adapter Board.

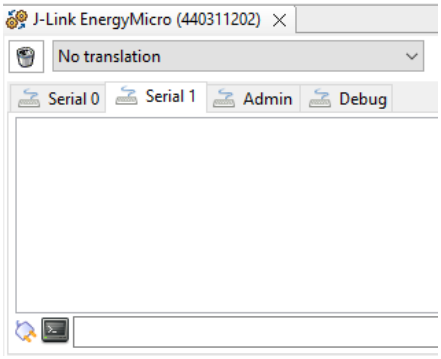


A new tab will open, with the name of the J-Link adapter and it's ID.

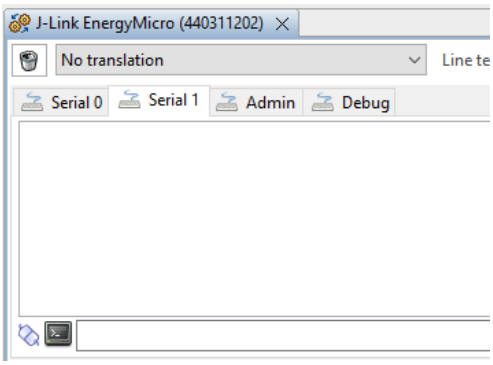
3. In this new tab, select the **Serial 1** tab to access the application's UART console. This will show all strings printed from the application using `printf()`.

> TIP: Strings printed by the Wi-SUN Stack will not be sent to the **Serial 1** console. These are accessible as RTT traces, using J-Link RTT Viewer.

When the **Serial 1** console opens, it's in a non-connected state by default, as shown by the small icon in the bottom left corner.



4. Select the console's text area and press **Enter** to connect to the application. The icon changes to indicate a connected state.



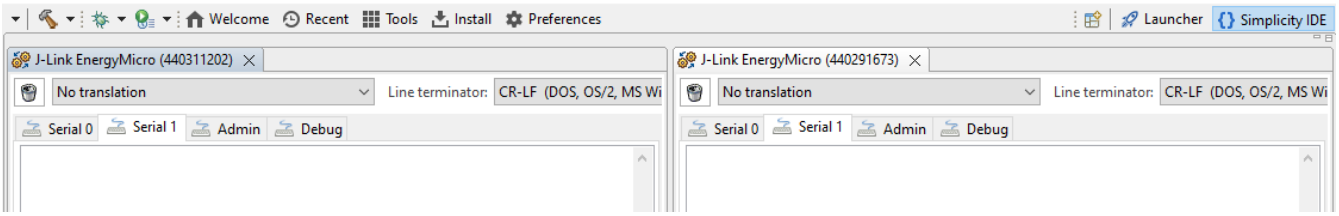
The Console is now connected. You can flash your application, but you need to start a Border Router first to be able to connect over your Wi-SUN network.

> TIP: Having the Serial 1 console ready to print the application output is important to check messages indicating proper application start. Most example applications will display the application's name at startup.

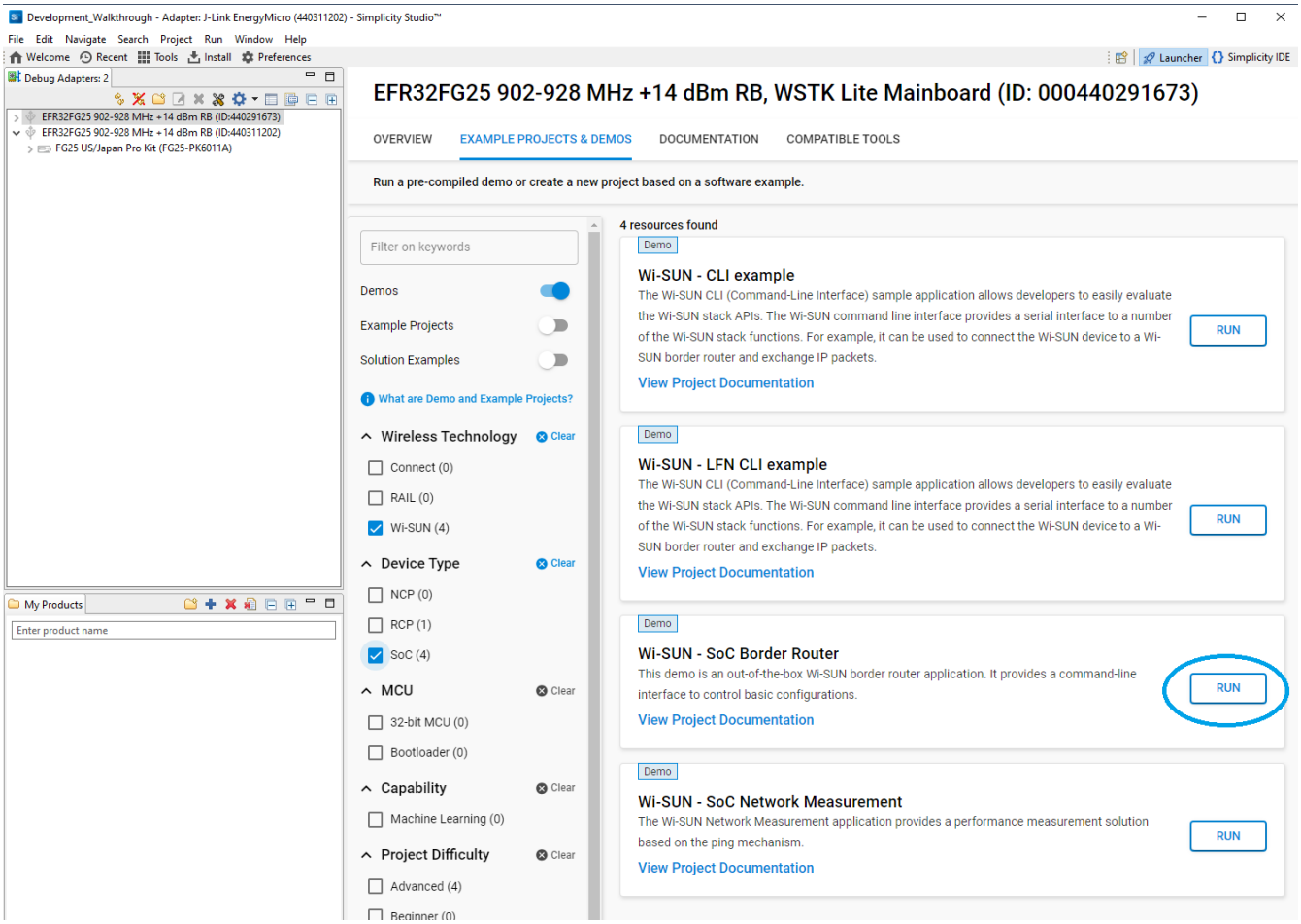
Time to Start a Border Router Demo

To make your development walkthrough more pleasant, it's time now to start a Border Router with default settings, such that your Empty application will be able to connect to an existing Wi-SUN network.

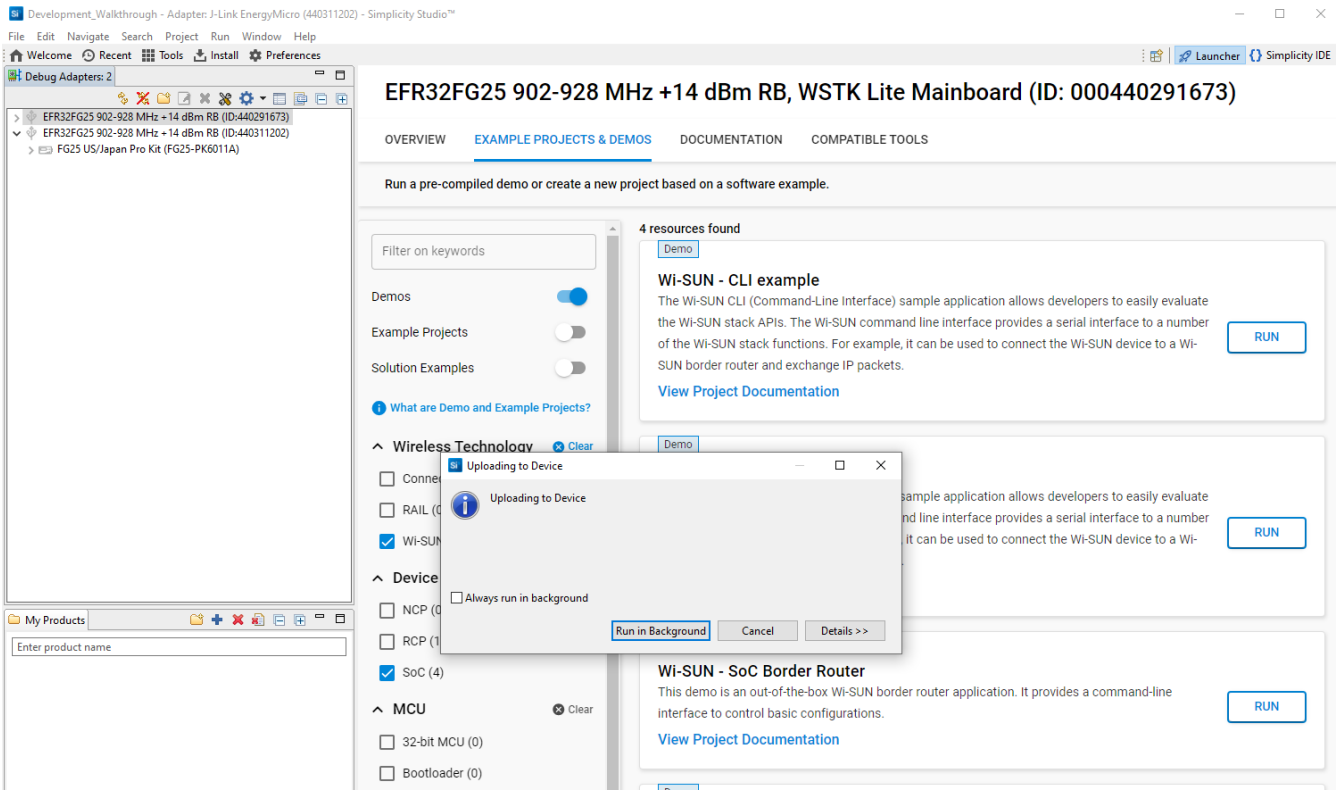
1. Plug a second Development Kit.
2. Select it in the **Debug Adapters** view.
3. As above, open the **Serial 1** console for this Development Kit.
4. Drag and drop the new console to the left side of the first one to see them side by side.



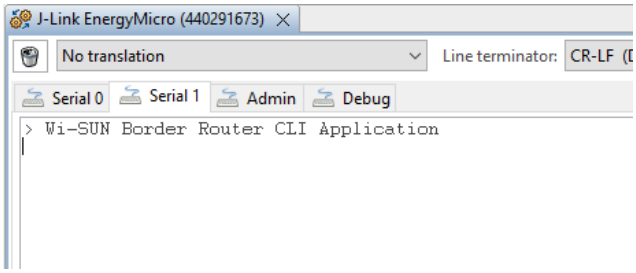
5. Click **Enter** in the new console to connect the UART.
6. Click **Launcher** at the top right to open the Launcher perspective.
7. Select the **EXAMPLE PROJECTS AND DEMOS**.
8. Leave only **Demos** selected.
9. Filter on **Wi-SUN**.
10. Filter on **SoC**.
    Only three applications match.
11. Move to the **Wi-SUN - SoC Border Router** frame and click **RUN**.



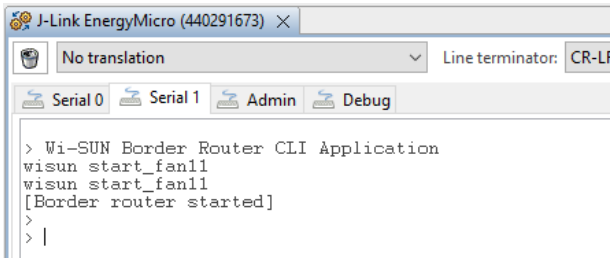
This immediately triggers flashing the corresponding pre-compiled binary to the target.



12. Go back to the **Simplicity IDE** perspective.
13. Check the Border Router Demo console, which should show the Border Router CLI initial message.

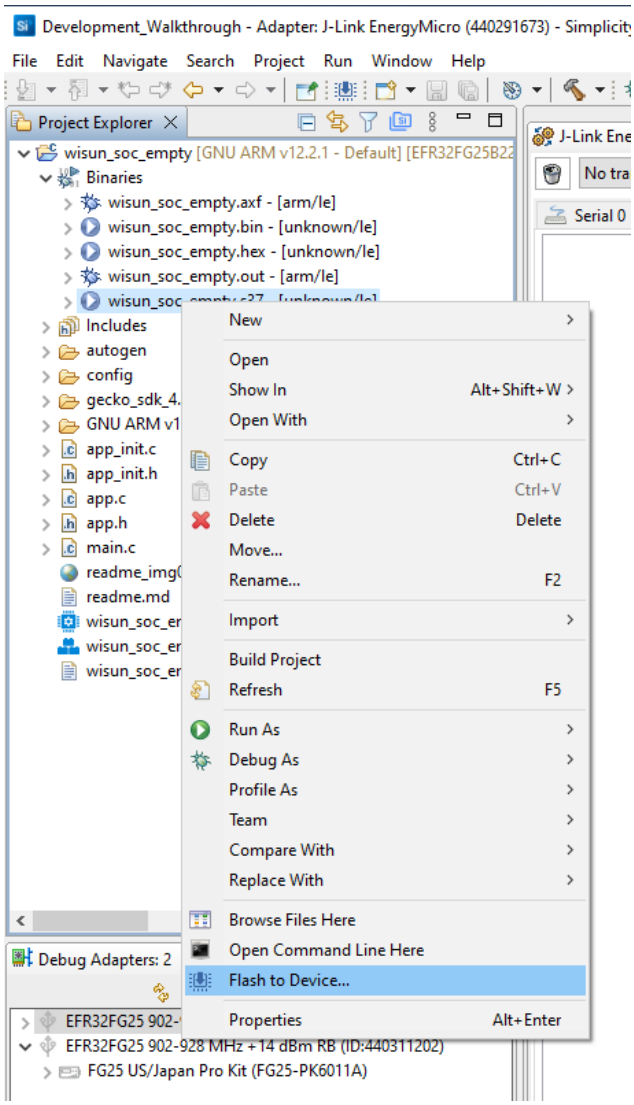


14. In this console, use `wisun start_fan11` to start the Border Router with default FAN1.1 settings.



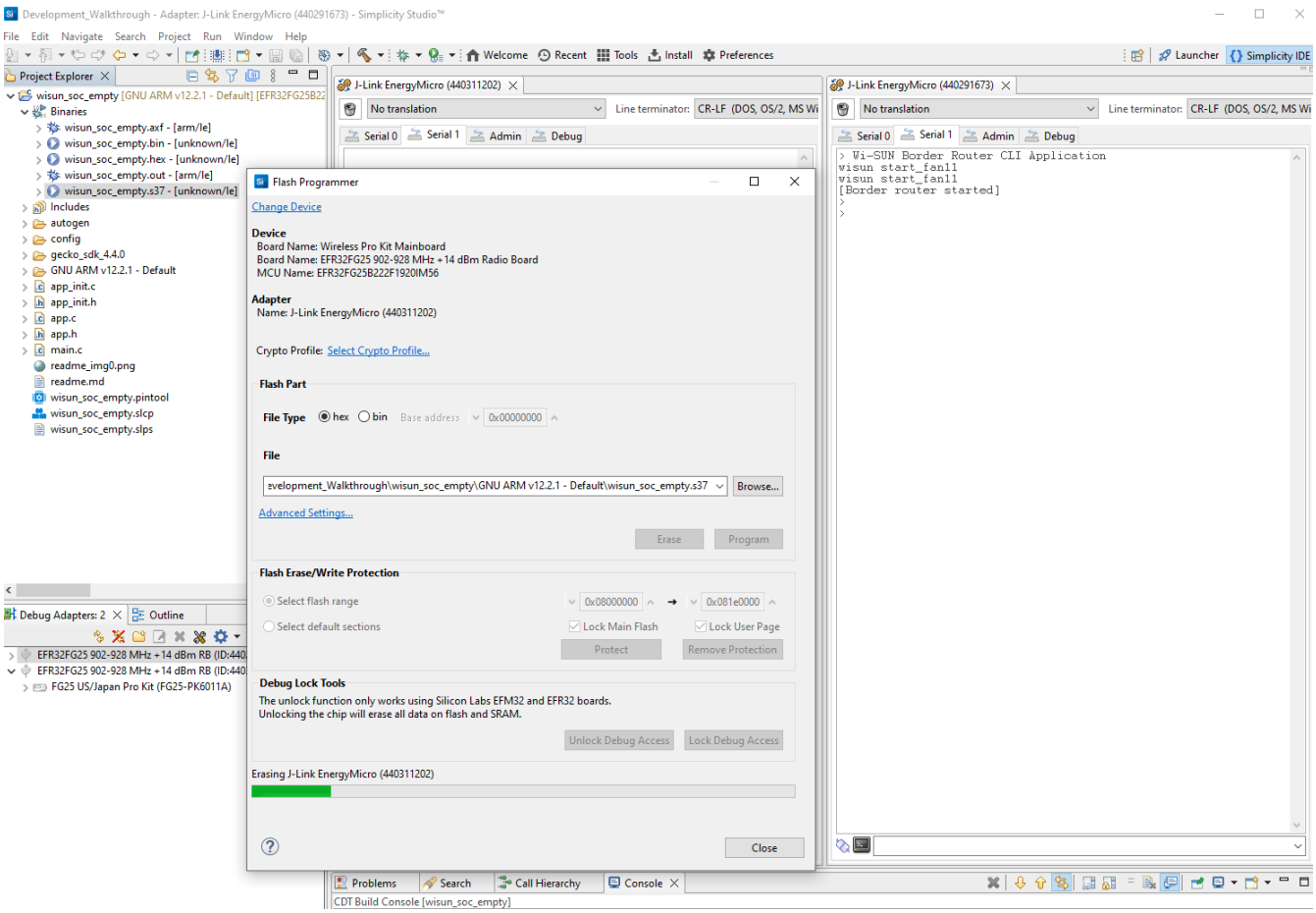
Flashing the Binary

1. In the **Project Explorer** view select the wisun_soc_empty project, open the `Binaries` folder and right-click the `.s37` file.
2. Select **Flash To Device**.

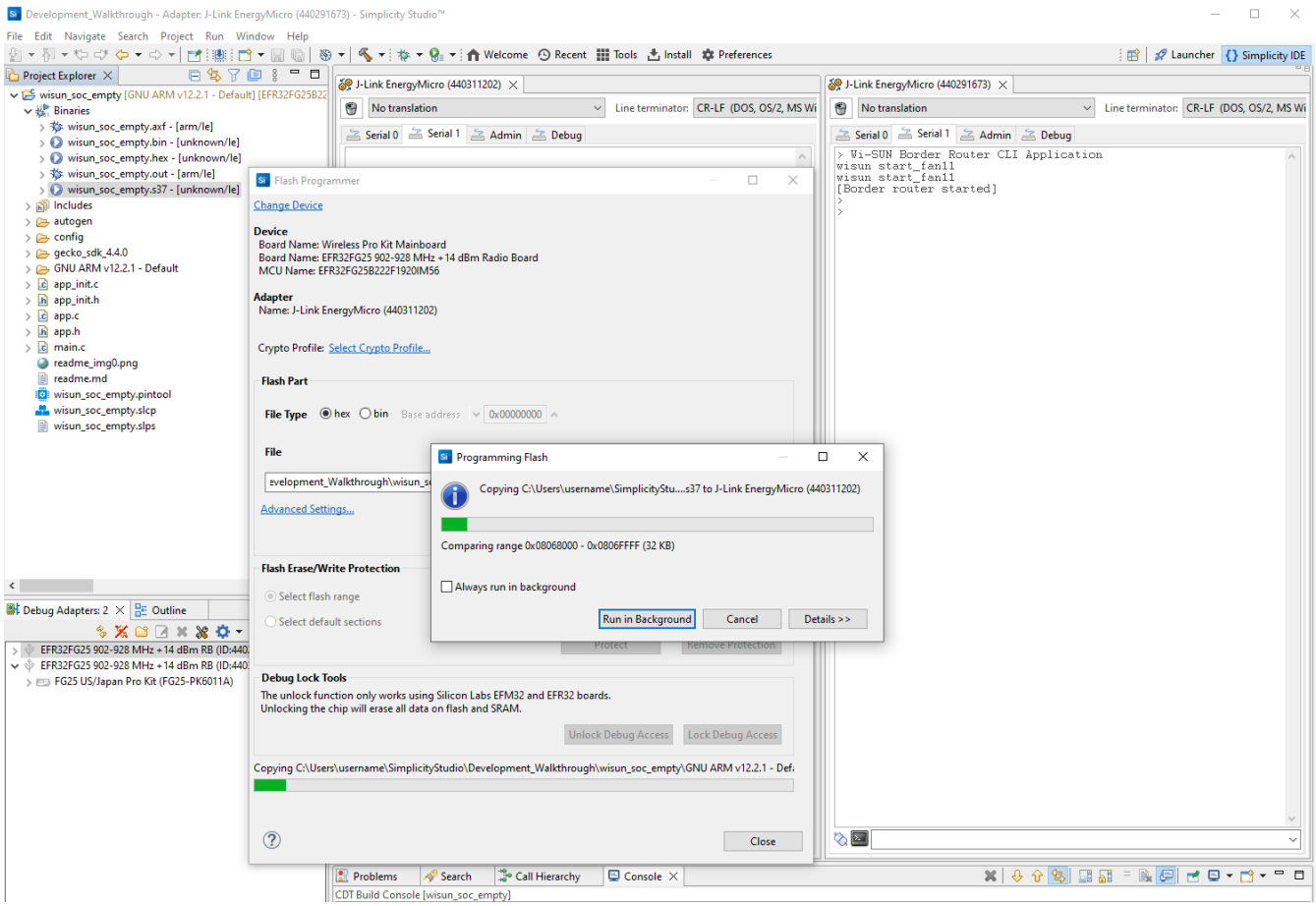


3. Select the device. Make sure not to select the one used as a Border Router.

> TIP: For a first flash of a new application on a Radio Board where other application may have been running before, use the **Erase** button to clear any data possibly stored in flash. A progress bar appears and closes when done.



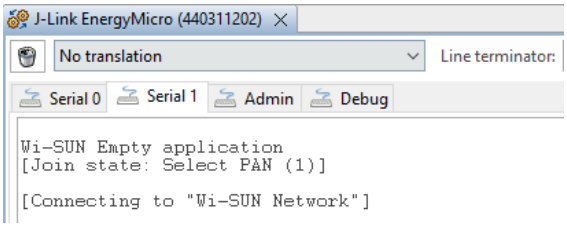
4. Use the **Program** button to flash the selected binary. A progress bar appears and closes when done.



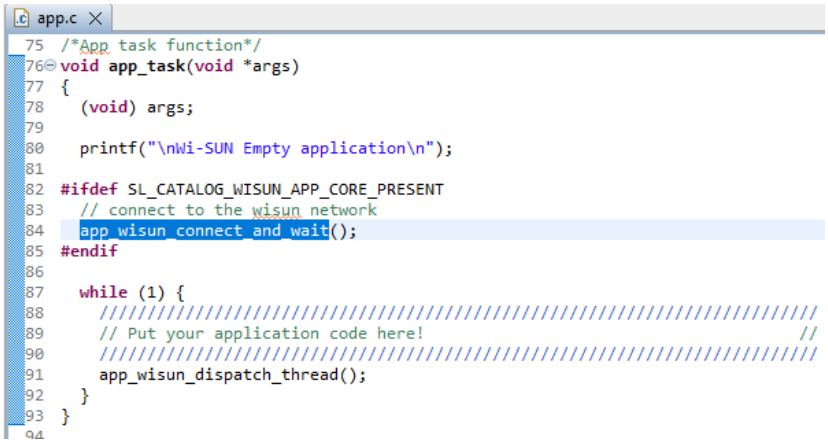
5. Close the Flash Programmer window.

Verifying Console Output

The Device Console now shows the startup message.



- The name of the application is printed.
  - This comes from `app.c/app_task()` , the entry point for the application code.



- Then, you see the first message generated by the application: `[Join state: Select PAN (1)]`
  - This comes from the **Application Core** component, which is tracking join state change events.
  - A similar message will be printed every time the join state will change.

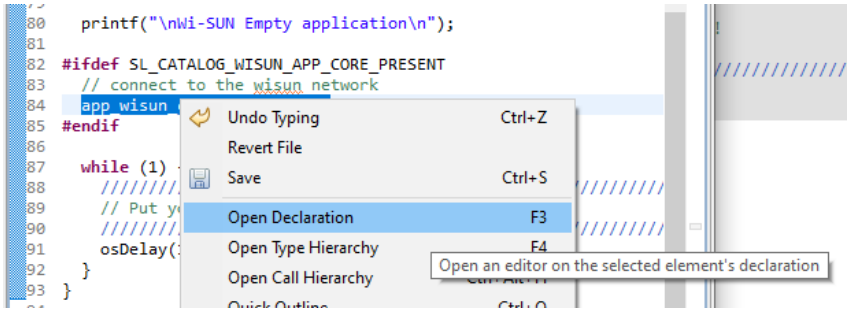
> TIP: When a component is present in the project, the corresponding code is shown with normal colors. If a component is not present, the corresponding code is grayed out in the editor, meaning that it is not compiled and can be ignored.

- Here, the fact that the `sl_wisun_on_event()` function (lines 32-72) is grayed out indicates that this code is not compiled, meaning that `SL_CATALOG_WISUN_EVENT_MGR_PRESENT` is not defined.
  - This means that the **Wi-SUN Event Manager** component is not installed in your project.
- Similarly, the fact that the `app_wisun_connect_and_wait()` function call (line 84) is displayed with normal colors indicates that this code is compiled, meaning that `SL_CATALOG_WISUN_APP_CORE_PRESENT` is defined.
  - This means that the **Wi-SUN Application Core** component is installed in your project.
- Last, you see a message coming from the application indicating the name of the Wi-SUN network that you are trying to connect to.
- If a Wi-SUN Border Router is within range with a Wi-SUN network named `Wi-SUN Network` and the same PHY settings as your application defaults, you will connect.

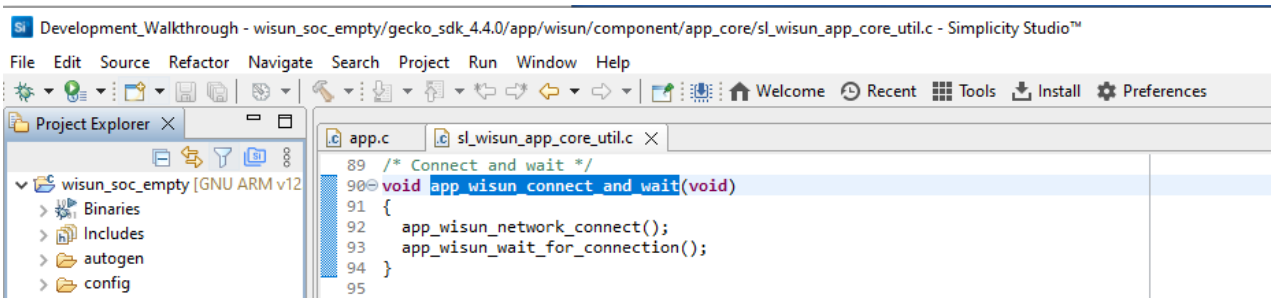
Tracking the Connection Message

For users not familiar with Simplicity Studio and C code in general, the following are some elementary code checks:

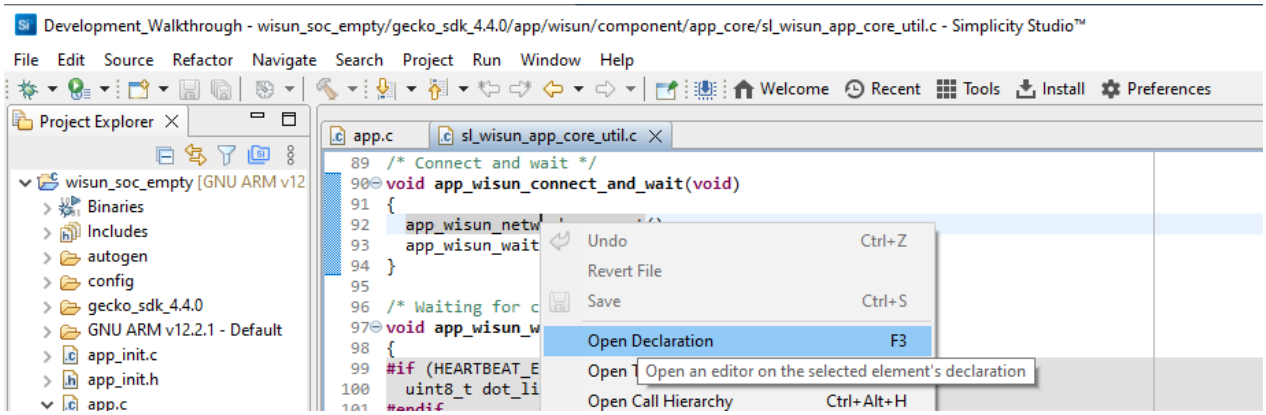
- In `app.c/app_task()`, you see that there is a call to `app_wisun_connect_and_wait();` on line 84.
- Select `app_wisun_connect_and_wait` and right-click to get access to the declaration of this function.



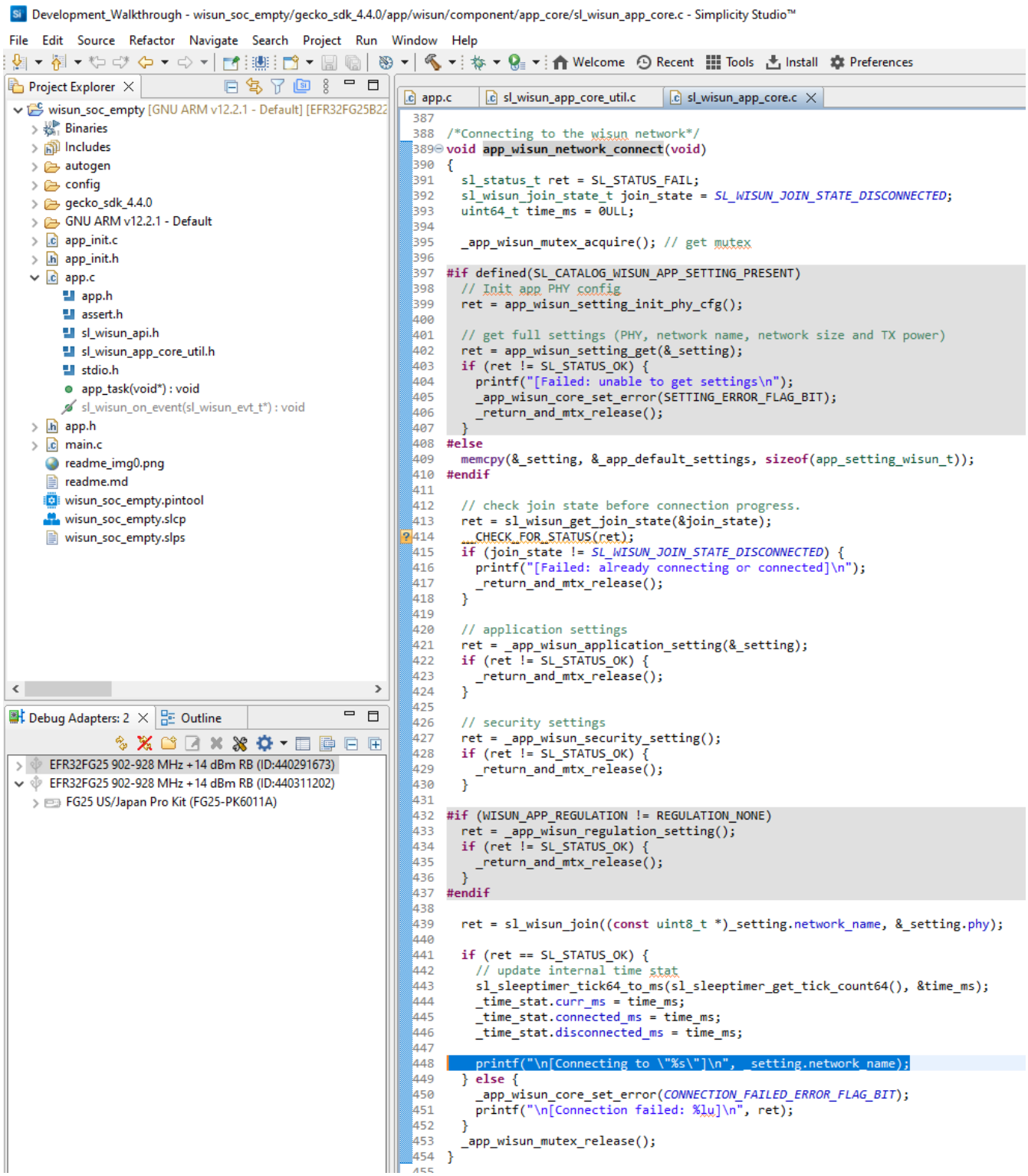
- You end up in `/wisun_soc_empty/gecko_sdk_x.y.z/app/wisun/component/app_core/sl_wisun_app_core_util.c`, in the code implementing `app_wisun_connect_and_wait()` on lines 90-94.



- You find two function calls here. Given the function names, you get a pretty good idea of what they do. Since the **Connecting to...** message probably comes from `app_wisun_network_connect()`, look for it's declaration as well.



- You end up in `/wisun_soc_empty/gecko_sdk_x.y.z/app/wisun/component/app_core/sl_wisun_app_core.c/app_wisun_network_connect()`, which is a longer function (lines 389-454).



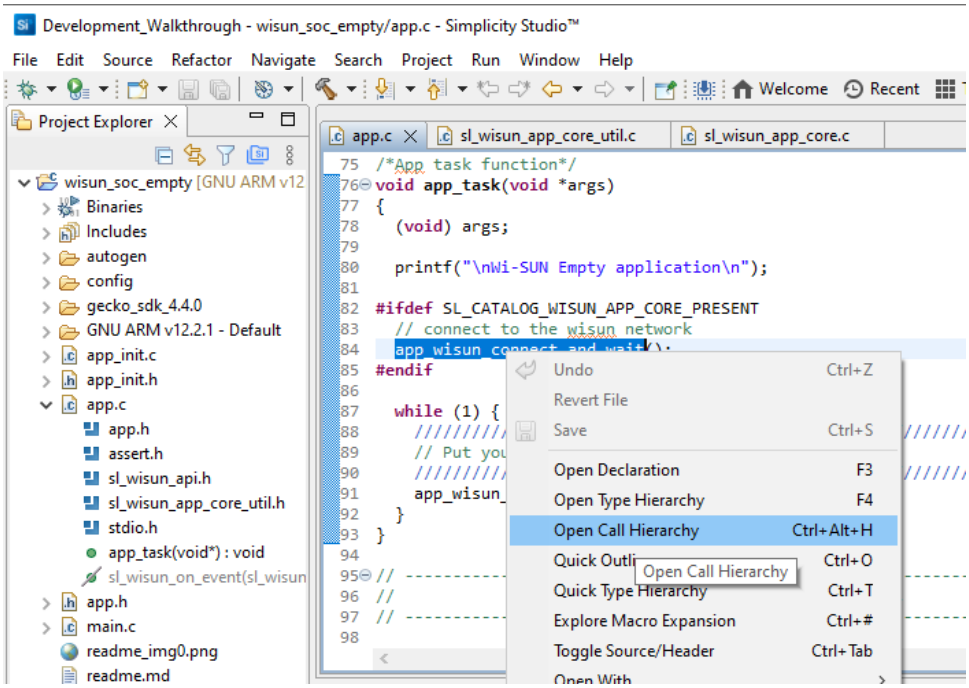
- You can find the `Connecting to` message on line 448.

While you are looking at `app_wisun_network_connect()`, you can see that there are several steps during the connection preparation:

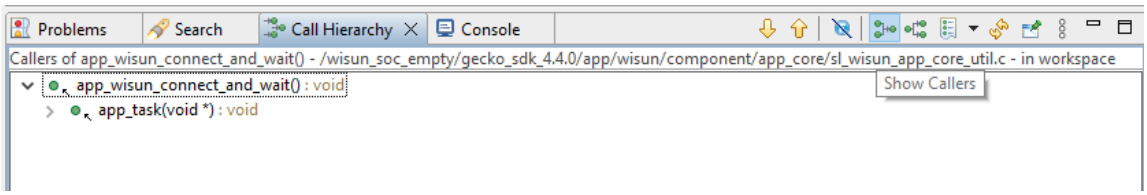
- Preparation of the PHY and network settings
- Check of the current join state
- Setting the network settings (to your project defaults)
- Setting Security

- So far, rely on the application components (i.e. Wi-SUN Application Core here) to do that.
- More details on this later.

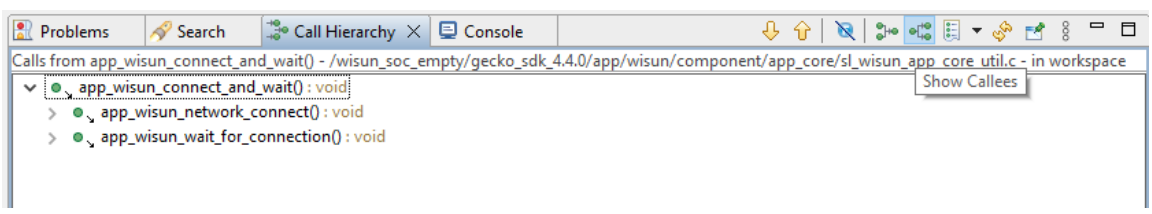
- An alternative method to follow the function calls is to use the **Open Call hierarchy** for the `app_wisun_connect_and_wait()` function.



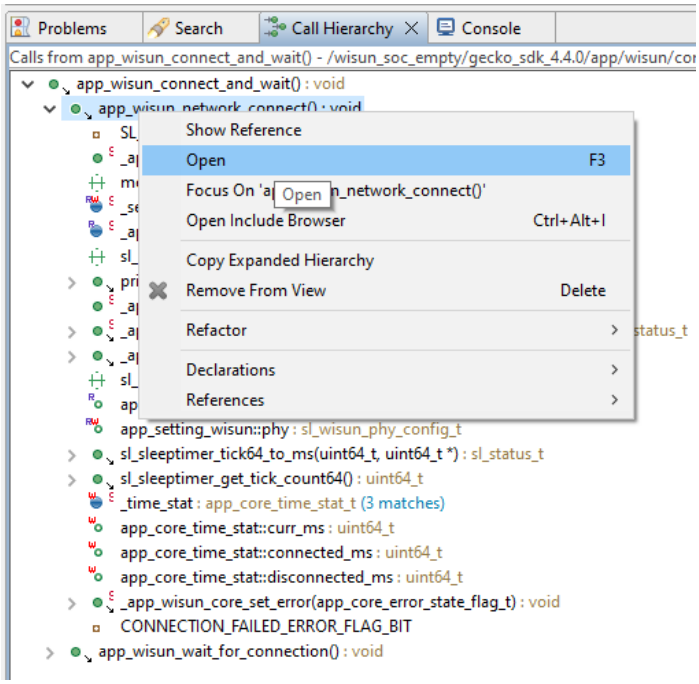
- The Call Hierarchy view proposes two modes, selectable using the 'tree' icons.
  - The **Show Callers** view shows which functions are calling `app_wisun_connect_and_wait()` :
    - Here, a single function: `app_task()` , as you've already seen.
    - Using the Call Hierarchy, you are now sure that `app_wisun_connect_and_wait()` is called from a single location, information you didn't have before.



  - The **Show Callees** view shows which functions are called by `app_wisun_connect_and_wait()` :



    - Here, two functions are called, as you've already seen:
      - `app_wisun_network_connect()`
      - `app_wisun_wait_for_connection()`
      - Using the Call Hierarchy, you can expand `app_wisun_network_connect()` and check which functions it calls.



- This view still shows that `app_wisun_network_connect()` calls `printf`, the function called to print your message.

> TIP: Depending on the user, everyone prefers a method to find the call hierarchy based on personal preferences and the situation. The method above proposes two ways to follow the function calls in a Wi-SUN application.
> TIP: When going down the call hierarchy, you will end in a situation where you can't go further and are left with the definition of a given function in a header file. It means that you have reached the tip of the Wi-SUN Stack API, that part of the Wi-SUN code which is under the responsibility of Silicon Labs R&D and where you don't have access to the source code.

GSDK Code vs User Code

> TIP: All files under `/wisun_soc_empty/gecko_sdk_x.y.z/` are GSDK files, common to all projects. Consider that for all your projects, `/<project_name>/gecko_sdk_x.y.z/` is the same content. This is why there are blocks of code conditionally compiled depending on the project's configuration (settings and components). These files should normally not be modified. User code and changes should stay out of `<project_name>/gecko_sdk_x.y.z/`, since changing this code will impact all your projects the next time you build them.

What Has Changed

If you started using Git, it's time to check the changes and commit the current state before proceeding.

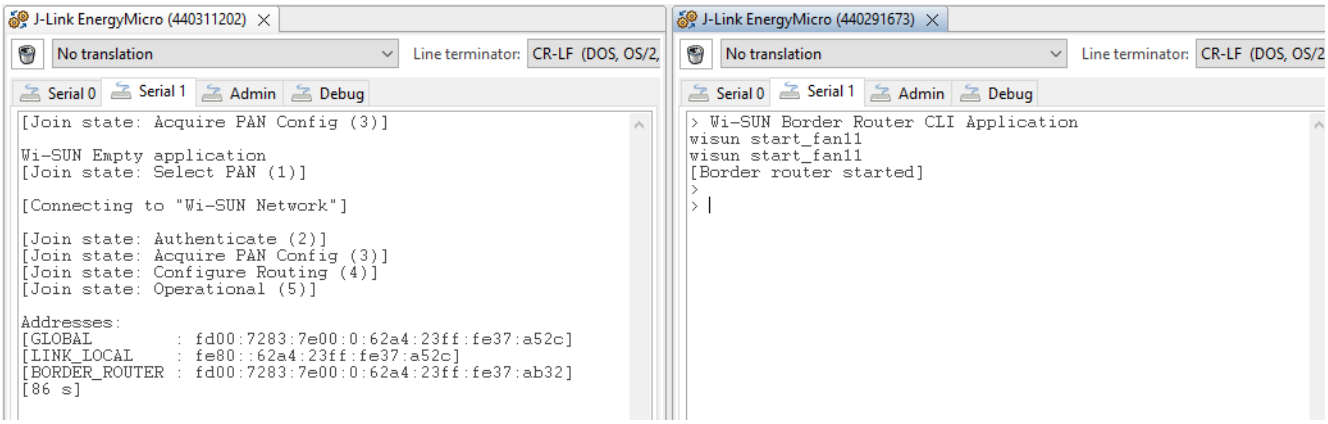
Using `git status`, you see that no change has been made to the settings and source files. Since, thanks to the `.gitignore` file, many files generated at compilation time are not tracked (they would otherwise change on most commits), you can use Git to track your settings changes.

Use:

- `git add --all` to 'stage' the changes (i.e. get them ready for committing).
- `git commit -m "All defaults"` to commit the changes (i.e save them to the repository) with a commit message.

Checking the Wi-SUN Connection

While you were looking at the source code, time passed and there is a good chance that your device connected to the Wi-SUN Border Router. This should be visible in the Device's Serial 1 console.



- The Wi-SUN connection took 86 seconds. This is extremely variable in a Wi-SUN environment, and depends on the network settings. Detailed connection times are provided in AN1330: Silicon Labs Wi-SUN Mesh Network Performance.
- Once connected, the IPv6 addresses are displayed:
  - GLOBAL is the device's IPv6 to connect to it from anywhere, provided that IPv6 forwarding is allowed on the Border Router, which is not the case with the Border Router SoC Demo. To test pinging the device from the Border Router, Silicon Labs recommends our open source Linux Border Router.
  - LINK_LOCAL is the device's IPv6 address to communicate with its neighbors.
  - BORDER_ROUTER is the Border Router's IPv6 address. You can check this on the Border Router Demo using 'wisun get wisun.ip_addresses'. It is listed as the 'dodagid' address.

What's Next

If you achieved a Wi-SUN connection, everything is fine in terms of Wi-SUN connectivity. The Wi-SUN stack will now maintain the connection, using the self-healing features of Wi-SUN to select an alternative path if the current parent fails or if there is a better option.

You can flash exactly the same application to many similar Adapter Boards. They will all end up connected to the Wi-SUN network. If you move the devices apart or you start more devices than what the Border Router can support directly, they will start creating hops to connect to the Border Router.

Basically, you have a working Wi-SUN network where you can add your custom application code to do whatever you need to.

There are still, however, a number of limitations in this approach:

- All devices use the same security credentials. In production devices, each device should have its own set of device key/device certificate.
- You didn't control the network name.
- You used the default PHY settings (with lowest throughput).
- You may not be using a PHY matching your region, so you may be transmitting in a forbidden frequency band.

It's time to learn how to check and change the settings, and then how to add your custom application.

Checking and Changing the Settings

Border Router Settings

To illustrate how to change the settings, using something different from the defaults regulatory_domain/chan_plan_id/phy_mode_id (NA/1/2 in our example), you will use BZ/3/8 and name your network 'BZ_3_8'.

> Note: The previous steps are valid for all Wi-SUN development kits, with default settings, because Simplicity Studio selected the proper libraries to match the Radio Boards. There are several frequency ranges used for Wi-SUN, with only some valid for a given country. The first part of the Development Walkthrough used Radio Boards with the 902-928 MHz frequency range valid for North America (NA), Mexico (MX), and Brazil (BZ). Selecting BZ/3/8 is only possible when using Radio Boards for these regions. With 'EU' Radio Boards or any other region, it is necessary to select a valid PHY for the Radio Boards to go through the following part.

Using the Wi-SUN Border Router Demo, use the following to change the settings.

Hardware reset on the Border Router (to stop the Border Router):

```
> Wi-SUN Border Router CLI Application
wisun get wisun.state
wisun.state = initialized (0)
```

> TIP: If the wisun.state is not `initialized (0)` at this step, the new settings won't be applicable. Make sure you reset the Border Router.

```
> wisun set wisun.regulatory_domain BZ
wisun.regulatory_domain = BZ
> wisun set wisun.chan_plan_id 3
wisun.chan_plan_id = 3
> wisun set wisun.phy_mode_id 8
wisun.phy_mode_id = 8
> wisun start_fan11
[Border router started]
> wisun get wisun.state
wisun.state = operational (1)
```

Wi-SUN Router Settings

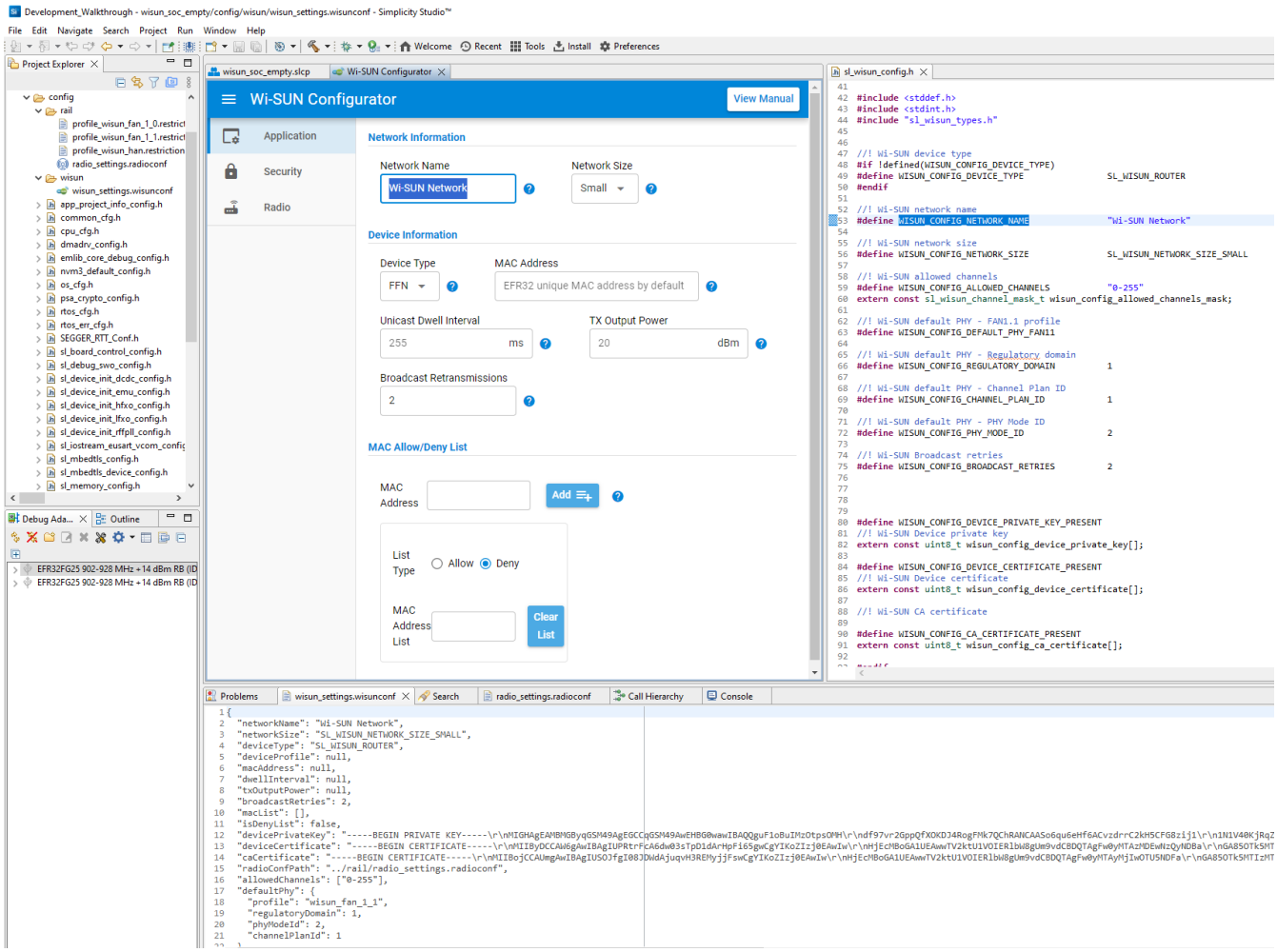
Checks

First observe what you have by default:

1. Open the Wi-SUN Configurator (double-click the `.slcp` file).
2. Open the `autogen/sl_wisun_config.h` file, and move it aside to get a view on both.
3. Open the `config/wisun_settings.wisunconf` file with the option **Open With > Text Editor**, and move it below to see it also.

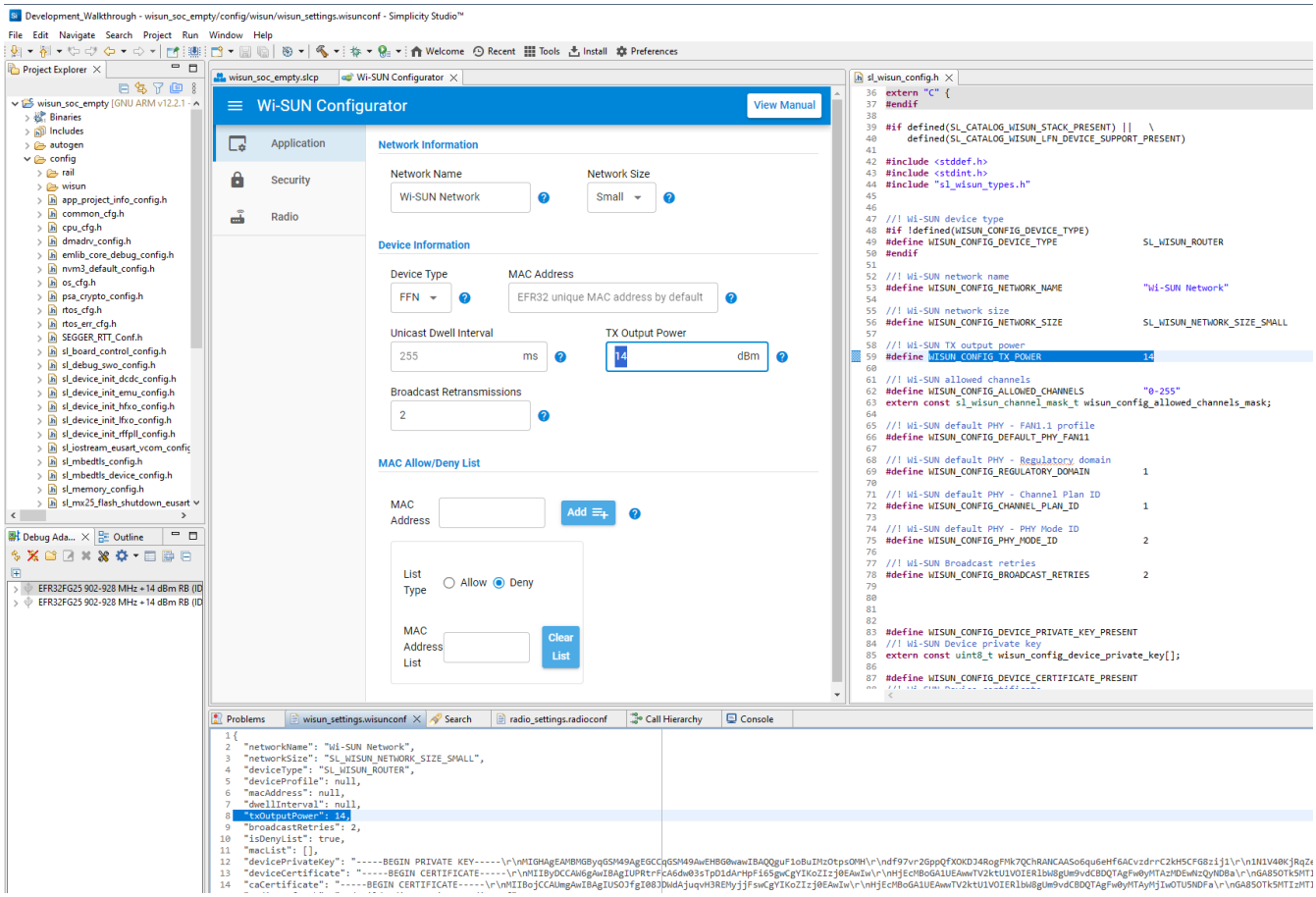
In the Wi-SUN Configurator, the **Application** tab gives access to:

- The network name, which corresponds to `WISUN_CONFIG_NETWORK_NAME` in `autogen/sl_wisun_config.h` and `"networkName"` in `config/wisun/wisun_settings.wisunconf`
- The network size, which corresponds to `WISUN_CONFIG_NETWORK_SIZE` in `autogen/sl_wisun_config.h` and `"networkSize"` in `config/wisun/wisun_settings.wisunconf`
- The device type, i.e. `WISUN_CONFIG_DEVICE_TYPE`
- The broadcast retransmission, i.e. `WISUN_CONFIG_BROADCAST_RETRIES`



No Tx Output Power in the files?

- This is because the default GUI value is set to the code's default value (20 dBm). If you change the Tx Output Power in the GUI, you will see new items added to the files:
  - `WISUN_CONFIG_TX_POWER` in `autogen/sl_wisun_config.h` and `"txOutputPower"` in `config/wisun/wisun_settings.wisunconf`



- A quick search for `WISUN_CONFIG_TX_POWER` will show how this is handled by `gecko_sdk_x.y.z/app/wisun/component/app_core/sl_wisun_app_core.c` :

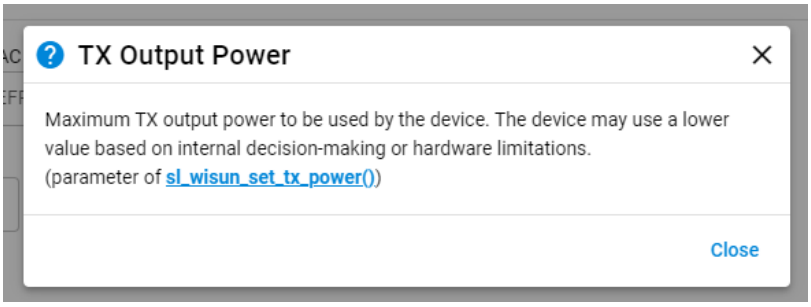
sl_wisun_config.h

```
36 extern "C" {
37 #endif
38
39 #if defined(SL_CATALOG_WISUN_STACK_PRESENT) ||   \
40     defined(SL_CATALOG_WISUN_LFN_DEVICE_SUPPORT_PRESENT)
41
42 #include <stddef.h>
43 #include <stdint.h>
44 #include "sl_wisun_types.h"
45
46
47 //! Wi-SUN device type
48 #if !defined(WISUN_CONFIG_DEVICE_TYPE)
49 #define WISUN_CONFIG_DEVICE_TYPE                SL_WISUN_ROUTER
50 #endif
51
52 //! Wi-SUN network name
53 #define WISUN_CONFIG_NETWORK_NAME               "Wi-SUN Network"
54
55 //! Wi-SUN network size
56 #define WISUN_CONFIG_NETWORK_SIZE               SL_WISUN_NETWORK_SIZE_SMALL
57
58 //! Wi-SUN TX output power
59 #define WISUN_CONFIG_TX_POWER                   14
60
```

sl_wisun_app_core.c

```
160 //                                 Static Variables
161 // -----------------------------------------------------------------------------
162
163 /// Create default setting if app settings is not available
164 #if !defined(SL_CATALOG_WISUN_APP_SETTING_PRESENT)
165 static const app_setting_wisun_t _app_default_settings = {
166 #if defined(WISUN_CONFIG_NETWORK_NAME)
167   .network_name = WISUN_CONFIG_NETWORK_NAME,
168 #else
169   .network_name = "Wi-SUN Network",
170 #endif
171 #if defined(WISUN_CONFIG_NETWORK_SIZE)
172   .network_size = WISUN_CONFIG_NETWORK_SIZE,
173 #else
174   .network_size = SL_WISUN_NETWORK_SIZE_SMALL,
175 #endif
176 #if defined(WISUN_CONFIG_TX_POWER)
177   .tx_power = WISUN_CONFIG_TX_POWER,
178 #else
179   .tx_power = 20U,
180 #endif
181   .is_default_phy = true,
182 #if defined(WISUN_CONFIG_DEVICE_TYPE)
183   .device_type = WISUN_CONFIG_DEVICE_TYPE,
184 #else
```

> TIP: The code default Tx Output Power is 20. As explained by the helptext, when you click the icon, the actual output power is generally less than that. Indeed, for best MCS6 performance (OFDM MCS6 uses a 16-QAM modulation) with EFR32xG25, do not go above 14 dBm to avoid clipping the outer edges of the constellation and get better performances.

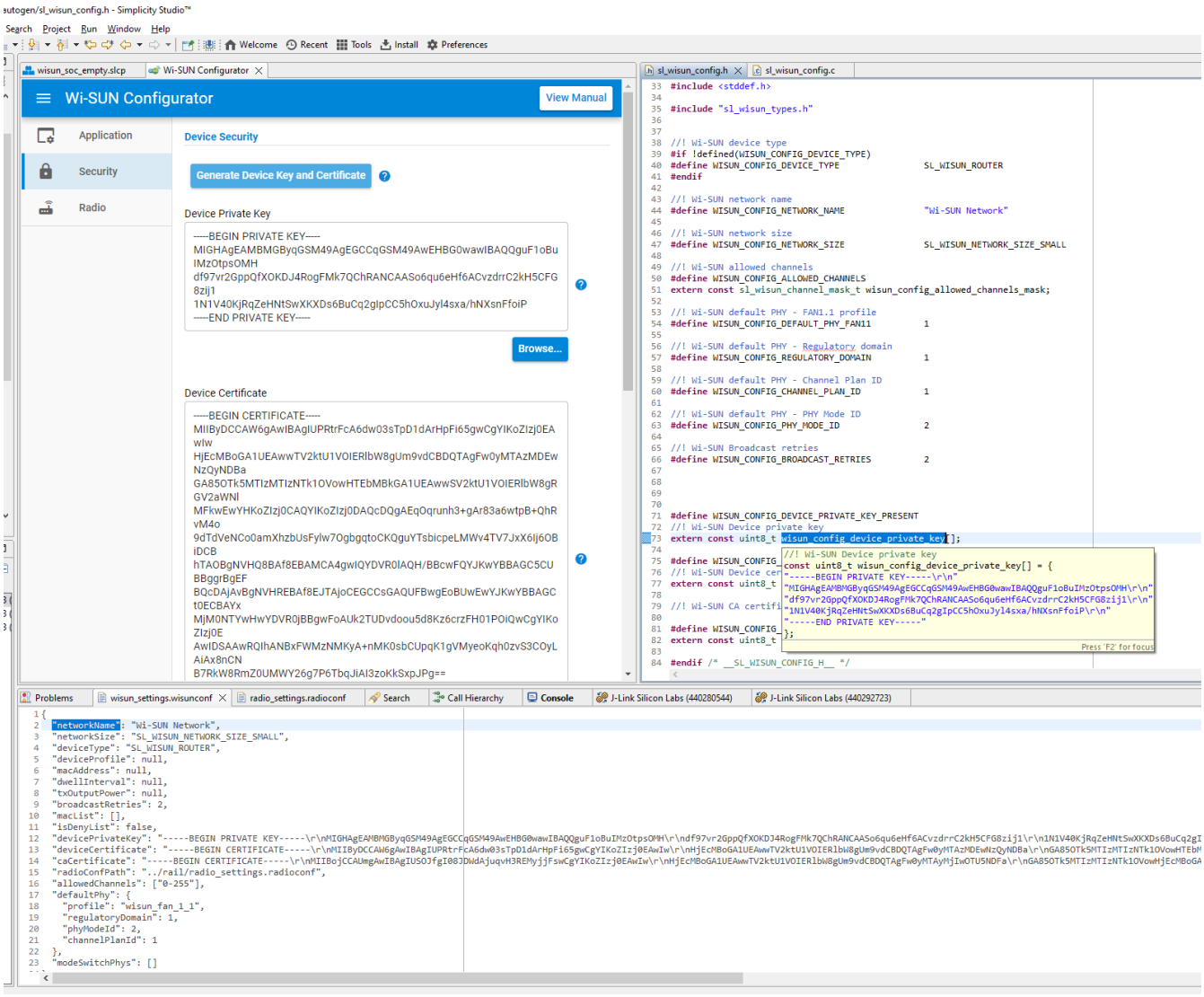


There are close relationships between:

- The GUI (the Wi-SUN Configurator)
- It's control file `config/wisun_settings.wisunconf`
- Some of the `.h` and `.c` files which you will ultimately use to compile your application

The **Security** tab gives access to:

- The Device Private Key, i.e. `wisun_config_device_private_key[];`
  - You can see its current value by hovering over it, as shown above.
  - You can also right-click it and select **Open Declaration**. This opens `autogen/sl_wisun_config.c`, where `wisun_config_device_private_key[]` is declared on lines 47-53.
  - You can check that the text string:
    - Is identical to the one in the GUI
    - Starts with `-----BEGIN PRIVATE KEY-----`
    - Ends with `-----END PRIVATE KEY-----`
- The Device Certificate, i.e. `wisun_config_device_certificate[];`
- The CA Certificate, i.e. `wisun_config_ca_certificate[];`



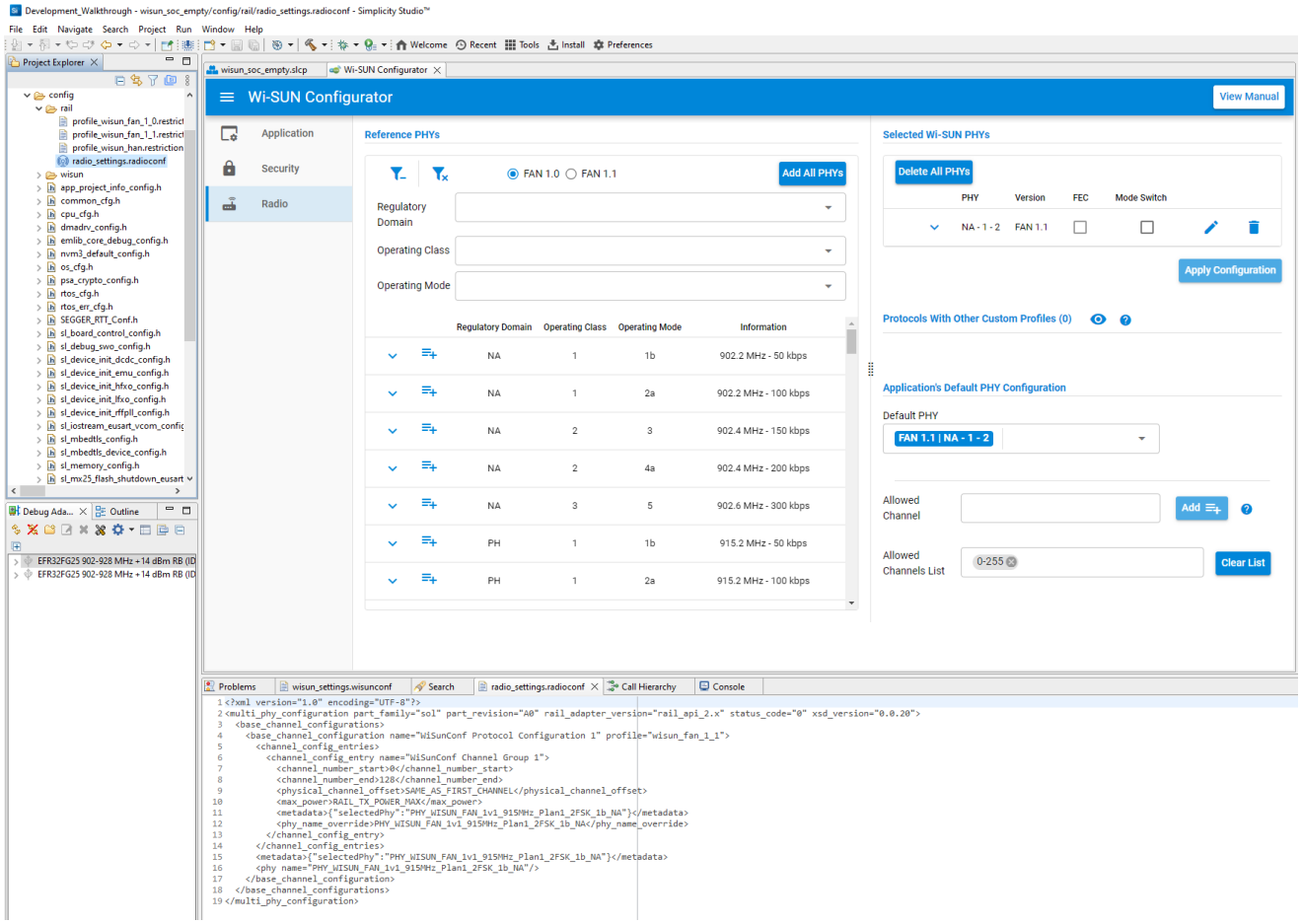
You can cross-check these three values.

> TIP: As already stated, in a production application, these will need to be different per device. The CA certificate

> may be shared, but the Device key and certificate should be device-specific.

Open the `config/rail/radio_settings.radioconf` file with **Open With > Text Editor**, and move it in the same frame as `config/wisun/wisun_settings.wisunconf`.

- The **Radio** tab gives access to:
  - In the **Selected Wi-SUN PHYs** frame, the list of currently selected PHYs. Here, only the default `NA - 1 -2` PHY is selected.
  - In the **Application's Default PHY Configuration** frame, the default PHY.
    - Obviously, this needs to be present in the selected PHYs, and in your case (using a single PHY) is also `NA - 1 -2`.
  - In **Reference PHYs**, tools to select the PHYs.

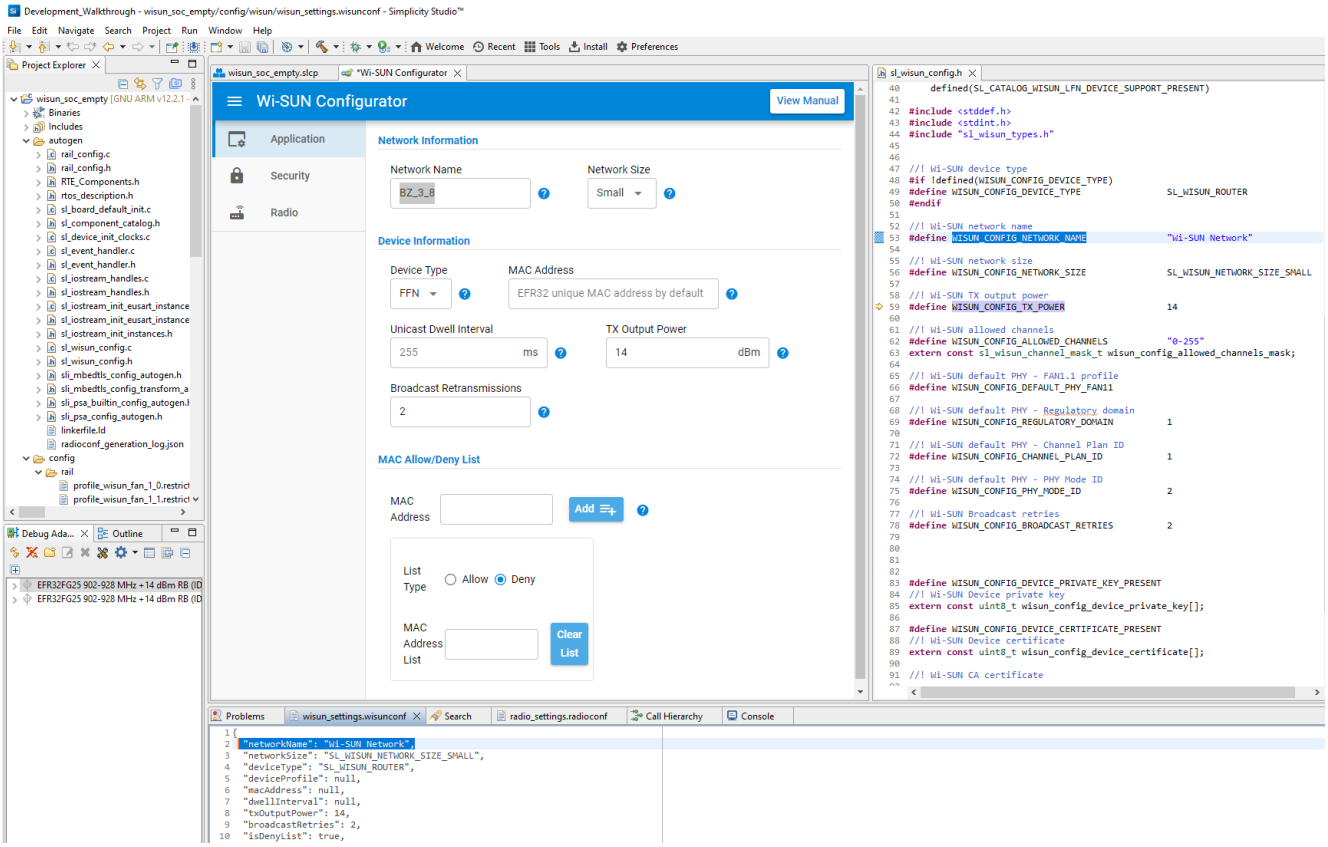


Observing `config/rail/radio_settings.radioconf`, you can check that `<phy_name_override>` matches the select PHY.

Once again, you see some relationship between the GUI, the control files, and the file used to compile.

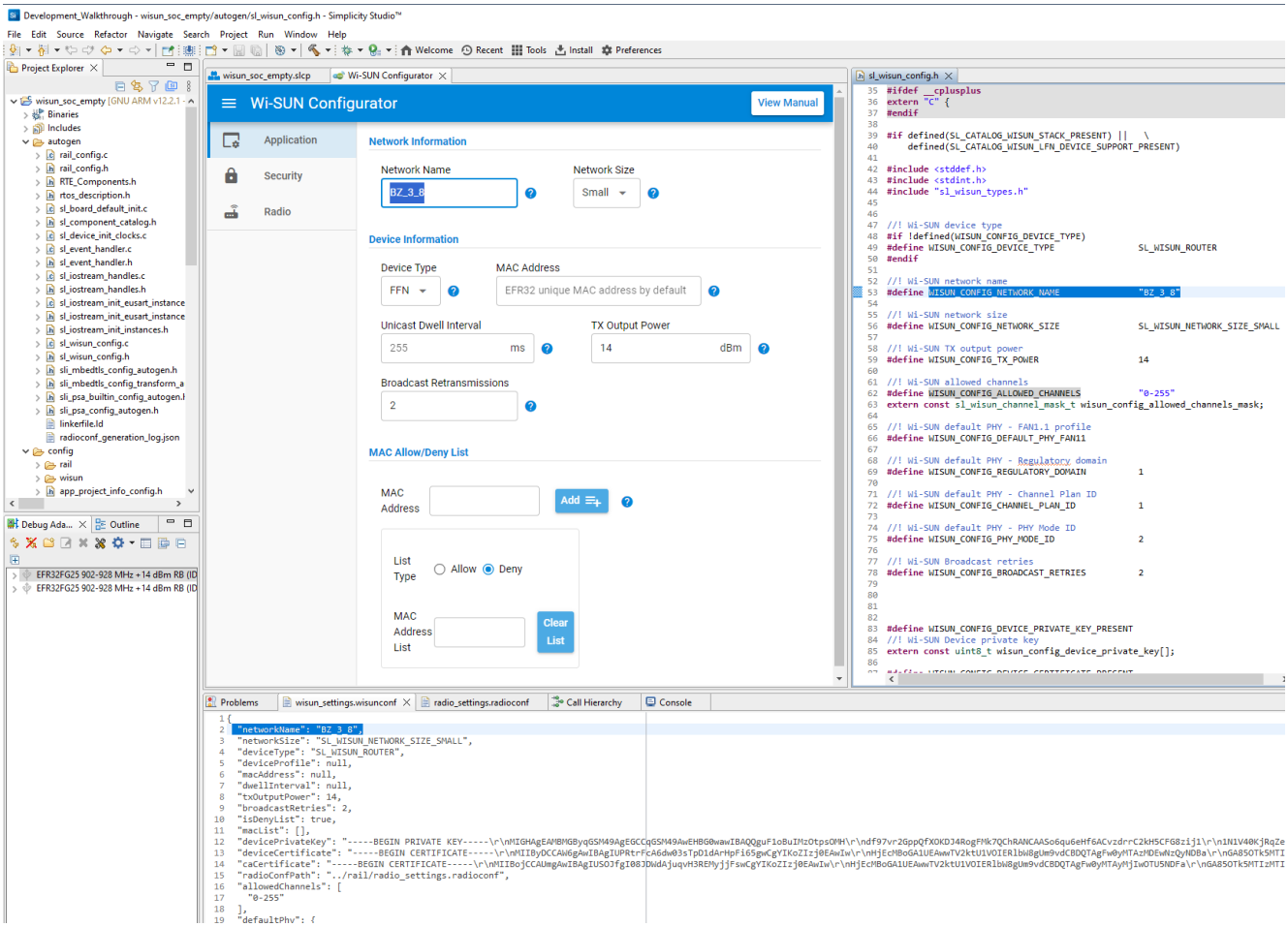
Changes

Now, change your Network name to `BZ_3_8` in the **Wi-SUN Configurator Application** tab.



- You can see an `*` next to the **Wi-SUN Configurator** tab's name. This indicates that the settings are not saved yet.
- No change to the other files (`autogen/sl_wisun_config.h` and `config/wisun_settings.wisunconf`).

Click the  in Simplicity Studio or 'Ctrl-s' to save your changes.



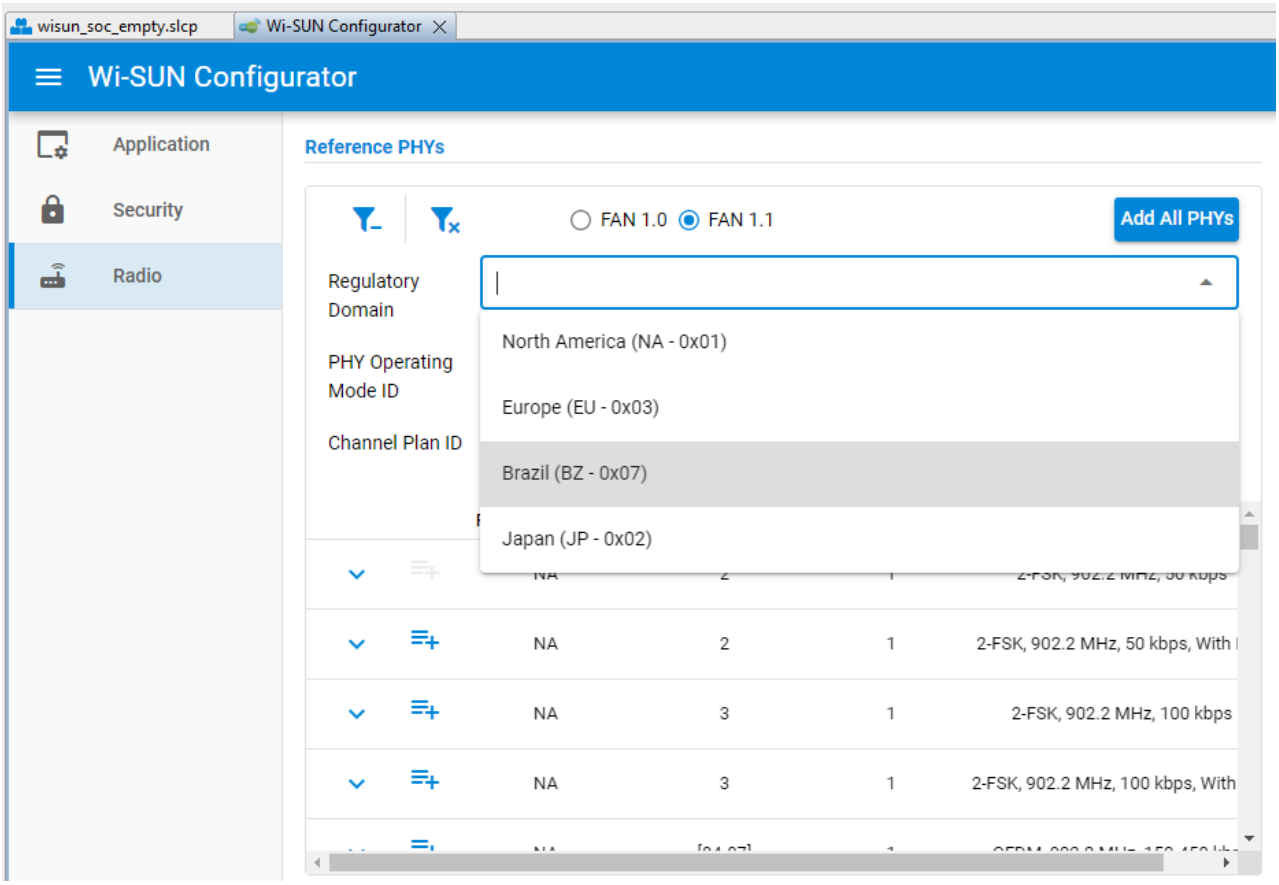
Changes are propagated to both `autogen/sl_wisun_config.h` and `config/wisun_settings.wisunconf`.

Now move to the **Radio** tab, since you won't change the security settings yet.

On the right side of the **Radio** tab, you see that you have by default only the `NA - 1 - 2` PHY in the list.

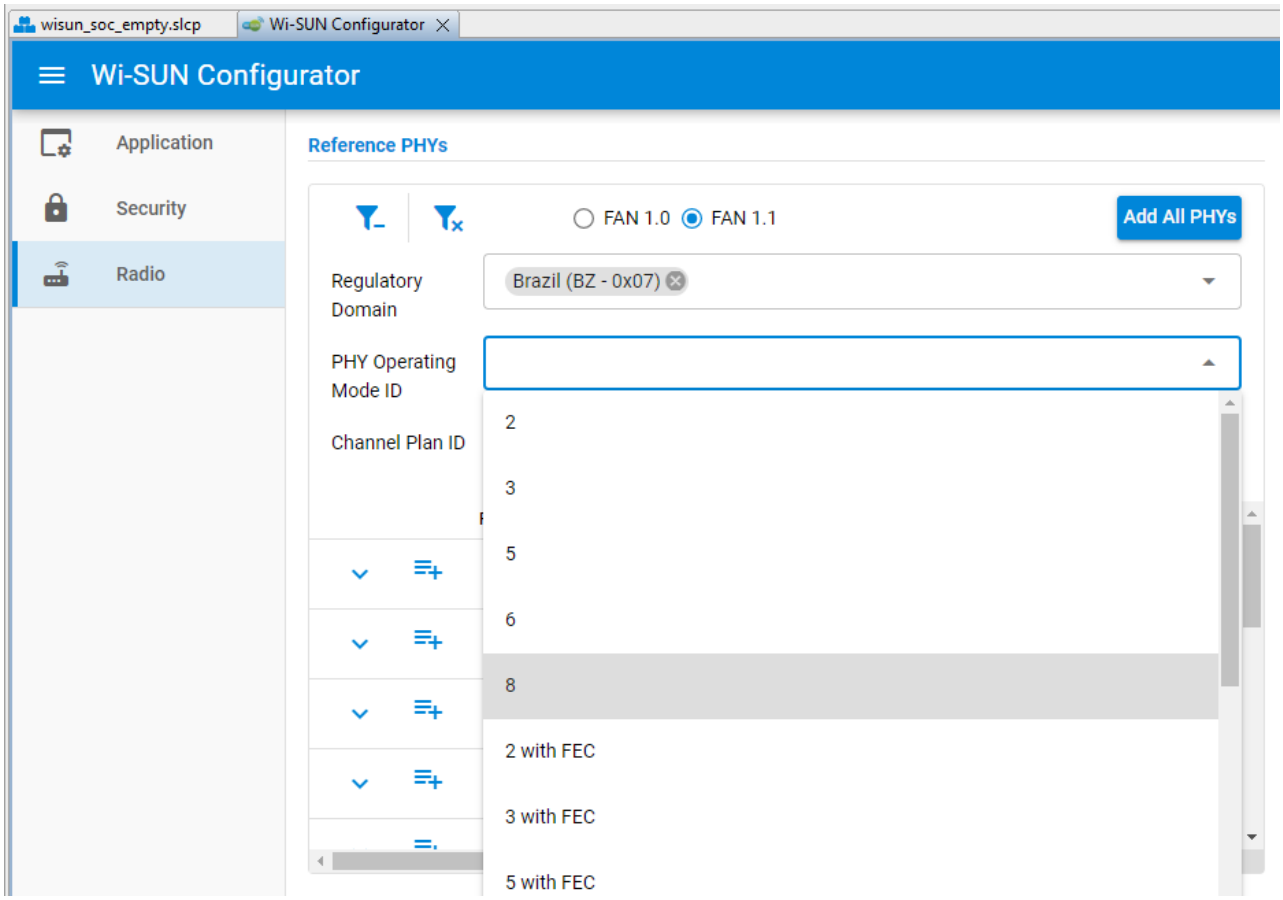
Now add the `BZ - 3 -8` PHY.

1. Slide left to see the **Reference PHYs** area.
2. Select the **FAN 1.1** radio button. This changes the content of the frame to match FAN 1.1 parameters.
3. Select the **Brazil (BZ - 0x07)** option in the **Regulatory Domain** dropdown.



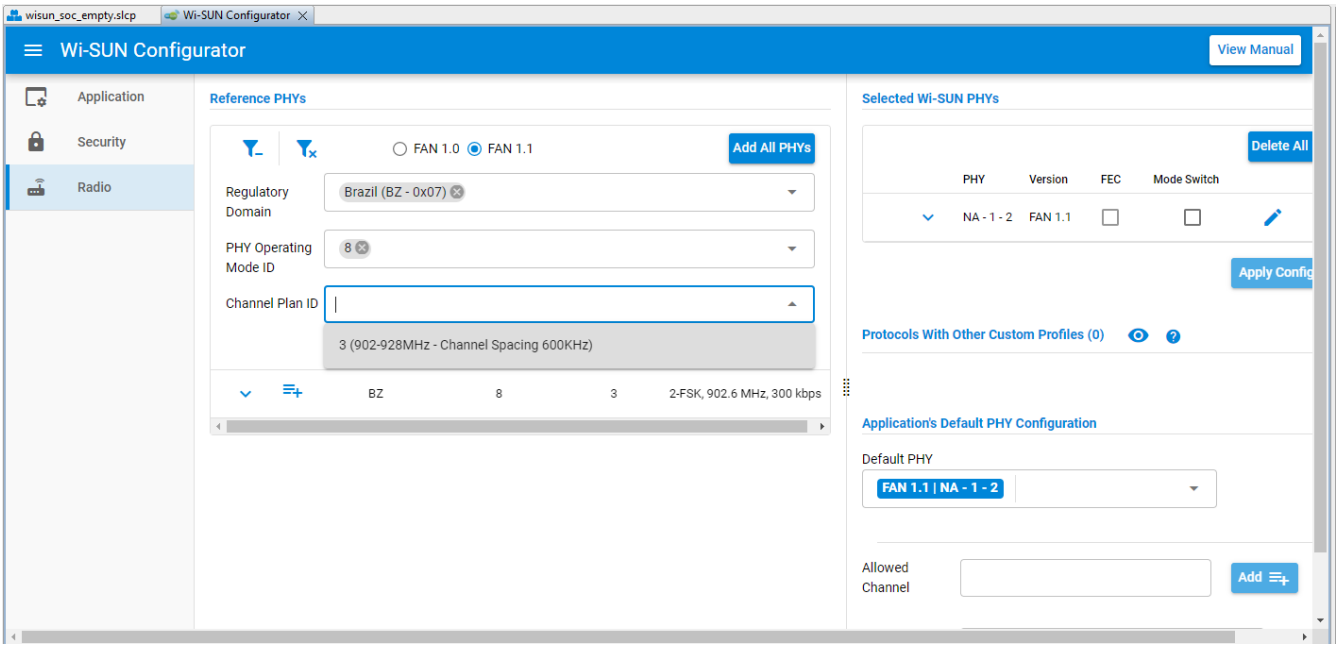
You may have noticed that the list of possible PHYs at the bottom is now reduced to 'BZ' PHYs.

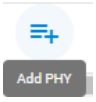
4. Select **phy_mode_id_8** in the **PHY Operating Mode ID** dropdown.



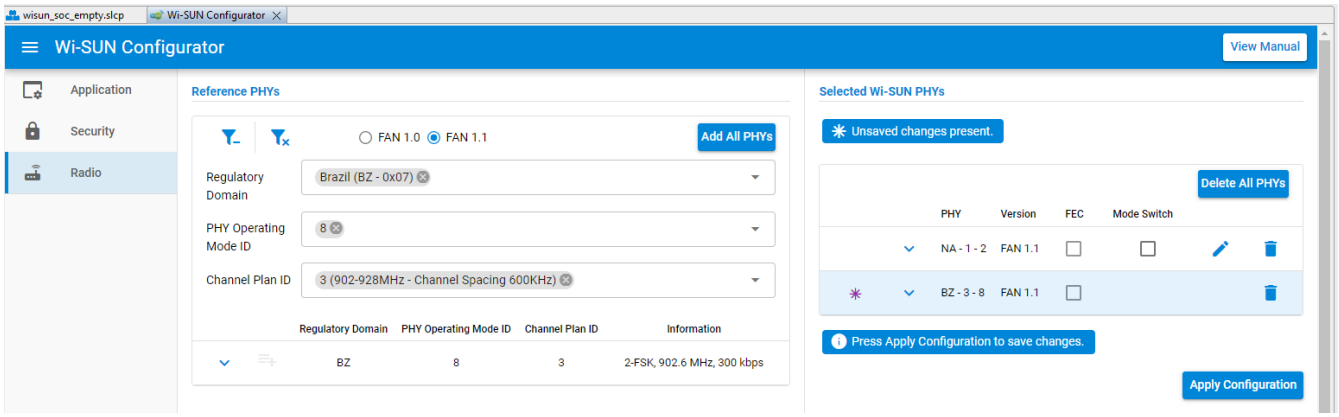
The list of possible PHYs at the bottom is further reduced to only those with '8' as their PHY Operating Mode ID (a single one).

5. You can continue by selecting **3 (902-928MHz - Channel Spacing 600 kHz)** in the **Channel Plan ID** list.
   This is not really required, since the above operations were used to reduce the list of possible PHYs, making it easy to select the desired PHY.

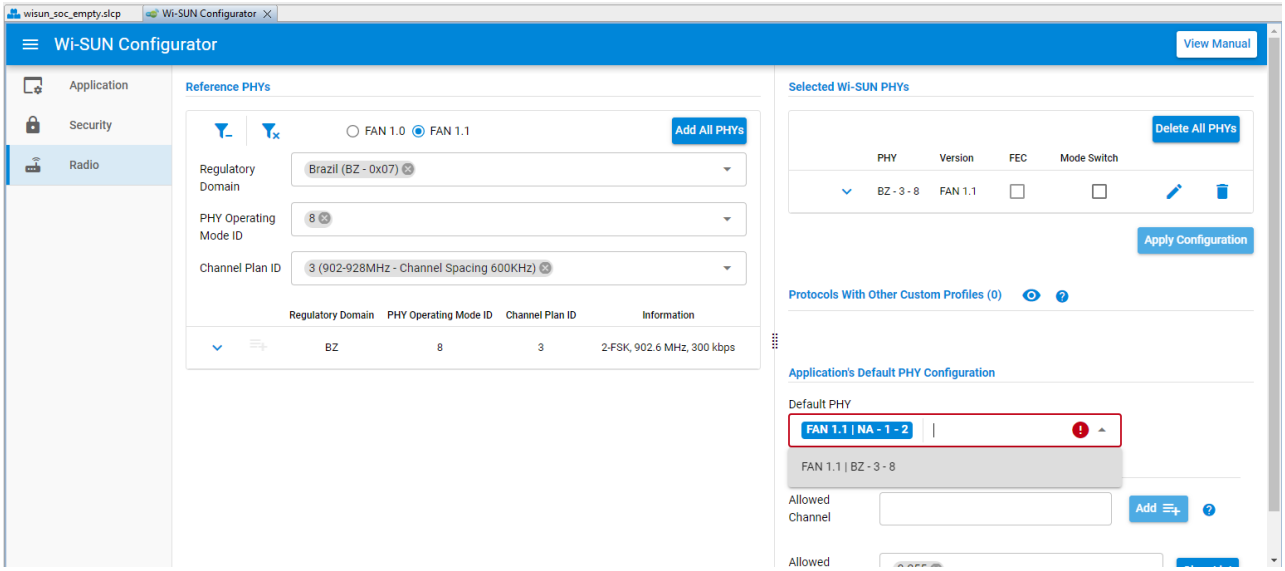


Click the blue 

icon on the PHY line to add it to the **Selected Wi-SUn PHYs** list.

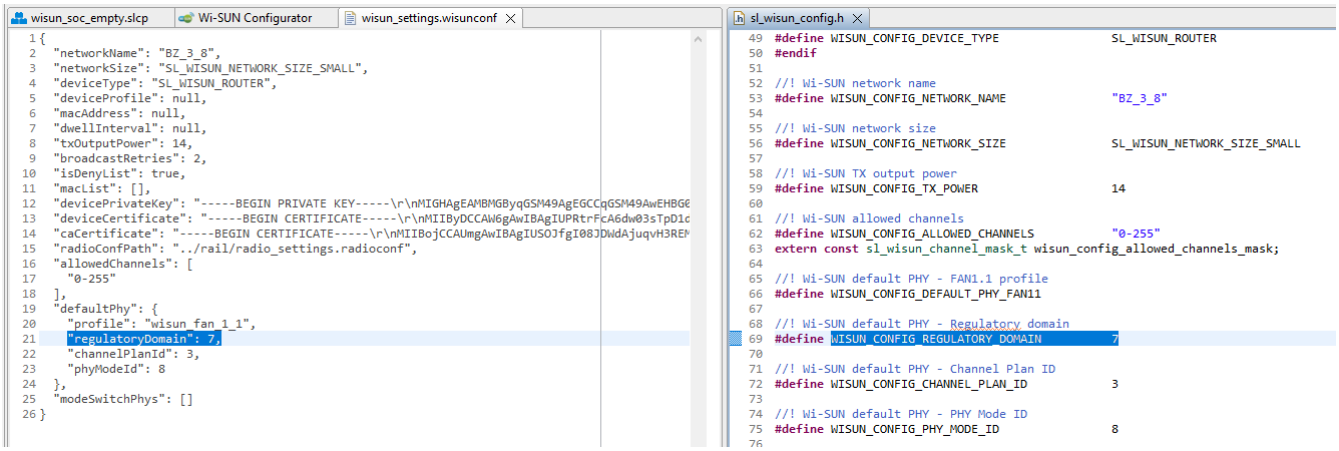


- The icon you just clicked turns light gray, indicating that this PHY is already in the list.
- A message is present in **Selected Wi-SUN PHYs** to indicate unsaved changes.
- A message informs you that you should save your new configuration by clicking the **Apply Configuration** button.
- Once saved, you can check in the `config/rail/radio_settings.radioconf` file that you now have two PHYs.
- In most cases, your Wi-SUN device will be targeting a single PHY, so also delete the `NA - 1 -2` PHY from the list, using the  icon. Confirm the deletion and save the configuration again.



- Only one PHY is now in `config/rail/radio_settings.radioconf`, your 'BZ' PHY.

7. A red box appears around the **Default PHY**, because the selected default is not part of the list anymore, so select **FAN 1.1 | BZ - 3 - 8** and save as instructed.



8. You can check in `config/wisun/wisun_settings.wisunconf` that your settings are now in the GUI control file, and also in `autogen/sl_wisun_config.h` that the following items match your selection:
   - WISUN_CONFIG_DEFAULT_PHY_FAN11
   - WISUN_CONFIG_REGULATORY_DOMAIN
   - WISUN_CONFIG_CHANNEL_PLAN_ID
   - WISUN_CONFIG_PHY_MODE_ID

Now, you can rebuild your project, and it will attempt to connect to your new 'BZ_3_8" Wi-SUN network.

While it's compiling, you can use 'git status' in your project folder to see which files have been modified.

```
$ git status
On branch master
Changes not staged for commit:
  (use "git add <file>..." to update what will be committed)
  (use "git restore <file>..." to discard changes in working directory)
        modified:   autogen/radioconf_generation_log.json
        modified:   autogen/rail_config.c
        modified:   autogen/rail_config.h
        modified:   autogen/sl_wisun_config.h
        modified:   config/rail/radio_settings.radioconf
        modified:   config/wisun/wisun_settings.wisunconf
```

You can use 'git diff' on any of these files to see the actual modifications.

```
$ git diff autogen/sl_wisun_config.h
diff --git a/autogen/sl_wisun_config.h b/autogen/sl_wisun_config.h
index 6c8cb63..3983737 100644
--- a/autogen/sl_wisun_config.h
+++ b/autogen/sl_wisun_config.h
@@ -41,7 +41,7 @@
 #endif

 //! Wi-SUN network name
-#define WISUN_CONFIG_NETWORK_NAME             "Wi-SUN Network"
+#define WISUN_CONFIG_NETWORK_NAME             "BZ_3_8"

 //! Wi-SUN network size
 #define WISUN_CONFIG_NETWORK_SIZE             SL_WISUN_NETWORK_SIZE_SMALL
@@ -54,13 +54,13 @@ extern const sl_wisun_channel_mask_t wisun_config_allowed_channels_mask;
 #define WISUN_CONFIG_DEFAULT_PHY_FAN11        1

 //! Wi-SUN default PHY - Regulatory domain
-#define WISUN_CONFIG_REGULATORY_DOMAIN        1
+#define WISUN_CONFIG_REGULATORY_DOMAIN        7

 //! Wi-SUN default PHY - Channel Plan ID
-#define WISUN_CONFIG_CHANNEL_PLAN_ID          1
+#define WISUN_CONFIG_CHANNEL_PLAN_ID          3

 //! Wi-SUN default PHY - PHY Mode ID
-#define WISUN_CONFIG_PHY_MODE_ID              2
+#define WISUN_CONFIG_PHY_MODE_ID              8

 //! Wi-SUN Broadcast retries
 #define WISUN_CONFIG_BROADCAST_RETRIES        2
```

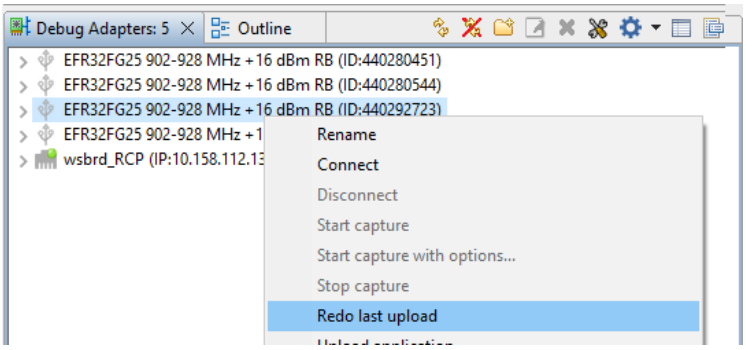
These are the expected changes.

Take a minute to flash your new application to the Router to see if it displays the new network name and can connect to your new Wi-SUN network.

> TIP: Before reloading, it's often useful to click the  icon in the Serial 1 Console to clear old messages.

The most convenient feature in Simplicity Studio to flash the same application after recompiling it is **Redo last upload** when selecting a Debug Adapter.

> TIP: You can even select multiple targets and trigger the last upload for all in a click. It will work even if the binaries are different, reloading the previous binary for each target.



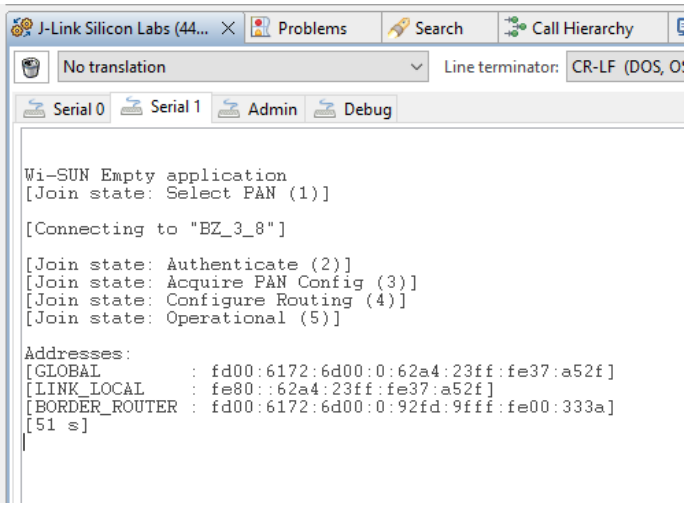
> TIP: Understand 'redo last upload' as 'reload the binary from the same location, with the same name'. If the binary content changed (your case), it will be updated.

While it's connecting, you can go back to Git and commit your changes.

```
git add --all
git commit -m "After changing to BZ 3 8"
```

After this is done, you can track your next changes.

In the meantime, it connected!



If the device is connected to the Wi-SUN network, the Wi-SUN specific part of the application is complete.

Resources Once Wi-SUN SoC Empty is Connected

- PING capability
- Listening to IPv6 Broadcast addresses:
  - The following IPv6 addresses are ready to receive broadcast messages:

Scope	IPv6	nanostack #define	Send to
Link Local (= neighbors)	FF02::1	ADDR_LINK_LOCAL_ALL_NODES	BR + FFN (1 hop) + LFN (1 hop)

Scope	IPv6	nanostack #define	Send to
Link Local (= neighbors)	FF02::2	ADDR_LINK_LOCAL_ALL_ROUTERS	BR + FFN (1 hop)
Realm Local (= all network)	FF03::1	ADDR_REALM_LOCAL_ALL_NODES	BR + FFN (n hops) + LFN (n hops)
Realm Local (= all network)	FF03::2	ADDR_REALM_LOCAL_ALL_ROUTERS	BR + FFN (n hops)

From this point, your device will:

- Reconnect to your Wi-SUN network automatically in case it's powered down then up.
- Regularly check for connection options within the surrounding Wi-SUN routers, and select a new parent if there is a better one or the current parent goes down. That is the self-healing part of Wi-SUN.
- Once connected, advertise the Wi-SUN network for other routers to connect, and become a parent for other devices if need be.
- If the Border Router goes down but another Border Router with the same Network name and credentials is within range, connect to this Border Router.
  - Adding several Border Routers to a Wi-SUN network (with Ethernet IPv6 connectivity between such Border Routers) is a good way to share the load within the network.

You can flash the same binary to several Evaluation kits. They will all connect in a tree-like fashion to the Border Router, routers being direct children of the Border Router acting as parents for devices located further away, and so on.

As explained earlier, in a production environment the credentials will need to be unique per device, but for development purposes, it is sufficient to use the same credentials for a number of devices.

> TIP: On the Linux Border Router, you can use `watch wsbrd_cli status` to monitor the network topology. You can also use the wsbrd GUI to get a graphical representation of this Wi-SUN network topology.

It's now time to look at the Wi-SUN connection process in detail.

API Calls to Connect

Wi-SUN API Calls to Connect to a Wi-SUN Network

In the first part of this series, you connected to a Wi-SUN Network without looking at how this is achieved.

Connection was easy because you relied on the Wi-SUN Configurator GUI and the underlying **Wi-SUN Application Core component**.

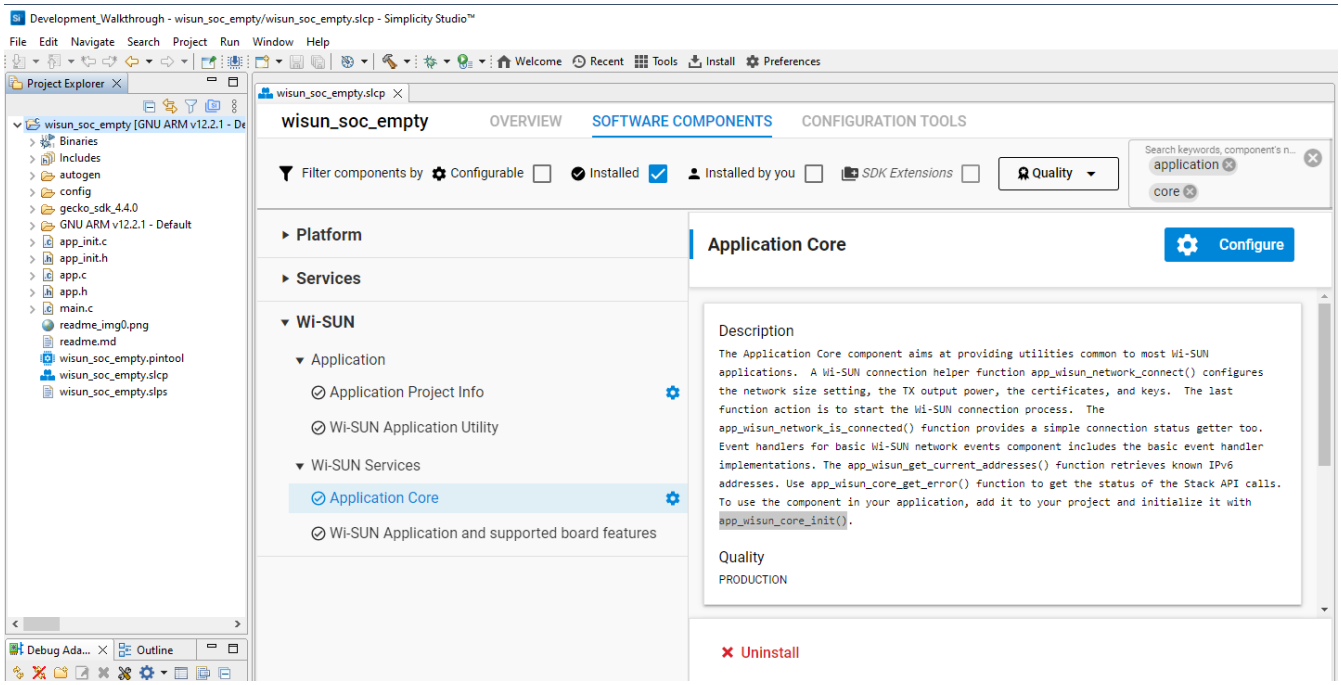
This is a lecture on how the **Wi-SUN Application Core component** uses the Wi-SUN Stack API functions to achieve successful connection. Key points to understand are:

- Some general principles used when developing a Wi-SUN application
- Details of the connection process

Wi-SUN Application Core Component

There are at least four ways to check that the Wi-SUN Application Core component is part of the project.

1. Check the installed components in the **SOFTWARE COMPONENT** tab. You get access to an abstract of the component's readme, indicating the use of `app_wisun_core_init()` to initialize it.

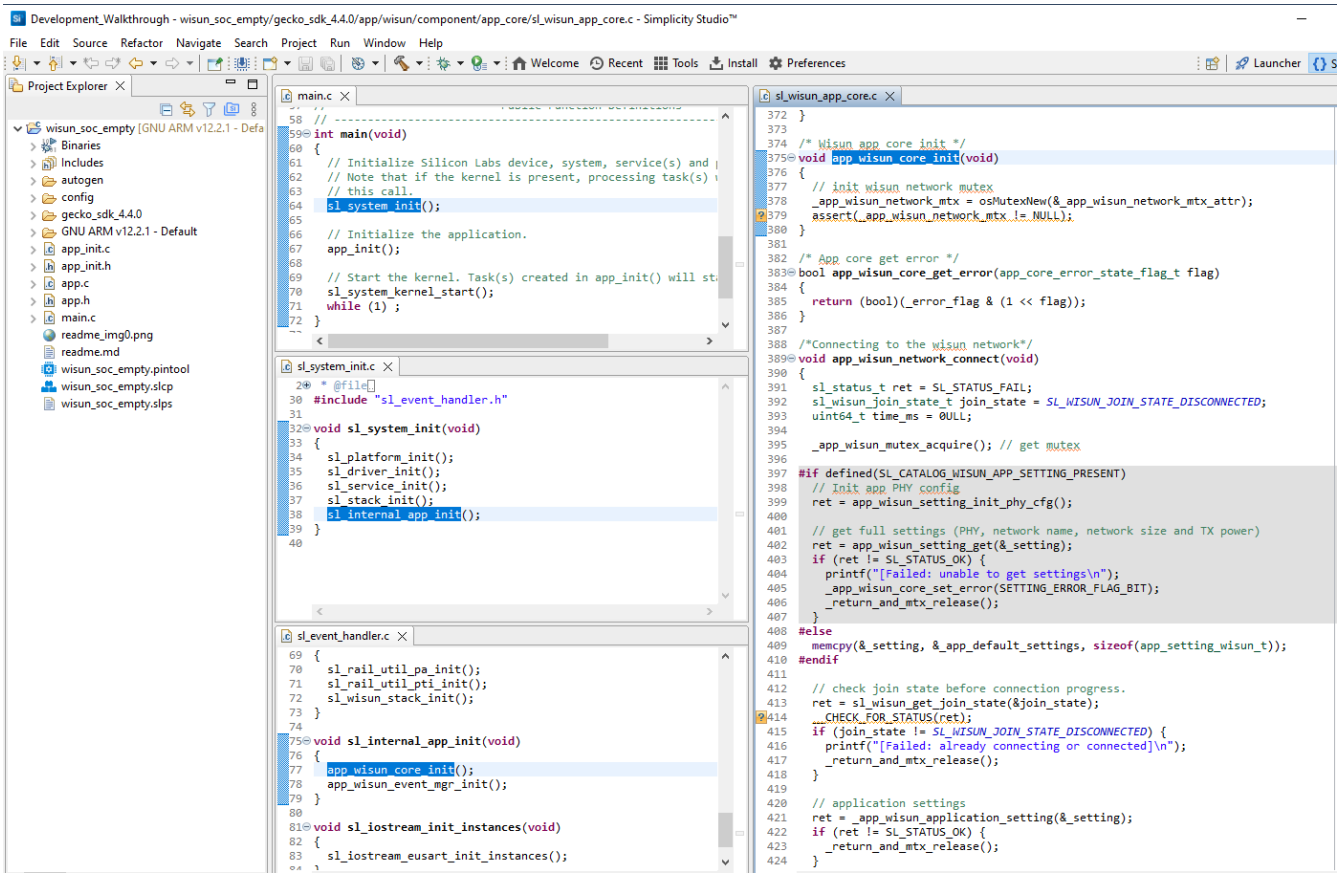


2. Check the content of `wisun_soc_empty.slcp` with a text editor.

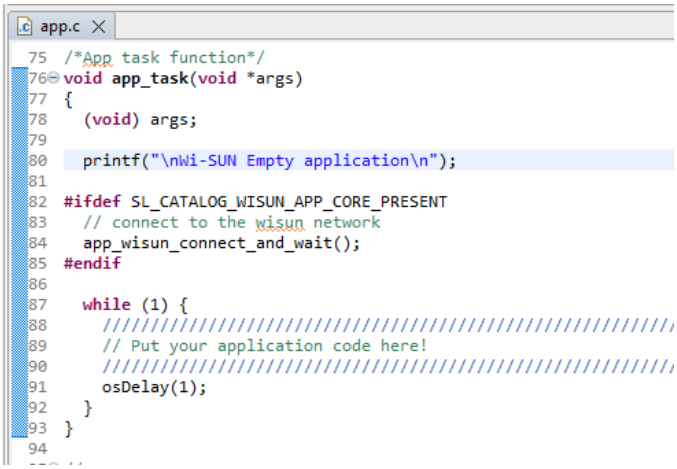
wisun_soc_empty.slcp

```
45 - {id: restrictions_profile_wisun_fan_1_0}
46 - {id: restrictions_profile_wisun_fan_1_1}
47 - {id: wisun_stack_debug}
48 - {id: silabs_core_sl_malloc}
49 - {id: EFR32FG25B222F1920IM56}
50 - {id: printf}
51 - {id: wisun_stack}
52 - {id: wisun_config}
53 - {id: sleeptimer}
54 - {id: sl_wisun_app_core}
55 other_file:
56 - {path: readme_img0.png}
57 define:
58 - {name: DEBUG_EFM}
```

3. In the source code, you can see that the `main()` function calls `sl_system_init()`, which calls `sl_internal_app_init()`, which calls `app_wisun_core_init()`. This code is automatically added when adding the component via the GUI.



4. In `app.c/app_task()`, you see that the code surrounded by `SL_CATALOG_WISUN_APP_CORE_PRESENT` is not grayed out, indicating that `SL_CATALOG_WISUN_APP_CORE_PRESENT` is defined and the corresponding code will be compiled.

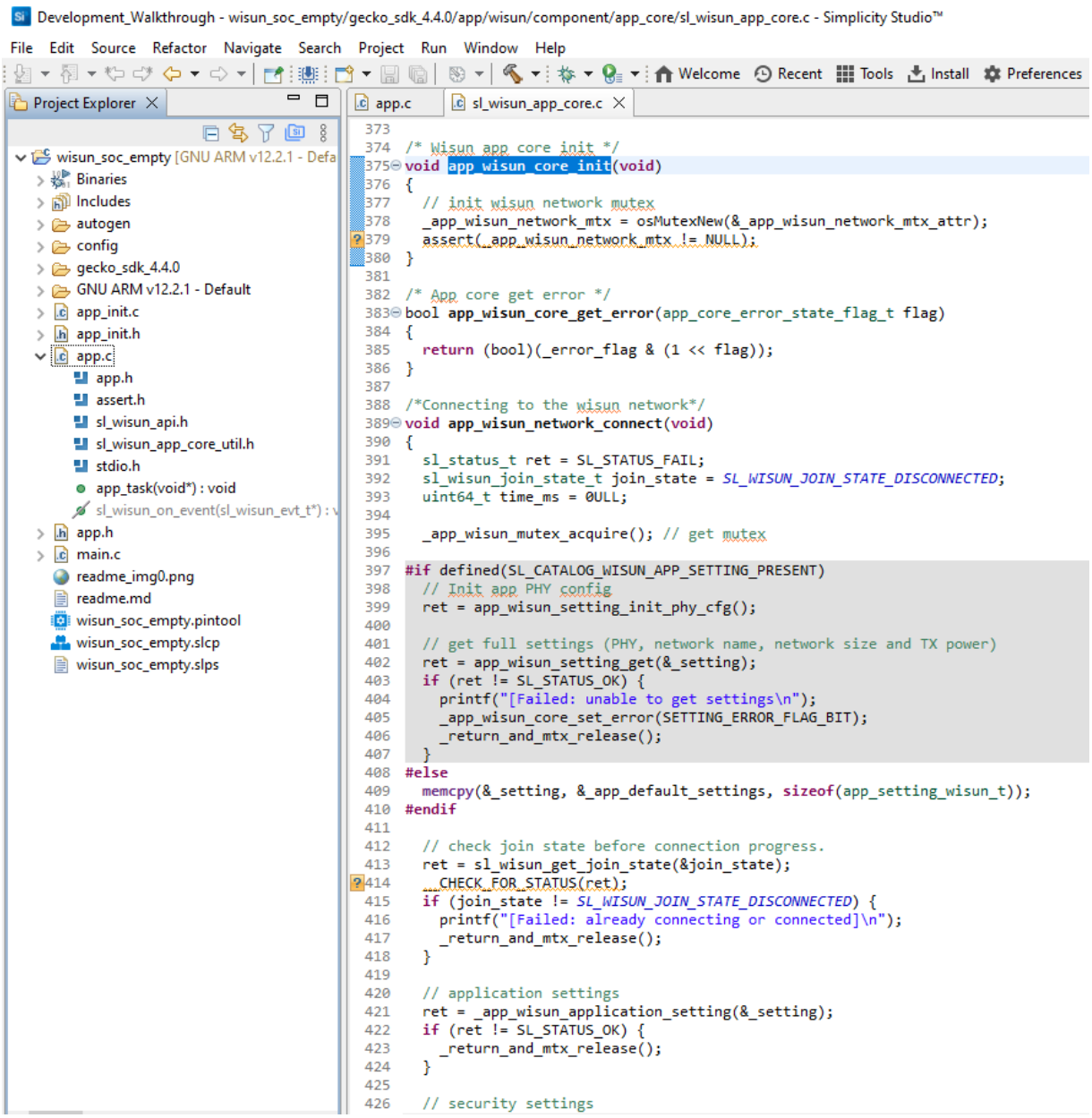


The Wi-SUN Application Core component uses the Wi-SUN Configurator settings to achieve connection to a Wi-SUN Network with only GUI actions. This is the simplest way to connect to any Wi-SUN Network.

Now consider which function calls are required to connect to a Wi-SUN Network.

Wi-SUN Network Connection Walkthrough

You saw earlier the connection process in `/wisun_soc_empty/gecko_sdk_x.y.z/app/wisun/component/app_core/sl_wisun_app_core.c`, with several steps during the connection preparation.



Preparation of the PHY and Network Settings

The grayed out part is not compiled, so the code uses line 409.

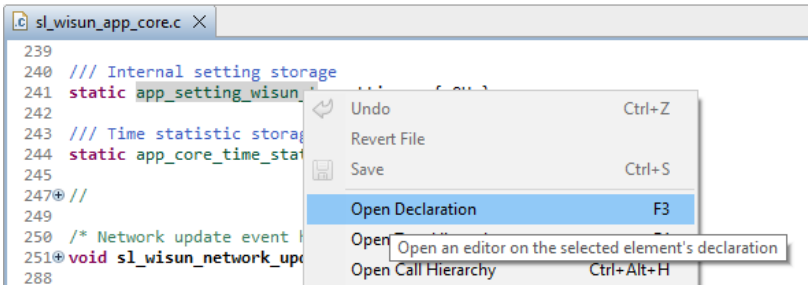
```
memcpy(&_setting, &_app_default_settings, sizeof(app_setting_wisun_t));
```

This line copies the `_app_default_settings` into `_setting`, a `app_setting_wisun_t` structure defined earlier as:

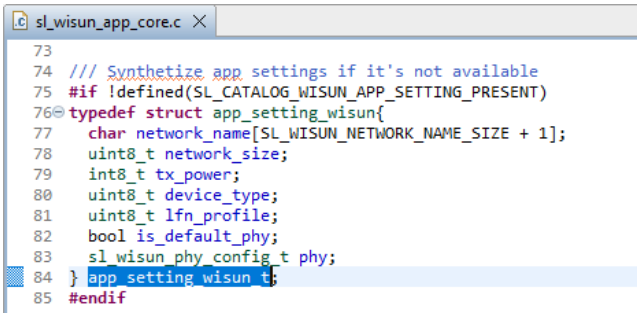
```
/// Internal setting storage
static app_setting_wisun_t _setting = { 0 };
```

`app_setting_wisun_t` structure

You can check the declaration of the `app_setting_wisun_t` structure by selecting the **Open Declaration** menu item.



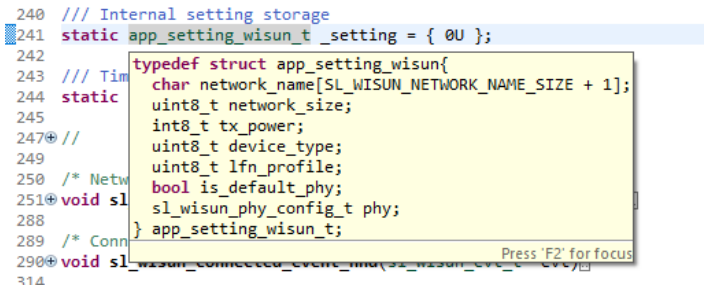
This brings you higher up in `/wisun_soc_empty/gecko_sdk_x.y.z/app/wisun/component/app_core/sl_wisun_app_core.c` to the declaration of the structure.



You can see here that the structure stores:

- The network name (as a string, which always end up with an additional `0x00` byte as the string terminator, hence the `+1` in the array definition)
- The network_size
- The Tx power
- The device type
- The LFN profile (only used for LFNs)
- A flag indicating if the device is using the default PHY
- Another `sl_wisun_phy_config_t` structure describing the PHY

> TIP: You can also check the definition of a structure by hovering over it until a pop-up appears with the definition.



`sl_wisun_phy_config_t` structure

sl_wisun_types.h

```
602 /// PHY configuration
603 SL_PACK_START(1)
604 typedef struct {
605   /// Configuration type (#sl_wisun_phy_config_type_t)
606   uint32_t type;
607   /// Configuration
608   union {
609     /// Configuration for #SL_WISUN_PHY_CONFIG_FAN10 type
610     sl_wisun_phy_config_fan10_t fan10;
611     /// Configuration for #SL_WISUN_PHY_CONFIG_FAN11 type
612     sl_wisun_phy_config_fan11_t fan11;
613     /// Configuration for #SL_WISUN_PHY_CONFIG_EXPLICIT type
614     sl_wisun_phy_config_explicit_t explicit_plan;
615     /// Configuration for #SL_WISUN_PHY_CONFIG_IDS type
616     sl_wisun_phy_config_ids_t ids;
617     /// Configuration for #SL_WISUN_PHY_CONFIG_CUSTOM_FSK type
618     sl_wisun_phy_config_custom_fsk_t custom_fsk;
619     /// Configuration for #SL_WISUN_PHY_CONFIG_CUSTOM_OFDM type
620     sl_wisun_phy_config_custom_ofdm_t custom_ofdm;
621     /// Configuration for #SL_WISUN_PHY_CONFIG_CUSTOM_OQPSK type
622     sl_wisun_phy_config_custom_oqpsk_t custom_oqpsk;
623   } config;
624 } sl_wisun_phy_config_t;
625 SL_PACK_END()
626
```

The `sl_wisun_phy_config_t` structure is a more complex `union` type, which requires clarification for people not familiar with C coding.

A `union` is a place where several underlying structures can be stored, but to save memory, only one will be stored at a time. In your example, you know which underlying structures can be used by looking at each `struct {} <STRUCT_NAME>;` block. You can locate:

- `fan10` on lines 503-512
- `fan11` on lines 515-522
- `explicit` on lines 525-534

sl_wisun_types.h

```
501
502 /// FAN1.0 PHY configuration
503 typedef struct {
504   /// Regulatory domain (#sl_wisun_regulatory_domain_t)
505   uint8_t reg_domain;
506   /// Operating class (#sl_wisun_operating_class_t)
507   uint8_t op_class;
508   /// Operating mode (#sl_wisun_operating_mode_t)
509   uint8_t op_mode;
510   /// 1 if FEC is enabled, 0 if not
511   uint8_t fec;
512 } sl_wisun_phy_config_fan10_t;
513
514 /// FAN1.1 PHY configuration
515 typedef struct {
516   /// Regulatory domain (#sl_wisun_regulatory_domain_t)
517   uint8_t reg_domain;
518   /// Channel plan ID
519   uint8_t chan_plan_id;
520   /// PHY mode ID
521   uint8_t phy_mode_id;
522 } sl_wisun_phy_config_fan11_t;
523
524 /// Explicit PHY configuration
525 typedef struct {
526   /// Ch0 center frequency in kHz
527   uint32_t ch0_frequency_khz;
528   /// Number of channels
529   uint16_t number_of_channels;
530   /// Channel spacing (#sl_wisun_channel_spacing_t)
531   uint8_t channel_spacing;
532   /// PHY mode ID
533   uint8_t phy_mode_id;
534 } sl_wisun_phy_config_explicit_t;
```

Usually, unions start with a `type` information, here on line 499, which is used to select the underlying structure.

In your example application, you're using `fan11`, so the `type` will be set accordingly and the FAN 1.1 values will be accessed as:

- `phy.fan11.reg_domain`
- `phy.fan11.chan_plan_id`
- `phy.fan11.phy_mode_id`

if `phy` is a `sl_wisun_phy_config_t`

or

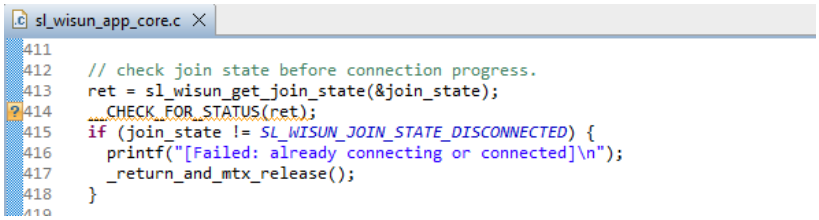
- `phy->fan11.reg_domain`
- `phy->fan11.chan_plan_id`
- `phy->fan11.phy_mode_id`

if `phy` is a pointer to a `sl_wisun_phy_config_t`

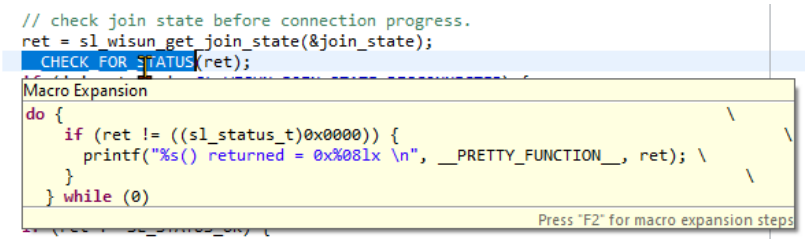
> TIP: When moving around files in Simplicity Studio, use **Alt+left_arrow** or **Alt+Right_arrow** to move between recent locations.

Check of the Current Join State

Back to `app_wisun_network_connect()`, you see that there is a check of the current join state. This is because, before calling the join function, the device must be disconnected so that it properly starts using the new parameters and clears its state machine.



Here it's worth looking at what the `__CHECK_FOR_STATUS` is doing, since you may see it used often. Use the pop-up since it's relatively short.



You can understand looking at the macro definition that an error message will be printed with the error number when the `ret` value is not `0x0000`, and if `ret` is `0x0000`, nothing will happen. If an error message will be printed, it will also contain the name of the function from where it is originating, (here `app_wisun_network_connect`), as a replacement for the `__PRETTY_FUNCTION__` identifier. Defining macros such as these is a good way to avoid repeating the same code block in your code. It makes the code more compact and less error prone (once the macros are properly tested).

> TIP: Generally, in C code, specific identifiers such as those declared by the compilers are prefixed and suffixed with `__`. Avoid using this in your own variable names to avoid confusion. Here, using the C++

> `__PRETTY_FUNCTION__` is not necessary, since the code is compiled as C code, not C++. Using `__FUNCTION__` is sufficient.

`_app_wisun_mutex_acquire()` and `_return_and_mtx_release()`

You see `_app_wisun_mutex_acquire()` and `_return_and_mtx_release()` in the code around some code blocks.

The `_app_wisun_mutex_acquire()` name is a bit more explicit. It indicates that you are acquiring a 'mutex', i.e, a mutual-exclusion flag. This is commonly used in C code when several parts of the code (from different threads) may need to access common variables.

Whenever the code needs to access such data, you want to make sure no other function is changing it at the same time, which could lead to unexpected results.

When calling `_app_wisun_mutex_acquire()`, you're waiting for the number of current users of the flag to be `0`. Until the flag is `0`, you wait. Once it is `0`, the underlying system sets it to `1` and gives you the access. From this point on, you 'own' the mutex flag and the right to access the shared data.

This means that, if you never release it, all other users (including yourself) will never get access to the shared data!

When you're done, release the mutex by calling `_return_and_mtx_release()`, which, if you look into it, starts by releasing the mutex and then returns.

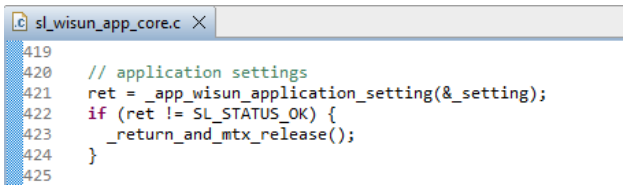
A typical pattern when using a mutex is:

- Acquiring the mutex at the beginning of the function (as you see here on line 354)
- Releasing the mutex before every return

> TIP: You can see several occurrences of `_return_and_mtx_release()` in a single function, because you need to release the mutex before returning. To achieve this, all `return` calls are replaced by `_return_and_mtx_release()` on every place in the function where you return from.

Filling the Network Settings

Next you get to the application settings, on line 379.



You see here that the code calls `_app_wisun_application_setting` with the content of `_setting` from the preparation of the PHY and Network settings.

Now look into this function (only showing code which is not grayed out, i.e. compiled code).

sl_wisun_app_core.c

```
596
597 static sl_status_t app_wisun_application_setting(const app_setting_wisun_t *
598 {
599   sl_status_t ret = SL_STATUS_FAIL;
600   const sl_wisun_connection_params_t *conn_param = NULL;
601 #if defined(WISUN_CONFIG_BROADCAST_RETRIES)
602   sl_wisun_connection_params_t update_param = { 0 };
603 #endif
604
605   conn_param = sl_wisun_get_conn_param_by_nw_size((sl_wisun_network_size_t) s
606
607   ret = sl_wisun_set_device_type((sl_wisun_device_type_t)setting->device_type
608   if (ret != SL_STATUS_OK) {
609     printf("[Failed: unable to set device type: %lu]\n", ret);
610     return ret;
611   }
612 #if defined(SL_CATALOG_WISUN_LFN_DEVICE_SUPPORT_PRESENT)
613   if (setting->device_type == SL_WISUN_LFN) {
614     // Store LFN profile based on wisun config
615     ret = sl_wisun_set_lfn_parameters(app_wisun_get_lfn_params());
616     if (ret != SL_STATUS_OK) {
617       printf("[Failed: unable to set device type: %lu]\n", ret);
618       return ret;
619     }
620   }
621 #endif
622
623 #if defined(WISUN_CONFIG_BROADCAST_RETRIES)
624   memcpy(&update_param, conn_param, sizeof(sl_wisun_connection_params_t));
625   update_param.mpl.trickle_expirations = WISUN_CONFIG_BROADCAST_RETRIES;
626   conn_param = &update_param;
627 #endif
628
629   // sets the network name
630   ret = sl_wisun_set_connection_parameters(conn_param);
631   if (ret != SL_STATUS_OK) {
632     printf("[Failed: unable to set network size: %lu]\n", ret);
633     _app_wisun_core_set_error(SET_NETWORK_SIZE_ERROR_FLAG_BIT);
634     return ret;
635   }
636
637   // sets the TX power
638   ret = sl_wisun_set_tx_power(setting->tx_power);
639   if (ret != SL_STATUS_OK) {
640     printf("[Failed: unable to set TX power: %lu]\n", ret);
641     _app_wisun_core_set_error(SET_TX_POWER_ERROR_FLAG_BIT);
642     return ret;
643   }
644 #if defined(WISUN_CONFIG_ALLOWED_CHANNELS)
645   ret = sl_wisun_set_channel_mask(&wisun_config_allowed_channels_mask);
646   if (ret != SL_STATUS_OK) {
647     printf("[Failed: unable to set allowed channels: %lu]\n", ret);
648     return ret;
649   }
650 #endif
651 #if defined(WISUN_CONFIG_MODE_SWITCH_PHYS_NUMBER)
652   ret = sl_wisun_set_pom_ie(WISUN_CONFIG_MODE_SWITCH_PHYS_NUMBER,
653                             (uint8_t *)wisun_config_ms_phys,
654                             APP_WISUN_MDR_COMMAND_CAPABILITY);
655   if (ret != SL_STATUS_OK) {
656     printf("[Failed: unable to set mode switch phys: %lu]\n", ret);
657     return ret;
658   }
659 #endif
660
661 #if defined(WISUN_CONFIG_DWELL_INTERVAL)
662   // sets unicast
663   ret = sl_wisun_set_unicast_settings(WISUN_CONFIG_DWELL_INTERVAL);
664
665   if (ret != SL_STATUS_OK) {
666     printf("[Failed: unable to set dwell interval: %lu]\n", ret);
667     _app_wisun_core_set_error(SET_DWELL_INTERVAL_ERROR_FLAG_BIT);
668     return ret;
669   }
670 #endif
671
672 #if defined(WISUN_CONFIG_MAC_ADDRESS_PRESENT)
673   // set mac address
```

On line 605, the `sl_wisun_get_conn_param_by_nw_size()` function is called to set the `conn_param` structure with values matching the network size selected by the user.

`conn_param` is a `sl_wisun_connection_params_t` structure (declared on line 600), defined as:

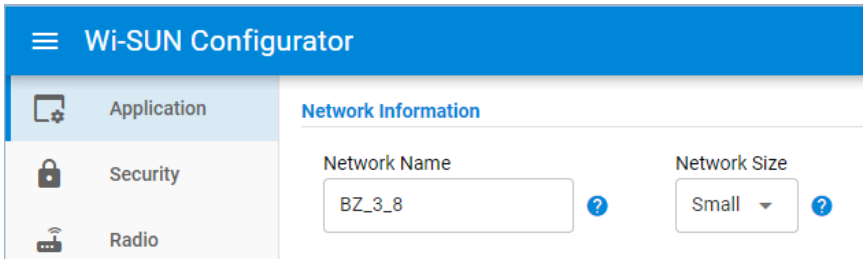
sl_wisun_connection_params_api.h

```
183 typedef struct {
184   /**
185    * Version of this API.
186    *
187    * This field allows to store the parameters in an NVM and check on reload
188    * that they are compatible with the stack if there was an update.
189    */
190   uint32_t version;
191   /// PAN discovery parameter set
192   sl_wisun_params_discovery discovery;
193   /// PAN configuration parameter set
194   sl_wisun_params_configuration configuration;
195   /// Authentication parameter set
196   sl_wisun_params_eapol eapol;
197   /// RPL parameter set
198   sl_wisun_params_rpl rpl;
199   /// MPL parameter set
200   sl_wisun_params_mpl mpl;
201   /// Misc parameter set
202   sl_wisun_params_misc misc;
203 } sl_wisun_connection_params_t;
```

As you can see, there are several underlying structures, used to store:

- discovery parameters
- configuration parameters
- eapol parameters
- rpl parameters
- misc parameters

Your current **Network Size** setting is **Small**.



`sl_wisun_get_conn_param_by_nw_size` simply selects the corresponding settings based on this selection:

sl_wisun_trace_util.c

```
590 #if !defined(SL_CATALOG_WISUN_NCP_PRESENT)
591 const sl_wisun_connection_params_t *sl_wisun_get_conn_param_by_nw_size(const sl_wisun_network_size_t nw_size)
592 {
593   switch (nw_size) {
594     // Small
595     case SL_WISUN_NETWORK_SIZE_SMALL:
596       return &SL_WISUN_PARAMS_PROFILE_SMALL;
597
598     // Medium
599     case SL_WISUN_NETWORK_SIZE_MEDIUM:
600       return &SL_WISUN_PARAMS_PROFILE_MEDIUM;
601
602     // Large
603     case SL_WISUN_NETWORK_SIZE_LARGE:
604       return &SL_WISUN_PARAMS_PROFILE_LARGE;
605
606     // Test
607     case SL_WISUN_NETWORK_SIZE_TEST:
608       return &SL_WISUN_PARAMS_PROFILE_TEST;
609
610     // Certifacete and automatic size are not supported
611     default:
612       return NULL;
613   }
614 }
615 #endif
616
```

In the declaration of `SL_WISUN_PARAMS_PROFILE_SMALL`, you see all parameters and their values.

sl_wisun_connection_params_api.h

```
371 /// Profile for a small network
372 static const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_SMALL = {
373   .version = SL_WISUN_PARAMS_API_VERSION,
374   .discovery = {
375     .trickle_pa = {
376       .imin_s = 15,
377       .imax_s = 60,
378       .k = 1
379     },
380     .trickle_pas = {
381       .imin_s = 15,
382       .imax_s = 60,
383       .k = 1
384     },
385     .eapol_target_min_sens = DBM_TO_RSL_RANGE(-60),
386     .allow_skip = true
387   },
388   .configuration = {
389     .trickle_pc = {
390       .imin_s = 15,
391       .imax_s = 60,
392       .k = 1
393     },
394     .trickle_pcs = {
395       .imin_s = 15,
396       .imax_s = 60,
397       .k = 1
398     }
399   },
400   .eapol = {
401     .sec_prot_trickle = {
402       .imin_s = 60,
403       .imax_s = 120,
404       .k = 0,
405     },
406     .pmk_lifetime_m = MONTH_TO_MIN(4),
407     .ptk_lifetime_m = MONTH_TO_MIN(2),
408     .sec_prot_retry_timeout_s = 450,
409     .initial_key_min_s = 3,
410     .initial_key_max_s = 30,
411     .initial_key_retry_min_s = 180,
412     .initial_key_retry_max_s = 420,
413     .initial_key_retry_max_limit_s = 420,
414     .temp_min_timeout_s = 330,
415     .gtk_request_imin_m = 1,
416     .gtk_request_imax_m = 4,
417     .gtk_max_mismatch_m = 64,
418     .lgtk_max_mismatch_m = 60,
419     .sec_prot_trickle_expirations = 4,
420     .initial_key_retry_limit = 2,
421     .allow_skip = true
422   },
423   .rpl = {
424     .dis_max_delay_first_s = 2,
425     .dis_max_delay_s = 180,
426     .init_parent_selection_s = 10,
427     .etx_probe_period_max_s = 4,
428     .etx_samples_init = 1,
429     .etx_samples_refresh = 4,
430     .candidate_parents_max = 5,
431     .parents_max = 2,
432   },
433   .mpl = {
434     .trickle = {
435       .imin_s = 1,
436       .imax_s = 10,
437       .k = 8,
438     },
439     .seed_set_entry_lifetime_s = 180,
440     .trickle_expirations = 2,
441   },
442   .misc = {
443     .temp_link_min_timeout_s = 260,
444     .pan_timeout_m = 30,
445   }
446 };
```

> TIP: By default, the Wi-SUN Stack API defines three Network Sizes for general use: SMALL, MEDIUM, and LARGE. Each set of parameters corresponds to a good compromise for most applications. The application may

customize these or add more if required; it's up to the user. An application not using the Wi-SUN Application Core component will need to define its own `sl_wisun_connection_params_t` structure.

TIP: If you want to customize the connection parameters and start editing the file, Simplicity Studio will ask you whether you want to create a copy of the file inside your project or edit the GSDK file. (Remember that GSDK files are normally only logical links to the GSDK files.) If you opt for the copy, the logical link will be replaced by a copy of the GSDK file, and your changes will be local to your project. If you opt for editing the GSDK file, your changes will affect all your Wi-SUN projects, but may be 'lost' when you update the GSDK. (They are not really lost, since they will still be present in the previous GSDK folder, but you will need to patch the new GSDK files with similar changes.)

Setting Security

Next you get to the security settings, on line 427.

sl_wisun_app_core.c

```
426   // security settings
427   ret = _app_wisun_security_setting();
428   if (ret != SL_STATUS_OK) {
429     _return_and_mtx_release();
430   }
431
```

Going to the declaration of `_app_wisun_security_setting()`, you find the calls to set:

- The CA (Certification Authority) certificate
- The Device's certificate
- The Device Private Key

sl_wisun_app_core.c

```
708 sl_status_t _app_wisun_security_setting(void)
709 {
710   sl_status_t ret = SL_STATUS_FAIL;
711   const uint32_t max_cert_str_len = 2048U;
712
713   // sets the trusted certificate
714   ret = sl_wisun_set_trusted_certificate(SL_WISUN_CERTIFICATE_OPTION_IS_REF,
715                                          _get_cert_str_len(wisun_config_ca_certificate, max_cert_str_len) + 1,
716                                          wisun_config_ca_certificate);
717   if (ret != SL_STATUS_OK) {
718     printf("[Failed: unable to set the tr
719     _app_wisun_core_set_error(SET_TRUSTED
720     return ret;
721   }
722
723   // sets the device certificate
724   ret = sl_wisun_set_device_certificate(S
725                                         _
726                                         w
727   if (ret != SL_STATUS_OK) {
728     printf("[Failed: unable to set the de
729     _app_wisun_core_set_error(SET_DEVICE_
730     return ret;
731   }
732
733   // sets the device private key
734   // NOTE: to use a wrapped PSA private key, the app needs to import the key
735   // and use the API sl_wisun_set_device_private_key_id() instead of the one below
736   ret = sl_wisun_set_device_private_key(SL_WISUN_PRIVATE_KEY_OPTION_IS_REF,
737                                         _get_cert_str_len(wisun_config_device_private_key, max_cert_str_len) + 1,
738                                         wisun_config_device_private_key);
739   if (ret != SL_STATUS_OK) {
740     printf("[Failed: unable to set the device private key: %lu]\n", ret);
741     _app_wisun_core_set_error(SET_DEVICE_PRIVATE_KEY_ERROR_FLAG_BIT);
742     return ret;
743   }
744
745   return ret;
746 }
```

```
//! Wi-SUN CA certificate
const uint8_t wisun_config_ca_certificate[] = {
"-----BEGIN CERTIFICATE-----\r\n"
"MIIBojCCAUmgAwIBAgIUSOJfgI08JDWdAjuqvH3REMyjjFswCgYIKoZIzj0EAwIw\r\n"
"HjEcMBoGA1UEAwwTV2ktU1VOIERlbW8gUm9vdCBDQTAgFw0yMTAyMjIwOTU5NDFa\r\n"
"GA85OTk5MTIzMTIzNTk1OVowHjEcMBoGA1UEAwwTV2ktU1VOIERlbW8gUm9vdCBD\r\n"
"QTBZMBMGByqGSM49AgEGCCqGSM49AwEHA0IABG1Mn4dd9+IVJZSEcjpFKehvvRyQ\r\n"
"t9QcIBCN2ysf+BJUlFfU8TvC3w2waFrLuC+JHM+1TBEm1GLNDF7piCgqltWjYzBh\r\n"
"MBIGA1UdEwEB/wQIMAYBAf8CAQIwCwYDVR0PBAQDAgEGMB0GA1UdDgQWBBSTZNQO\r\n"
"92ii7l3wrPpyvMUfTU86JDAfBgNVHSMEGDAWgBSTZNQO92ii7l3wrPpyvMUfTU86\r\n"
"JDAKBggqhkjOPQQDAgNHADBEAiAdlM3ENdd7GHHbTsTiZMc7T5DDFQ2abeUI1be+\r\n"
"ytGaAAIgZREIYV4yhjoluqT4+snj/zQkqEqcYh/DMbx2gLKDgZ4=\r\n"
                                                    Press 'F2' for focus
```

Hovering over the corresponding variables, you can check their values. You can also open their declarations, which you find in `autogen/sl_wisun_config.c`. As you've already seen, these can be set on the Wi-SUN Configurator's **Security** tab.

> TIP: For a production environment, the Device certificate and Private Key need to be unique per device.

Joining a Network with a Given Name and PHY Settings

Finally, you reach the point where you call the Wi-SUN Stack API `sl_wisun_join()` function and check its return value.

This is where you use the Network Name you set earlier (using the Wi-SUN Configurator) in `autogen/sl_wisun_config.h` as `WISUN_CONFIG_NETWORK_NAME`.

sl_wisun_app_core.c

```
438
439   ret = sl_wisun_join((const uint8_t *)_setting.network_name, &_setting.phy);
440
441   if (ret == SL_STATUS_OK) {
442     // update internal time stat
443     sl_sleeptimer_tick64_to_ms(sl_sleeptimer_get_tick_count64(), &time_ms);
444     _time_stat.curr_ms = time_ms;
445     _time_stat.connected_ms = time_ms;
446     _time_stat.disconnected_ms = time_ms;
447
448     printf("\n[Connecting to \"%s\"]\n", _setting.network_name);
449   } else {
450     _app_wisun_core_set_error(CONNECTION_FAILED_ERROR_FLAG_BIT);
451     printf("\n[Connection failed: %lu]\n", ret);
452   }
453   _app_wisun_mutex_release();
454 }
```

As you see in the code, the `sl_wisun_join()` function returns rapidly, before connection is complete, and then there is a check of the `sl_wisun_join()` return value.

- If the call is successful, it means that your network settings are all okay and match the available PHY(s) that you added using the Wi-SUN Configurator.
  - A timestamp is set to the current time to indicate how much time it takes to connect.
- If there is an error, most probably the PHY you selected is not in the list of PHYs in your configuration.
  - If you have been using the Wi-SUN Configurator and the default PHY, this should not happen.

Waiting for Connection

To check the connection state, follow the join state of the Wi-SUN device. This is done by `app_wisun_wait_for_connection()`.

sl_wisun_app_core_util.c

```
89  /* Connect and wait */
90  void app_wisun_connect_and_wait(void)
91  {
92    app_wisun_network_connect();
93    app_wisun_wait_for_connection();
94  }
95
```

sl_wisun_app_core_util.c

```
 96 /* Waiting for connection */
 97 void app_wisun_wait_for_connection(void)
 98 {
 99 #if (HEARTBEAT_ENABLED == 1)
100   uint8_t dot_line_cnt = 0;
101 #endif
102   msleep(10);
103   printf("\n");
104   while (1) {
105     if (app_wisun_network_is_connected()) {
106       break;
107     }
108 #if (HEARTBEAT_ENABLED == 1)
109     printf((dot_line_cnt == HEARBEAT_SECTION_LENGTH ? "[%ds]\n" : "#"), dot_line_cnt);
110     dot_line_cnt = dot_line_cnt == HEARBEAT_SECTION_LENGTH ? 0 : dot_line_cnt + 1;
111 #endif
112     msleep(1000); // 1s
113   }
114 }
```

What you see here is that there is an endless loop calling `app_wisun_network_is_connected()`.

sl_wisun_app_core_util.c

```
115
116 bool app_wisun_network_is_connected(void)
117 {
118   sl_wisun_join_state_t join_state = app_wisun_get_join_state();
119   return join_state == SL_WISUN_JOIN_STATE_OPERATIONAL ? true : false;
120 }
121
```

`app_wisun_network_is_connected()` checks the join_state and returns only when it's `SL_WISUN_JOIN_STATE_OPERATIONAL` = (`5`).

The join_state is returned by `app_wisun_get_join_state()`.

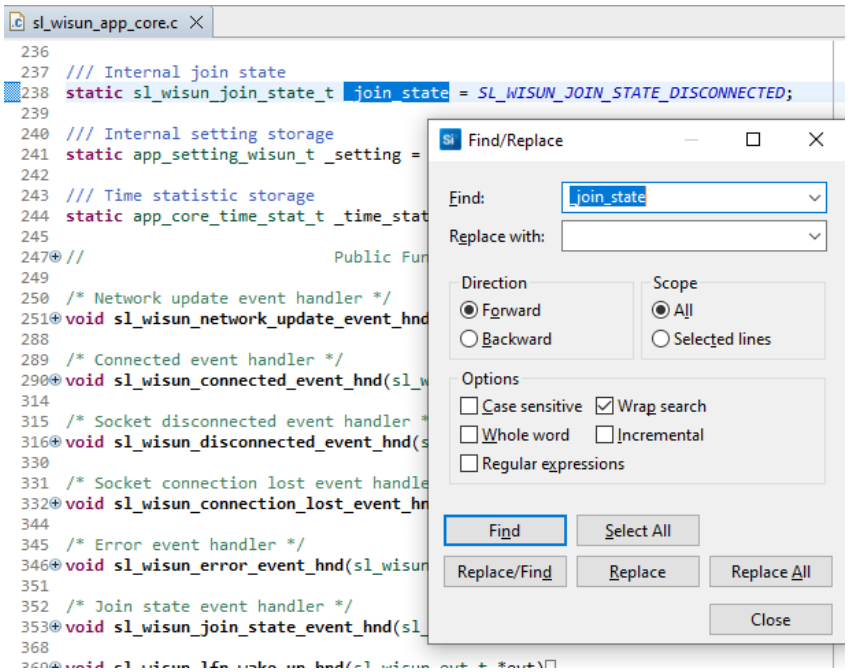
sl_wisun_app_core.c

```
529
530 sl_wisun_join_state_t app_wisun_get_join_state(void)
531 {
532   sl_wisun_join_state_t join_state = SL_WISUN_JOIN_STATE_DISCONNECTED;
533   _app_wisun_mutex_acquire();
534   join_state = _join_state;
535   _app_wisun_mutex_release();
536   return join_state;
537 }
538
```

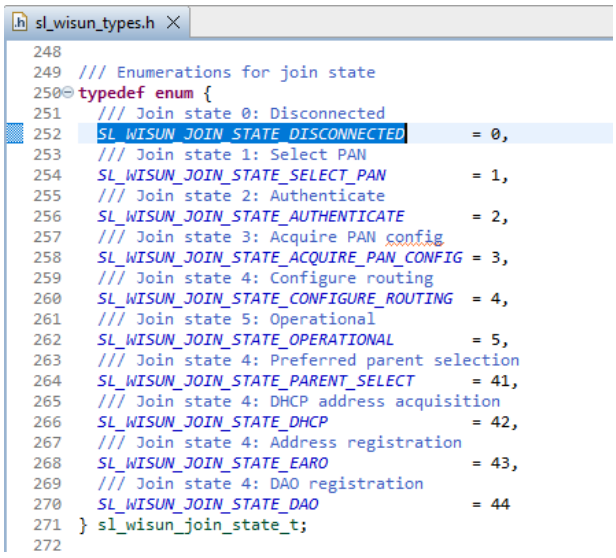
`app_wisun_get_join_state()` acquires the mutex to read `_join_state` (note that there is a `_` prefix, indicating that `_join_state` is an internal variable), stores it into `join_state`, releases the mutex, and returns `join_state`.

Using the mutex means that `_join_state` is set by another part of the code. To see how, look for occurrences of `_join_state`.

First, it's declared and initialized with `SL_WISUN_JOIN_STATE_DISCONNECTED`.



Possible join state values are defined in `gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`.



The `sl_wisun_join_state_t` enumeration tells us what steps are used for connecting (re-ordered below by chronological order):

Join state	enum	meaning	actions
0	SL_WISUN_JOIN_STATE_DISCONNECTED	Disconnected	Device not attempting to connect. Border Router and connected routers send 'PAN Advert' on all channels
1	SL_WISUN_JOIN_STATE_SELECT_PAN	Select PAN	Device listens for 'PAN Advert'. Sends 'PAN Advert Solicit' to shorten the transmission delay

Join state	enum	meaning	actions
2	SL_WISUN_JOIN_STATE_AUTHENTICATE	Authenticate	Security key exchange between device and Border Router (can use an external Radius server)
3	SL_WISUN_JOIN_STATE_ACQUIRE_PAN_CONFIG	Acquire PAN config	Device listens for 'PAN Config'. Sends 'PAN Advert Solicit' to shorten the transmission delay
4	SL_WISUN_JOIN_STATE_CONFIGURE_ROUTING	Configure routing	Network Route Selection (see sub-steps below)
41	SL_WISUN_JOIN_STATE_PARENT_SELECT	Preferred parent selection	Comparison between potential parents. Selection of 'best parent'
42	SL_WISUN_JOIN_STATE_DHCP	DHCP address acquisition	DHCP address received from Border Router
43	SL_WISUN_JOIN_STATE_EARO	Address registration	
44	SL_WISUN_JOIN_STATE_DAO	DAO registration	
5	SL_WISUN_JOIN_STATE_OPERATIONAL	Operational	Device is operational. Starts sending 'PAN Advert' for other nodes to connect (if FFN/router)

- By default, Wi-SUN SoC Empty doesn't track the intermediate 41-44 sub-steps between join state 4 and join state 5.
- When reconnecting, provided that the credentials are still valid, the connection process is sped up by avoiding steps 1 and 2.

Now look at how events are handled, and the concepts behind the Wi-SUN Stack API.

Wi-SUN Stack API Concepts

As stated in the Wi-SUN Stack API documentation:

- Wi-SUN Stack API is based on **requests from the application** to the stack and **events from the stack** to the application.
- Requests are made using function calls, where a function call either performs the required action immediately or initiates an internal operation within the stack, which terminates with an event. All events contain a status code, indicating the result of the requested operation. Events are also used by the stack to notify the application of any important information, such as the state of the connection.
- **The application is expected to override `sl_wisun_on_event()` to handle events from the stack**. Because all events share a common header, the function may be implemented as a switch statement. The event-specific data can be accessed through the `sl_wisun_evt_t::evt` union.

Event Handling with the Event Manager Component

`gsdk/protocol/wisun/stack/inc/sl_wisun_events.h` contains the list of possible events from the stack.

sl_wisun_events.h

```
559 /// @brief Wi-SUN event definitions
560 /// @details This structure contains a Wi-SUN API event and its associated data.
561 SL_PACK_START(1)
562 typedef struct {
563   /// @brief Common event header
564   /// @details This structure contains common information for all events.
565   ///          ID of the event is stored in the sl_wisun_msg_header_t.id
566   ///          field and is one of the values of @ref sl_wisun_msg_ind_id_t.
567   ///          The other fields can be ignored.
568   sl_wisun_msg_header_t header;
569   /// @brief Event-specific data
570   /// @details This structure contains the event-specific data.
571   union {
572     /// #SL_WISUN_MSG_CONNECTED_IND_ID event data
573     sl_wisun_msg_connected_ind_body_t connected;
574     /// #SL_WISUN_MSG_SOCKET_DATA_IND_ID event data
575     sl_wisun_msg_socket_data_ind_body_t socket_data;
576     /// #SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID event data
577     sl_wisun_msg_socket_data_available_ind_body_t socket_data_available;
578     /// #SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID event data
579     sl_wisun_msg_socket_connected_ind_body_t socket_connected;
580     /// #SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID event data
581     sl_wisun_msg_socket_connection_available_ind_body_t socket_connection_available;
582     /// #SL_WISUN_MSG_SOCKET_CLOSING_IND_ID event data
583     sl_wisun_msg_socket_closing_ind_body_t socket_closing;
584     /// #SL_WISUN_MSG_DISCONNECTED_IND_ID event data
585     sl_wisun_msg_disconnected_ind_body_t disconnected;
586     /// #SL_WISUN_MSG_CONNECTION_LOST_IND_ID event data
587     sl_wisun_msg_connection_lost_ind_body_t connection_lost;
588     /// #SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID event data
589     sl_wisun_msg_socket_data_sent_ind_body_t socket_data_sent;
590     /// #SL_WISUN_MSG_ERROR_IND_ID event data
591     sl_wisun_msg_error_ind_body_t error;
592     /// #SL_WISUN_MSG_JOIN_STATE_IND_ID event data
593     sl_wisun_msg_join_state_ind_body_t join_state;
594     /// #SL_WISUN_MSG_NETWORK_UPDATE_IND_ID event data
595     sl_wisun_msg_network_update_ind_body_t network_update;
596     /// #SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID event data
597     sl_wisun_msg_regulation_tx_level_ind_body_t regulation_tx_level;
598     /// #SL_WISUN_MSG_MODE_SWITCH_FALLBACK_IND_ID event data
599     sl_wisun_msg_mode_switch_fallback_ind_body_t mode_switch_fallback;
600     /// #SL_WISUN_MSG_RX_FRAME_IND_ID event data
601     sl_wisun_msg_rx_frame_ind_body_t rx_frame;
602     /// #SL_WISUN_MSG_LFN_WAKE_UP_IND_ID event data
603     sl_wisun_msg_lfn_wake_up_ind_body_t lfn_wake_up;
604     /// #SL_WISUN_MSG_LFN_MULTICAST_REG_IND_ID event data
605     sl_wisun_msg_lfn_multicast_reg_ind_body_t lfn_multicast_reg;
606   } evt;
607 } SL_ATTRIBUTE_PACKED sl_wisun_evt_t;
```

The first member of the `sl_wisun_event_t` structure is a `sl_wisun_msg_header_t` header.

sl_wisun_types.h

```
321 /// Wi-SUN Message API common header
322 SL_PACK_START(1)
323 typedef struct {
324   /// Total length of the message in bytes, this field included
325   uint16_t length;
326   /// ID (request, confirmation, indication) of the message
327   uint8_t id;
328   /// Processing metadata for the message
329   uint8_t info;
330 } SL_ATTRIBUTE_PACKED sl_wisun_msg_header_t;
331 SL_PACK_END()
```

From this header, the application needs to check the `id` value to know what the `evt` corresponds to, and then use the rest of the structure to get access to the event data.

Now see how this is implemented in `gsdk/app/wisun/component/event_manager/sl_wisun_event_mgr.c/sl_wisun_on_event()`.

sl_wisun_event_mgr.c

```
260 /* Wi-SUN event callback */
261 void sl_wisun_on_event(sl_wisun_evt_t *evt)
262 {
263   app_wisun_event_id_t idx = _decode_ind((sl_wisun_msg_ind_id_t) evt->header.id);
264
265   app_wisun_event_mgr_mutex_lock();
266
267   if (EVENT_IDX_NOTVALID == idx) {
268     printf("[Unknown event: %d]\r\n", evt->header.id);
269   } else {
270     _wisun_events[idx].callback(evt);
271     if (_wisun_events[idx].custom_callback != NULL) {
272       _wisun_events[idx].custom_callback(evt);
273     }
274   }
275
276   app_wisun_event_mgr_mutex_unlock();
277 }
```

On line `263`, the `_decode_ind()` function is checking the `evt->header.id` with the list of the events your application has decided to process (i.e. 'handle').

sl_wisun_event_mgr.c

```
295 /* Decode wisun event id to lookup index */
296 __STATIC_INLINE app_wisun_event_id_t _decode_ind(const sl_wisun_msg_ind_id_t ind)
297 {
298   switch (ind) {
299     case SL_WISUN_MSG_NETWORK_UPDATE_IND_ID:              return EVENT_IDX_NETWORK_UPDATE;
300     case SL_WISUN_MSG_CONNECTED_IND_ID:                   return EVENT_IDX_CONNECTED;
301     case SL_WISUN_MSG_SOCKET_DATA_IND_ID:                 return EVENT_IDX_SOCKET_DATA;
302     case SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID:       return EVENT_IDX_SOCKET_DATA_AVAILABLE;
303     case SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID:            return EVENT_IDX_SOCKET_CONNECTED;
304     case SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID: return EVENT_IDX_SOCKET_CONNECTION_AVAILABLE;
305     case SL_WISUN_MSG_SOCKET_CLOSING_IND_ID:              return EVENT_IDX_SOCKET_CLOSING;
306     case SL_WISUN_MSG_DISCONNECTED_IND_ID:                return EVENT_IDX_DISCONNECTED;
307     case SL_WISUN_MSG_CONNECTION_LOST_IND_ID:             return EVENT_IDX_CONNECTION_LOST;
308     case SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID:            return EVENT_IDX_SOCKET_DATA_SENT;
309     case SL_WISUN_MSG_ERROR_IND_ID:                       return EVENT_IDX_ERROR;
310     case SL_WISUN_MSG_JOIN_STATE_IND_ID:                  return EVENT_IDX_JOIN_STATE;
311     case SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID:         return EVENT_IDX_REGULATION_TX_LEVEL;
312     case SL_WISUN_MSG_LFN_WAKE_UP_IND_ID:                 return EVENT_IDX_LFN_WAKE_UP;
313     default:                                              return EVENT_IDX_NOTVALID;
314   }
315 }
```

> TIP: See the last value in the table, clearly indicating that any value not matching the preceding lines will return `EVENT_IDX_NOTVALID`.

In `sl_wisun_on_event()`:

- A message is printed whenever an 'invalid' (i.e. unprocessed) event is received for information purposes. Since this is informational, it's only displayed as an `Unknown event`.
- If the event is one of those that the application decided to handle, the corresponding callback function in the Application Core Component is called with the `evt` data.
  - If a 'custom callback' function exists in the customer application , it is also called.

In `GSDK/app/wisun/component/event_manager/sl_wisun_event_mgr.c`:

- You see all `.callback` functions
- You see that no `custom_callback` is defined by default

sl_wisun_event_mgr.c

```c
/*******************************************************************************//**
 * @brief Event Lookup table
 *        Lookup table indexes must be matched with app_wisun_event_id_t!
 ******************************************************************************/
static event_handler_t _wisun_events[] = {
  {
    .id = SL_WISUN_MSG_NETWORK_UPDATE_IND_ID,
    .callback = sl_wisun_network_update_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_CONNECTED_IND_ID,
    .callback = sl_wisun_connected_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_SOCKET_DATA_IND_ID,
    .callback = sl_wisun_socket_data_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID,
    .callback = sl_wisun_socket_data_available_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID,
    .callback = sl_wisun_socket_connected_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID,
    .callback = sl_wisun_socket_connection_available_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_SOCKET_CLOSING_IND_ID,
    .callback = sl_wisun_socket_closing_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_DISCONNECTED_IND_ID,
    .callback = sl_wisun_disconnected_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_CONNECTION_LOST_IND_ID,
    .callback = sl_wisun_connection_lost_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID,
    .callback = sl_wisun_socket_data_sent_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_ERROR_IND_ID,
    .callback = sl_wisun_error_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_JOIN_STATE_IND_ID,
    .callback = sl_wisun_join_state_event_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID,
    .callback = sl_wisun_regulation_tx_level_hnd,
    .custom_callback = NULL
  },
  {
    .id = SL_WISUN_MSG_LFN_WAKE_UP_IND_ID,
    .callback = sl_wisun_lfn_wake_up_hnd,
    .custom_callback = NULL
  }
};
```

The Wi-SUN Event Manager component provides `app_wisun_em_custom_callback_register()` to register custom callbacks.

If you want to add some custom processing on join state changes:

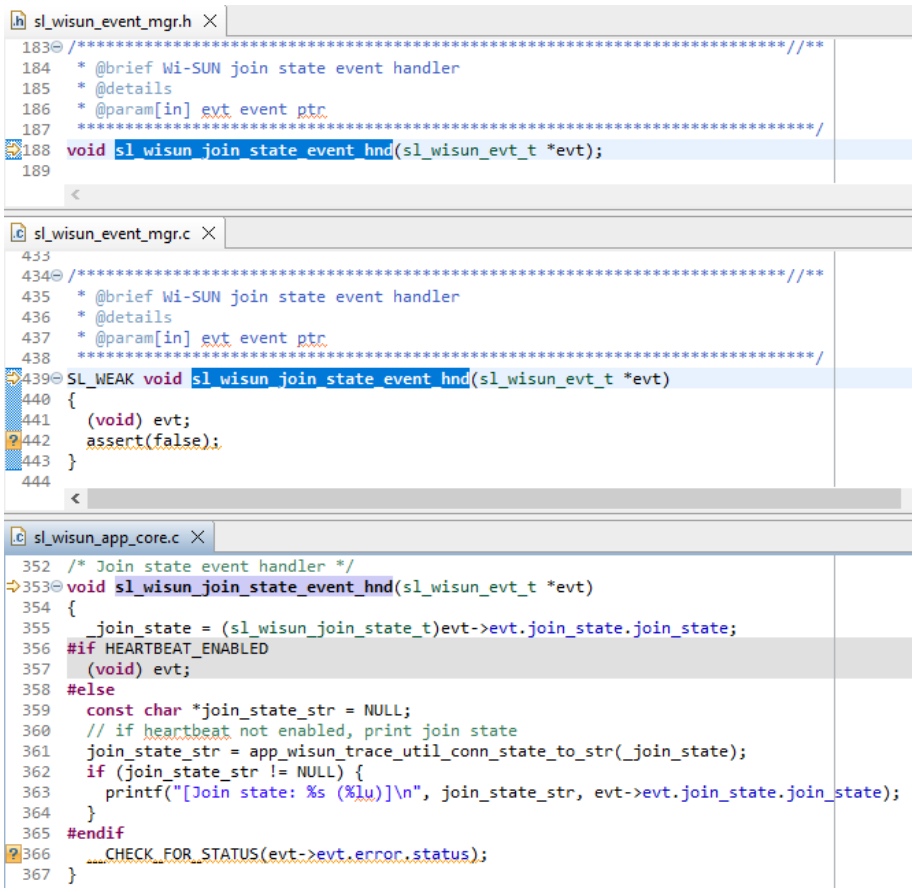
- You create a custom callback function and add into this function your custom actions.

```
void join_state_custom_callback(sl_wisun_evt_t *evt) {
  sl_wisun_join_state_t join_state;
  join_state = (sl_wisun_join_state_t)evt->evt.join_state.join_state;
  printf("Join State %d\n", join_state);
}
```

- You call `app_wisun_em_custom_callback_register(SL_WISUN_MSG_JOIN_STATE_IND_ID, join_state_custom_callback);` to register it.

The callback function for join state changes is `sl_wisun_join_state_event_hnd()`.

You can see it in three places:



- Defined in `gsdk/app/wisun/component/event_manager/sl_wisun_event_mgr.h` (describing the prototype of the function).
- Declared as `SL_WEAK` in `gsdk/app/wisun/component/event_manager/sl_wisun_event_mgr.c`.
  - A 'weak' declaration (i.e. implementation) of a function is a default implementation, which can be superseded by another located elsewhere in the code.
  - Most 'weak' functions do nothing. They are there to avoid compilation errors when there is no other implementation.
- Declared as a normal function in `gsdk/app/wisun/component/app_core/sl_wisun_app_core.c`.
  - This implementation supersedes the 'weak' implementation.

Finally, check the `sl_wisun_join_state_event_hnd()` function in `gsdk/app/wisun/component/app_core/sl_wisun_app_core.c` to follow what it does.

sl_wisun_app_core.c

```
352 /* Join state event handler */
353 void sl_wisun_join_state_event_hnd(sl_wisun_evt_t *evt)
354 {
355   _join_state = (sl_wisun_join_state_t)evt->evt.join_state.join_state;
356 #if HEARTBEAT_ENABLED
357   (void) evt;
358 #else
359   const char *join_state_str = NULL;
360   // if heartbeat not enabled, print join state
361   join_state_str = app_wisun_trace_util_conn_state_to_str(_join_state);
362   if (join_state_str != NULL) {
363     printf("[Join state: %s (%lu)]\n", join_state_str, evt->evt.join_state.join_state);
364   }
365 #endif
366   __CHECK_FOR_STATUS(evt->evt.error.status);
367 }
```

- This 'event handler' is called only when there are changes to the internal join state.
- `_join_state` is set on line `355` in the 'event handler' function `sl_wisun_join_state_event_hnd()`, as `evt->evt.join_state.join_state`.
- If there is a change of the join_state, the `app_wisun_trace_util_conn_state_to_str()` function returns a valid string to translate the state's decimal value into a human-readable string. If this string is not empty, it is traced in the application's console.
  - Printing a message is useful to follow the connection process when having access to the device's UART console. It's not actually required to achieve connection, but it is convenient.

Remarks on Function Names

- All Wi-SUN Stack API functions are prefixed with `sl_wisun_`.
  - Customers can't read the underlying Stack source code, which is under the responsibility of the Wi-SUN Stack developers.
  - The `OK` return code is `SL_STATUS_OK` = `0x0000`.
  - All return codes are defined in `gsdk/platform/common/inc/sl_status.h`. Checking this file in case of an error can help get minimal information on the error.
  - Wi-SUN Stack functions don't print traces in the device's UART console, instead in the form of RTT traces, accessible using J-LINK connected to the device via an evaluation kit (which embed SEGGER debug capabilities) or a external debugger.
- GSDK functions ultimately use the `sl_wisun_` Wi-SUN Stack API functions.
- GSDK component top-level API functions are also prefixed with `sl_wisun_`.
- If there are 'GSDK-internal' functions in the components, they are prefixed with `sli_`, the `i` meaning 'internal'.
- Additional API functions provided by GSDK components to be called by customer application code are prefixed with `app_`.
- As is common usage in C coding, functions that are internal to a single source file are prefixed with `_`, meaning that they are not intended to be called by other parts of the code.
- Customers have access to the GSDK function's source code.

Takeaway

What you have accomplished in this session is the join process available when you add the **Wi-SUN Application Core Component** to your project.

- The Wi-SUN Application Core Component uses calls to these Wi-SUN Stack API functions, in the following order (minimal set of calls for connecting to a basic Wi-SUN network as FFN). It also checks the return values for all calls to raise errors if needed:
  - `sl_wisun_set_device_type()`
  - `sl_wisun_set_connection_parameters()`
  - `sl_wisun_set_tx_power()`
  - `sl_wisun_set_trusted_certificate()`
  - `sl_wisun_set_device_certificate()`
  - `sl_wisun_set_device_private_key()`
  - `sl_wisun_join()`
- If the call to `sl_wisun_join()` fails, most probably there is an issue with your PHY selection.
  - If you use a single PHY and the Wi-SUN Configurator, this should not happen.

- Once the join call is made, connection starts. The customer application should wait for the connection before acting, since it can't communicate with the rest of the Wi-SUN network until it's connected.
- An application can directly use the Wi-SUN Stack API functions, while using the **Wi-SUN Application Core Component** makes the join process much easier.
- Calling `sl_wisun_join()` returns rapidly, once connection settings are set.
- The application communicates with the Wi-SUN Stack using function calls.
- The Wi-SUN Stack communicates with the application using events.
- As a consequence, following the connection process from the application is done using the `SL_WISUN_MSG_JOIN_STATE_IND_ID` event message.
  - Triggered by the Wi-SUN Stack.
  - That the application can monitor using an event-specific handler.

It's now time to add a custom application to a Wi-SUN network.

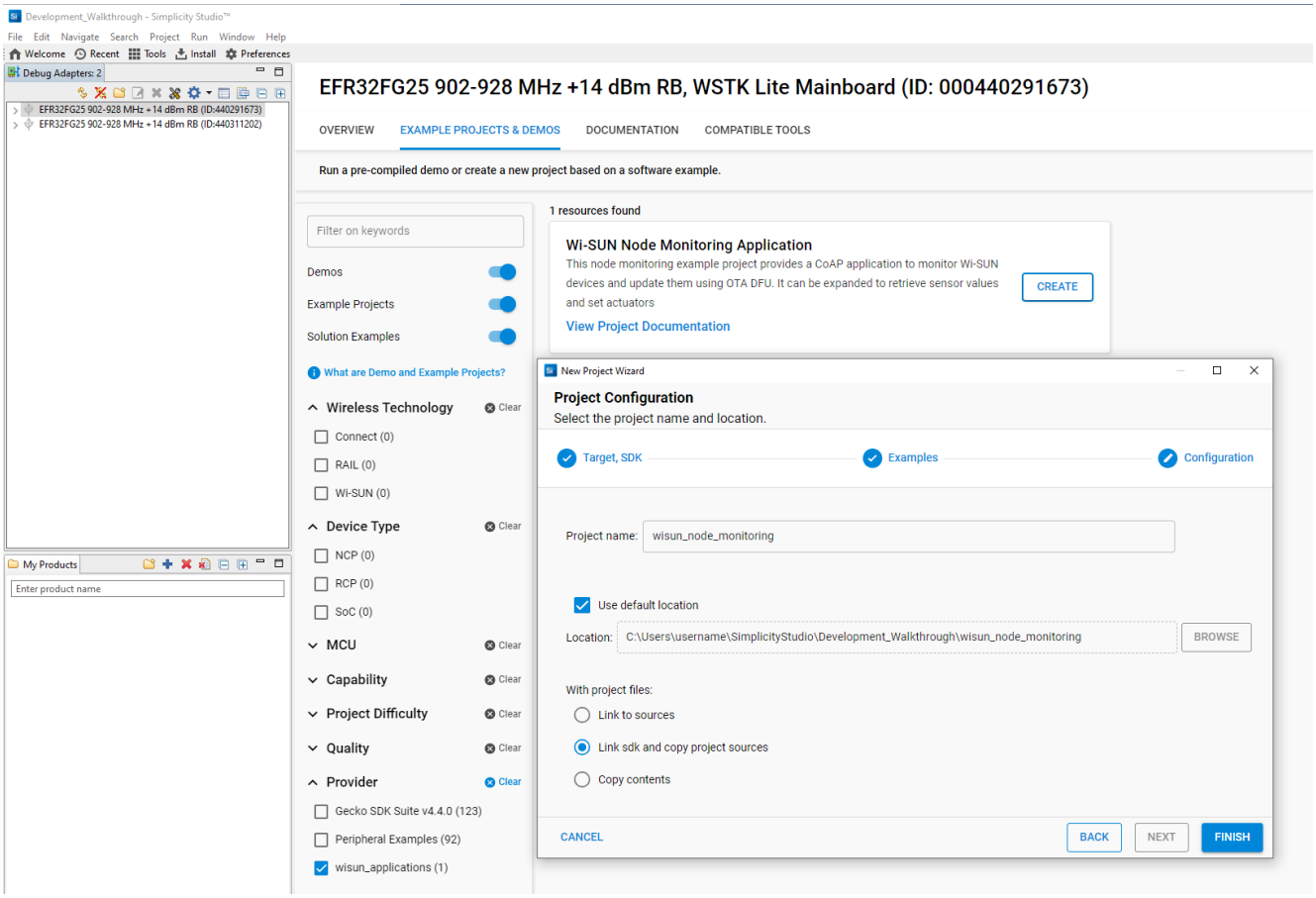
Add a Custom Application

Add a Custom Application

The example is the **Wi-SUN Node Monitoring** example application The reference code is provided on GitHub.

The example code being available on GitHub, it is recommended to:

- Add the 'Wi-SUN Applications' repository to Simplicity Studio
- Create a Wi-SUN Node Monitoring project after selecting the **wisun_applications** provider.



Using the **Wi-SUN Node Monitoring** project removes the difficulties related to adding pieces of code along the way to reach a compilable project.

The goal of this application is to:

- Follow the device connections from the Border Router.
  - The devices will send a UDP connection message to the Border Router on port 1237 after connecting for the first time.
  - The devices will automatically send a connection status message to the Border Router every `auto_send` seconds.
    - The default `auto_send` period is 30 seconds.
    - The `auto_send` period can be checked remotely via a CoAP `get` command for `settings\auto_send`.
    - It can be modified using the same CoAP command if adding `-e <n>` as the command payload, `<n>` being the new value.
    - This can be used as an example for sensor monitoring or actuator control in a final application.
    - The connection message contains time values in `dd-hh:mm:ss` format.
    - The start point of these time values is the moment the application is started.
    - The first connection message can be used to check the connection time.
- Follow the device join state changes.
  - The history of these changes can be retrieved from the Border Router once connected.
- Monitor application statistics.
  - Application statistics can be retrieved from the Border Router using CoAP requests.
  - A very important statistic is the availability rate, computed from the total connected and disconnected times, which should be as close to 100% as possible (counting starts on the very first connection).
- Monitor Wi-SUN Stack statistics.
  - All statistics provided by the Wi-SUN Stack are available via CoAP requests.
- Use OTA DFU to remotely update the WI-SUN devices via the Wi-SUN network.

Prerequisites

Use Git to Store the Current State

Before adding code, it is useful to use Git to track code changes, to be able to easily check the differences later on, and roll the code back if required.

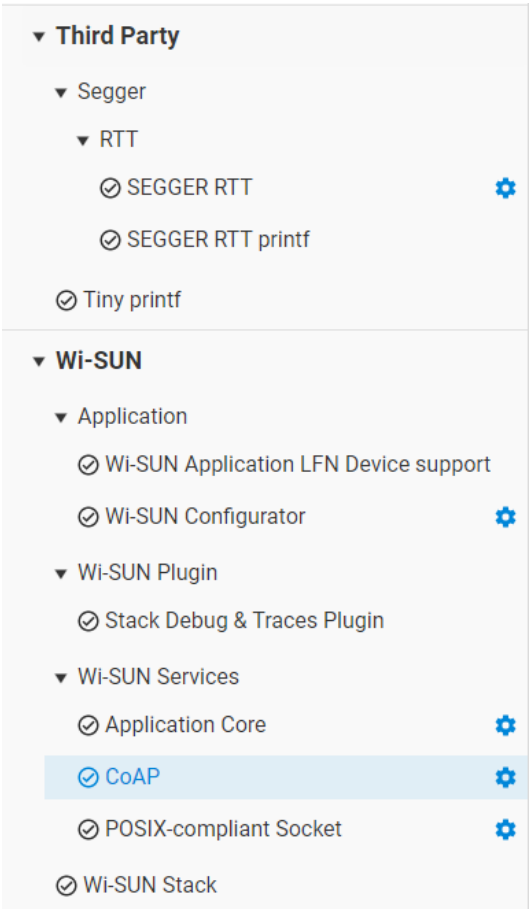
Software Components

The following software components are required to compile the application with the added code:

- Third-Party / Segger / SEGGER RTT printf
- Wi-SUN / Wi-SUN Services / CoAP
- Wi-SUN / Wi-SUN Services / PoSIX-compliant Socket
- Wi-SUN / Wi-SUN Services / Over-The-Air Firmware Upgrade (OTA DFU)

> Note: It is possible to check which components are installed in a project looking at the `.slcp` file and the **SOFTWARE COMPONENTS** tab, checking the **Installed** box.

Using the **Wi-SUN Node Monitoring** example application, all components should already be installed. Just check their presence to get familiar with components used. Install the components now if they are not present.



Installed Wi-SUN components are listed in `autogen\sl_component_catalog.h` :

sl_component_catalog.h

```
#ifndef SL_COMPONENT_CATALOG_H
#define SL_COMPONENT_CATALOG_H

// APIs present in project
#define SL_CATALOG_APP_PROJECT_INFO_PRESENT
#define SL_CATALOG_GECKO_BOOTLOADER_INTERFACE_PRESENT
#define SL_CATALOG_CMSIS_OS_COMMON_PRESENT
#define SL_CATALOG_DEVICE_INIT_PRESENT
#define SL_CATALOG_DEVICE_INIT_CLOCKS_PRESENT
#define SL_CATALOG_DEVICE_INIT_CORE_PRESENT
#define SL_CATALOG_DEVICE_INIT_DCDC_PRESENT
#define SL_CATALOG_DEVICE_INIT_EMU_PRESENT
#define SL_CATALOG_DEVICE_INIT_HFXO_PRESENT
#define SL_CATALOG_DEVICE_INIT_LFXO_PRESENT
#define SL_CATALOG_DEVICE_INIT_NVIC_PRESENT
#define SL_CATALOG_DEVICE_INIT_RFFPLL_PRESENT
#define SL_CATALOG_DEVICE_INIT_USBPLL_PRESENT
#define SL_CATALOG_EMLIB_CORE_PRESENT
#define SL_CATALOG_EMLIB_CORE_DEBUG_CONFIG_PRESENT
#define SL_CATALOG_IOSTREAM_PRESENT
#define SL_CATALOG_IOSTREAM_EUSART_PRESENT
#define SL_CATALOG_RETARGET_STDIO_PRESENT
#define SL_CATALOG_IOSTREAM_UART_COMMON_PRESENT
#define SL_CATALOG_KERNEL_PRESENT
#define SL_CATALOG_MICRIUMOS_KERNEL_PRESENT
#define SL_CATALOG_MX25_FLASH_SHUTDOWN_EUSART_PRESENT
#define SL_CATALOG_NVM3_PRESENT
#define SL_CATALOG_PRINTF_PRESENT
#define SL_CATALOG_PSA_CRYPTO_PRESENT
#define SL_CATALOG_RADIO_CONFIG_SIMPLE_WISUN_SINGLEPHY_PRESENT
#define SL_CATALOG_RAIL_LIB_PRESENT
#define SL_CATALOG_RAIL_UTIL_PTI_PRESENT
#define SL_CATALOG_SE_MANAGER_PRESENT
#define SL_CATALOG_FTP_PRESENT
#define SL_CATALOG_MEMPOOL_PRESENT
#define SL_CATALOG_WISUN_APP_BRD_FEATURE_PRESENT
#define SL_CATALOG_WISUN_APP_CORE_PRESENT
#define SL_CATALOG_WISUN_COAP_PRESENT
#define SL_CATALOG_WISUN_EVENT_MGR_PRESENT
#define SL_CATALOG_FTP_WISUN_POSIX_PORT_PRESENT
#define SL_CATALOG_WISUN_FULL_CONFIG_PRESENT
#define SL_CATALOG_WISUN_NS_LIST_PRESENT
#define SL_CATALOG_WISUN_OTA_DFU_PRESENT
#define SL_CATALOG_WISUN_TRACE_UTIL_PRESENT
#define SL_CATALOG_SLEEPTIMER_PRESENT
#define SL_CATALOG_WISUN_CONFIG_PRESENT
#define SL_CATALOG_WISUN_CRYPTO_PRESENT
#define SL_CATALOG_WISUN_LEGACY_SOCKET_WRAPPER_PRESENT
#define SL_CATALOG_WISUN_STACK_PRESENT

#endif // SL_COMPONENT_CATALOG_H
```

Custom Application

The Custom Application pages explain how the custom application features are added to a Wi-SUN project. These features can be added to multiple projects if needed.

- In the Device Application
  - Add time stamping and tracing functions
  - Register a custom callback to follow join state changes
  - Prepare JSON connection strings
  - Send status strings to the Border Router's IPv6 on UDP port 1237
  - Add CoAP resources to the Device Application
- On the Border Router
  - Retrieve UDP notifications on the Border Router
  - Retrieve device information on the Border Router Using CoAP

Timestamping

One Second Timestamping

Because Wi-SUN network changes are rare while connection times are in days or months, the underlying unit of time measurement used by the Wi-SUN Stack down to the microsecond is not necessary for long-term observability of network connection. We therefore:

- Set up a 1-second timer to count seconds
- Add functions to format current time and delays as dd-hh:mm:ss

To get a 1-second timestamp, do the following steps.

> Note: This is all done in `app_timestamp.c` and `app_timestamp.h`, which you can use by copy/pasting both files next to your `app.c` file.

Simplicity Studio will automatically compile any `.c` file present under the project folder, so no action is required to add `app_timestamp.c` to the compilation once the file is copied.

Declaration of a 64-bit `app_timestamp` Variable

```
sl_sleeptimer_timestamp_64_t app_timestamp;
```

Creation of a Callback Function to Increment the Timestamp by 1

```
void app_timer_callback(sl_sleeptimer_timer_handle_t *handle, void *data) {
  (void)handle;
  (void)data;
  app_timestamp++;
}
```

Protect Access to the Variable Using a Mutex

- Mutex declaration:

```
static const osMutexAttr_t _app_timestamp_mutex_attr = {
  .name      = "AppTimestampMutex",
  .attr_bits = osMutexRecursive,
  .cb_mem    = NULL,
  .cb_size   = 0U
};
```

- Mutex acquire function:

```
__STATIC_INLINE void _app_timestamp_mutex_acquire(void)
{
  assert(osMutexAcquire(_app_timestamp_mutex, osWaitForever) == osOK);
}
```

- Mutex release function:

```
__STATIC_INLINE void _app_timestamp_mutex_release(void)
{
  assert(osMutexRelease(_app_timestamp_mutex) == osOK);
}
```

Timestamp Init

```
sl_status_t app_timestamp_init(void) {
  sl_status_t status;
  uint32_t app_timer_timeout;

  // init mutex
  _app_timestamp_mutex = osMutexNew(&_app_timestamp_mutex_attr);
  assert(_app_timestamp_mutex != NULL);

  app_timestamp =  0;

  status = sl_sleeptimer_init();
  if (status != SL_STATUS_OK) {
     printf("Error initializing sleeptimer. Status %lu\n", status);
     return status;
  }
  app_timer_timeout = (uint32_t)1.0*sl_sleeptimer_get_timer_frequency();
  status = sl_sleeptimer_start_periodic_timer(&app_timer,
                                  app_timer_timeout,
                                  app_timer_callback,
                                  NULL, 0, 0);
  if (status != SL_STATUS_OK) {
     printf("Error starting periodic timer 'app_timer'. Status %lu\n", status);
     return status;
  }
  return status;
}
```

Following this:

- `app_timestamp` is set to `0` when `app_timestamp_init()` is called
- `app_timer_callback()` is called every second
  - `app_timestamp` is increased by `1` every second
- You can retrieve it safely using the mutex

Retrieve the Current Timestamp Decimal Value

```
uint64_t    now_sec     (void) {
  sl_sleeptimer_timestamp_64_t current_sec;
  _app_timestamp_mutex_acquire();
  current_sec = app_timestamp;
  _app_timestamp_mutex_release();
  return (uint64_t)current_sec;
}
```

You can use `now_sec()` to store the current time in decimal values:

```
  uint64_t current_time_secs;
  current_time_secs = now_sec();
```

Utility Functions to Process Timestamp Values

Transform a Timestamp in days/hours/mins/secs Units

Mainly used as a convenience function, this should rarely be used from outside this code:

```
sl_status_t  d_h_m_s_total(sl_sleeptimer_timestamp_64_t timestamp_secs,
                uint16_t* days,
                uint64_t* hours,
                uint64_t* mins,
                uint64_t* secs
) {
  *days  = timestamp_secs / 60 / 60 / 24;
  *hours = timestamp_secs / 60 / 60;
  *mins  = timestamp_secs / 60;
  *secs  = timestamp_secs;
  return SL_STATUS_OK;
}
```

Transform a Timestamp as days:hours:mins:secs Decimal Values

```
sl_status_t  d_h_m_s     (sl_sleeptimer_timestamp_64_t timestamp_secs,
                uint16_t* days,
                uint8_t* hours,
                uint8_t* mins,
                uint8_t* secs) {
  uint64_t hours_total;
  uint64_t mins_total;
  uint64_t secs_total;

  d_h_m_s_total(timestamp_secs, days, &hours_total, &mins_total, &secs_total);

  *days = *days;
  *hours = hours_total % 24;
  *mins  = mins_total  % 60;
  *secs  = secs_total  % 60;

  return SL_STATUS_OK;
}
```

Format a Timestamp as a days:hours:mins:secs Text String

```
char*       dhms        (sl_sleeptimer_timestamp_64_t timestamp_secs) {
  uint16_t days;
  uint8_t  hours, mins, secs;

  d_h_m_s(timestamp_secs, &days, &hours, &mins, &secs);

  snprintf(time_str, TIME_STRING_LEN, "%3d:%02d:%02d:%02d", days, hours, mins, secs);

  return time_str;
}
```

This can be used in traces as follows:

```
printf("Current time is %s\n", now_str());
```

With a `Current time is 0:01:23:45` result, if the application has been running for 0 day, 1 hour, 23 minutes and 45 seconds.

Get `app_timestamp` as a Text String

```
char*       now_str    (void) {
 return dhms(now_sec());
}
```

Trace Timestamping and Routing Macros

A set of macros are defined to allow sending traces to the console, the RTT traces, or both:

```
#define printfTime(...)     printf("[%s] ", now_str()); printf(__VA_ARGS__)
#define printfRTT(...)      SEGGER_RTT_printf(0, __VA_ARGS__)
#define printfTimeRTT(...)  SEGGER_RTT_printf(0, "[%s] ", now_str()); SEGGER_RTT_printf(0, __VA_ARGS__)
#define printfBoth(...)     SEGGER_RTT_printf(0, __VA_ARGS__); printf(__VA_ARGS__)
#define printfBothTime(...) SEGGER_RTT_printf(0, "[%s] ", now_str()); SEGGER_RTT_printf(0, __VA_ARGS__); printf("[%s] ", now_str());
printf(__VA_ARGS__)
```

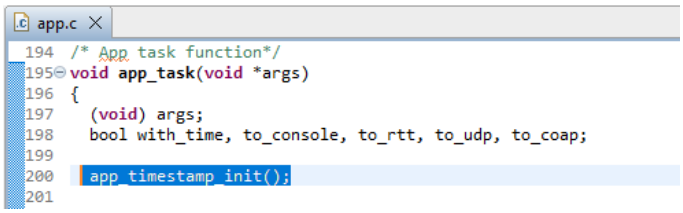
The above makes it easy in the application to:

- Select to add a timestamp or not to traces.
- Print a trace to the console or the RTT traces, or both.

Use the Timestamp

In `app.c`, `#include "app-timestamp.h"` to get access to the timestamping resources.

In `app.c/app_task()`, initialize the one second timestamp:



It is convenient to declare variables to store various time values:

```
uint64_t connect_time_sec;         // time stamp of Wisun connect call
uint64_t connection_time_sec;      // last connection time stamp
uint64_t disconnection_time_sec;   // last disconnection time stamp
```

These can be used as follows (example code, not part of the sample application):

```
printfBothTime("running since %s\n", now_str());
connect_time_sec = now_sec();
app_wisun_connect_and_wait();
/* Some time will pass until we're connected... */
connection_time_sec = now_sec();
printfBothTime("Connected in %s\n", dhms(connection_time_sec - connect_time_sec));
```

Custom Callback

Register a Custom Callback to Follow the Join State

As described in Event Handling with the Event Manager, it is possible to register custom callbacks for indications listed in `sl_wisun_events.h/sl_wisun_msg_ind_id_t`.

> Note: This is all done in `app.c`.

To follow what is happening on your Wi-SUN network, a dedicated function is added to do whatever you need and is registered to be called on each join state change.

It is registered with the Event Manager in `app.c/app_task()` as follows:

```
// Register our join state custom callback function with the event manager (aka 'em')
app_wisun_em_custom_callback_register(SL_WISUN_MSG_JOIN_STATE_IND_ID, _join_state_custom_callback);
```

The callback function itself is `app.c/_join_state_custom_callback()`, relying on our 1-second timestamping to store time information. The function uses other parts of the code, so replacing the original `app.c` by this file is an easy way to get compilable code.

Code Behavior

- Check the join state.
- Only do something when there is a join state change.
  - Do nothing for intermediate join states (with values above 5), which the Stack also handles. These are mainly intermediate stages between join state 4 and join state 5, such as DHCP and DNS, with code between 41 and 49.
  - Trace the previous and current join state when it changed.
  - Store the current time in `app_join_state_sec[join_state]`. This will be used in traces and to compute the `app_join_state_delay_sec[join_state]` values, indicating how much time was spent moving to the new join state.
  - If the join state is 'OPERATIONAL' (i.e. 'connected'):
    - Store the time in `connection_time_sec`.
    - Increase the number of connections.
    - If this is the first connection, clear `disconnected_total_sec`; otherwise, set it to the time spent since the disconnection.
    - Trace the duration spent connecting.
    - Because it is possible to retrieve the parent MAC address, set the `parent_tag` from it. This is a convenient way to identify devices using a shorter string than their entire MAC or IPv6 address. It also has the advantage of being visible in both the MAC and IPv6 addresses.
    - If a UDP socket has been opened with the Border Router, send it a connection message with all interesting values. It looks like: `{"device":"8486", "chip":"xG25", "parent":"333a", "running":"0:00:00:23", "join_states_sec":[0,0,9,1,12]}`. This simple message allows:
      - Identifying which devices are connected, and which part they are built on.
      - Knowing which device is their parent.
        - It becomes possible to draw the network topology based on this.
      - Knowing how much time has been spent moving between join states.
        - In the above example, we see that:
          - No time has been spent to reach join states 1 and 2, due to Wi-SUN stack optimizations allowing reuse of credentials if they are still valid.
          - It took `9 + 1 + 12 = 22` seconds to reconnect. No time spent in join states 1 and 2 means it was a reconnection with previously saved credentials.

    - Store the time in `disconnection_time_sec`.
    - Trace the duration spent before disconnecting.
    - Add this to 'connected_total_sec'.
  - Set `previous_join_state` to the new join state.

Using the `app_join_state_sec[]` array, it becomes possible for the application to compute the values in the connection message.

Using the various timestamp values, it is possible to compute an availability ratio equal to:

`availability = 100.0*(connected_total_sec)/(connected_total_sec + disconnected_total_sec)`.

This value gives an interesting indication on the network availability, even if it is 100+ days long, which should be common in Wi-SUN.

JSON Connection Strings

Prepare JSON Connection Strings

Using the values set by the custom callback to follow the join state, you can easily print connection/disconnection information using the 1-second timestamping, and compute the network availability ratio.

> Note: This is all done in `app.c` .

Two messages are used:

Initial Connection Message

```
{
"device":"8486",
"chip":  "xG25",
"parent":"333a",
"running":    "  0:22:21:17",
"join_states_sec":[0,0,9,1,12]
}
```

This message is printed in the device console and sent to the Border Router right after connecting.

- "device" is the `device_tag` string, formatted from the MAC address of the device. This is a convenient way to identify devices using a shorter string than their entire MAC or IPv6 address. It has also the advantage of being visible in both the MAC and IPv6 addresses. This is fixed and set once only in `app.c/app_task()` .
- "chip" is the type of the part used, based on the following code (which can easily be extended to support more parts):

```
#ifdef  _SILICON_LABS_32B_SERIES_1
 #ifdef _SILICON_LABS_32B_SERIES_1_CONFIG_2
  #define CHIP "xG12"
 #endif
#endif

#ifdef  _SILICON_LABS_32B_SERIES_2
 #ifdef _SILICON_LABS_32B_SERIES_2_CONFIG_5
  #define CHIP "xG25"
 #endif
 #ifdef _SILICON_LABS_32B_SERIES_2_CONFIG_8
  #define CHIP "xG28"
 #endif
#endif

#ifndef CHIP
 #define CHIP SL_BOARD_NAME
#endif
```

- "parent" is the `parent_tag` string, formatted from the MAC address of the parent as soon as connection is achieved. This can change over time, when the device selects a different parent, which is normal in a Wi-SUN network depending on the reception conditions. It is set inside the custom callback following join state changes.

- The "device" and "parent" information allow building the network topology, to compare it to what `wsbrd_cli status` shows. The topology in this case is using information from the devices, not information retrieved using DBus from wsbrd. Both topology graphs should match.
- "running" is the amount of time elapsed since the timestamp has been initialized in `app.c/app_task()`, calling `app_timestamp_init()`. In the example, it shows a 22+ hour run. *Note that the very first value is a days count, required for long-term monitoring in a Wi-SUN network.*
- "join_state_sec" is an array of delays between join state changes, starting with join state 1. Five values are present, each corresponding to the time required to reach the corresponding join state. In the example, we can say that:
  - The device took 9 + 1 + 12 = 22 seconds to connect.
  - No time was spent going to join states 1 and 2, meaning that the device could reuse already existing credentials, thus saving time by skipping the authentication exchanges in steps 1 and 2. This is part of Wi-SUN stack optimizations for faster reconnection. If `sl_wisun_clear_credential_cache()` is called before connecting, authentication is required to retrieve credentials.

Status Message


```
{
"device":"8486",
"chip":  "xG25",
"parent":"333a",
"running":    "  0:22:21:17",
"connected":   "  0:22:20:35",
"disconnected":"no",
"connections": "1",
"availability":"100.00",
"connected_total":   "  0:22:20:35",
"disconnected_total":"  0:00:00:00"
}
```


This message is printed in the console and sent to the Border Router every `auto_send_sec` seconds.

By default, `auto_send_sec` is set to 60 seconds. This can be customized in the source code. It is also possible with the example application to change it remotely using a dedicated CoAP message. Obviously, if the device is not connected, the message is only printed in the console.

- "device", "chip", and "running" are exactly the same as in the initial connection message.
- "connected" is the duration of the current connection. This is reset when reconnecting, such that it monitors only the last connection last. It is "no" when not connected.
- "disconnected" is the duration since the last disconnection. It is "no" when connected.
- "availability" is the ratio between the connected time and the total time since the first connection. It is computed as `100.0*(connected_total_sec)/(connected_total_sec + . disconnected_total_sec)`.
- "connected_total" is the sum of the connected times since the very first connection, used to compute the availability ratio.
- "disconnected_total" is the sum of the disconnected times since the very first connection, used to compute the availability ratio. When "disconnected_total" is `"  0:00:00:00"`, the availability rate is 100%.

> Note: To get a fair value of the availability ratio, disconnection and connection times accumulation only starts on the very first connection. Otherwise, the statistics would be impacted when the Border Router is started after the devices. With this configuration, the availability ratio should be very close to 100% under normal conditions, with devices quickly reconnecting following a disconnection.

If the Border Router is ever stopped and restarted, the devices will reconnect and send their status messages again, all showing a "disconnected" duration in the same range. This is a clear indication that the disconnection was due to the Border Router, and debugging should be oriented towards the Border Router.

When a single device has issues, its next status messages will show a "disconnected" value different from all other nodes. In this situation, debugging should be oriented at the device itself.

If a device is disconnected, printing the messages into its console (and RTT traces) still allows debugging up to a certain level, as long as access to the console or to RTT traces is possible (only in a lab environment in this case):

- If the messages don't appear in the console, the application is probably stalled and debugging is required, starting with RTT traces (which keep a bit of history on the last traces).
- If the messages still appear in the console, the application is still running. Further debugging may be possible looking at RTT traces for any error message coming from the Wi-SUN stack.

Code Walkthrough

Three macros are used to format the first part of both messages and to add JSON start and end strings:

```
// JSON common format strings
#define START_JSON "{\n"

#define END_JSON   "}\n"

#define DEVICE_PARENT_RUNNING_JSON \
  "\"device\":\"%s\",\n"          \
  "\"chip\":  \"%s\",\n"          \
  "\"parent\":\"%s\",\n"          \
  "\"running\":    \"%s\",\n"
```

The initial connection message is formatted in `_connection_json_string()`.

```
char* _connection_json_string () {
  #define CONNECTION_JSON_FORMAT_STR                \
    START_JSON                                      \
    DEVICE_PARENT_RUNNING_JSON                      \
    "\"join_states_sec\":[%llu,%llu,%llu,%llu]\n" \
    END_JSON

  char sec_string[20];
  sprintf(sec_string, "%s", now_str());
  snprintf(json_string, SL_WISUN_COAP_RESOURCE_HND_SOCK_BUFF_SIZE,
    CONNECTION_JSON_FORMAT_STR,
    device_tag,
    CHIP,
    parent_tag,
    sec_string,
    app_join_state_delay_sec[1],
    app_join_state_delay_sec[2],
    app_join_state_delay_sec[3],
    app_join_state_delay_sec[4],
    app_join_state_delay_sec[5]
  );
  return json_string;
};
```

The status message is formatted in `_status_json_string()`.

```
char* _status_json_string (char * start_text) {
 #define CONNECTED_JSON_FORMAT_STR       \
  "%s"                              \
  START_JSON                        \
  DEVICE_PARENT_RUNNING_JSON        \
  "\"connected\":   \"%s\",\n"      \
  "\"disconnected\":\"%s\",\n"      \
  "\"connections\": \"%d\",\n"      \
  "\"availability\":\"%6.2f\",\n"   \
  "\"connected_total\":   \"%s\",\n"  \
  "\"disconnected_total\":\"%s\"\n"   \
  END_JSON

 char running_sec_string[20];
 char current_sec_string[20];
 char connected_sec_string[20];
 char disconnected_sec_string[20];

 uint64_t current_state_sec;

 sprintf(running_sec_string, "%s", now_str());

 if (join_state == SL_WISUN_JOIN_STATE_OPERATIONAL) {
  current_state_sec = now_sec() - connection_time_sec;
  sprintf(current_sec_string,    "%s", dhms(current_state_sec));
  sprintf(connected_sec_string,  "%s", dhms(connected_total_sec + current_state_sec));
  sprintf(disconnected_sec_string,"%s", dhms(disconnected_total_sec));
  snprintf(json_string, SL_WISUN_COAP_RESOURCE_HND_SOCK_BUFF_SIZE,
    CONNECTED_JSON_FORMAT_STR,
    start_text,
    device_tag,
    CHIP,
    parent_tag,
    running_sec_string,
    current_sec_string,
    "no",
    connection_count,
    100.0*(connected_total_sec + current_state_sec)/(connected_total_sec + current_state_sec + disconnected_total_sec),
    connected_sec_string,
    disconnected_sec_string
  );
 } else {
  current_state_sec = now_sec() - disconnection_time_sec;
  sprintf(current_sec_string,    "%s", dhms(current_state_sec));
  sprintf(connected_sec_string,  "%s", dhms(connected_total_sec));
  sprintf(disconnected_sec_string,"%s", dhms(disconnected_total_sec + current_state_sec));
  snprintf(json_string, SL_WISUN_COAP_RESOURCE_HND_SOCK_BUFF_SIZE,
    CONNECTED_JSON_FORMAT_STR,
    start_text,
    device_tag,
    CHIP,
    parent_tag,
    running_sec_string,
    "no",
    current_sec_string,
    connection_count,
    100.0*(connected_total_sec)/(connected_total_sec + disconnected_total_sec + current_state_sec),
    connected_sec_string,
    disconnected_sec_string
  );
 }

 return json_string;
}
```

Send Status Strings

Send Status Strings to the Border Router on UDP Port 1237

Status strings are prepared as in JSON Connection Strings, getting ready to be sent to the Border Router or any other IPv6 address. To send them to their destination, the following steps are needed.

> Note: This is all done in `app.c`, which you can use as a replacement for the original code.

Set the Destination IPv6

IPv6 Declaration

What you want is to automatically select the Border Router IPv6, which the device will only know once it is connected, or set a static IPv6 if your system is set with a different destination for network monitoring messages.

- A text string is defined in `app.c`. It can be either `"border_router"` or a valid IPv6.

```
#define UDP_NOTIFICATION_DEST  "border_router" // "border_router" or fixed IPv6 string
```

- Once connected, a part of `app.c/_select_destinations()` takes care of setting the intended IPv6.

```
// Set the UDP notification destination (Border Router by default)
printfBothTime("UDP_NOTIFICATION_DEST: %s\n", UDP_NOTIFICATION_DEST);
if (sl_strcasecmp(UDP_NOTIFICATION_DEST, "border_router") == 0) {
  memcpy(udp_notification_ipv6.address,   border_router_ipv6.address, sizeof(device_global_ipv6.address));
} else {
  sl_wisun_stoip6(UDP_NOTIFICATION_DEST    , SL_WISUN_IP_ADDRESS_SIZE, udp_notification_ipv6.address);
}
sl_wisun_ip6tos(udp_notification_ipv6.address      , udp_notification_ipv6_string);
printfBothTime("UDP  Notification destination: %s\n", udp_notification_ipv6_string);
```

Convenience functions `sl_wisun_stoip6()` and `sl_wisun_ip6tos()` are used here to convert between IPv6s and strings. Both are defined in `sl_wisun_ipv6string.h`.

Set the Destination UDP Port

The destination port is hardcoded in `app.c` as `UDP_NOTIFICATION_PORT`. It can be customized here:

```
#define UDP_NOTIFICATION_PORT  1237
```

Open and Connect the UDP Socket

Each socket id is stored in a `sl_wisun_socket_id_t`, so `udp_notification_socket_id` is declared in `app.c`:

```
// Notification sockets
sl_wisun_socket_id_t udp_notification_socket_id = 0;
```

Once the device is connected to the Wi-SUN network, it becomes possible to open the UDP socket and connect it to the desired destination and port, as in `app.c/_open_udp_sockets_with_Border_Router()` :

```
// UDP Notifications (autonomously sent by the device)
ret = sl_wisun_open_socket (SL_WISUN_SOCKET_PROTOCOL_UDP, &udp_notification_socket_id);
NO_ERROR(ret, "Opened    the UDP  notification socket (%ld)\n", udp_notification_socket_id);
IF_ERROR_RETURN(ret, "[Failed: unable to open the UDP notification socket: 0x%04x]\n", (uint16_t)ret);

ret = sl_wisun_connect_socket(udp_notification_socket_id, &udp_notification_ipv6, UDP_NOTIFICATION_PORT);
NO_ERROR(ret, "Connected the UDP  notification socket (%ld %s/%d)]\n",
      udp_notification_socket_id, udp_notification_ipv6_string, UDP_NOTIFICATION_PORT);
IF_ERROR_RETURN(ret, "[Failed: unable to connect the UDP notification socket: 0x%04x]\n", (uint16_t)ret);
```

Print and Send Status Messages

Auto Send Loop

In `app.c/app_task()` , the code enters an endless loop after initial connection, with a pause of `auto_send_sec` seconds after each loop. Inside this loop, a call to `_print_and_send_messages()` is used to send the string formatted by `_status_json_string()` :

```
void app_task(void *args)
{
...
 bool with_time, to_console, to_rtt, to_udp, to_coap;
...
 with_time = to_console = to_rtt = true;
 to_udp = to_coap = false;
...
 while (1) {
  ////////////////////////////////////////////////////////////////////////////
  // Put your application code here!                                        //
  ////////////////////////////////////////////////////////////////////////////

  // We can only send messages outside if connected
  if (join_state == SL_WISUN_JOIN_STATE_OPERATIONAL) {
   to_udp = to_coap = true;
  } else {
   to_udp = to_coap = false;
  }

  // Print status message every auto_send_sec seconds
  _print_and_send_messages (_status_json_string(""),
         with_time, to_console, to_rtt, to_udp, to_coap);
  osDelay(auto_send_sec*1000);

 }
}
```

Messages Print Out and Transmission

In `app.c/_print_and_send_messages()` , the flags are used to select where to send the message, including to the selected UDP port:

```
uint8_t _print_and_send_messages (char *in_msg, bool with_time,
                        bool to_console, bool to_rtt, bool to_udp, bool to_coap) {
  sl_status_t ret = SL_STATUS_OK;
  uint8_t messages_processed = 0;
  uint16_t udp_msg_len;
  uint16_t coap_msg_len;

  if (to_console == true) { // Print to console
     if (with_time == true) {
       printfTime(in_msg);
     } else {
       printf(in_msg);
     }
    messages_processed++;
  }
  if (to_rtt == true) {    // Print to RTT traces
     if (with_time == true) {
       printfTimeRTT(in_msg);
     } else {
       printfRTT(in_msg);
     }
    messages_processed++;
  }
  if (to_udp == true) {    // Send to UDP port
    udp_msg_len  = snprintf(udp_msg,  1024, "%s", in_msg);
    ret = sl_wisun_send_on_socket(udp_notification_socket_id, udp_msg_len, (uint8_t *)udp_msg);
    IF_ERROR(ret, "[Failed (line %d): unable to send to the UDP notification socket (%ld %s/%d): 0x%04x. Check sl_status.h] udp_msg_len %d\n",
          __LINE__, udp_notification_socket_id, udp_notification_ipv6_string , UDP_NOTIFICATION_PORT, (uint16_t)ret, udp_msg_len);
    if (ret == SL_STATUS_OK) messages_processed++;
  }
...
  return messages_processed;
}
```

Add CoAP Resources

Add CoAP Resources to the Device Application

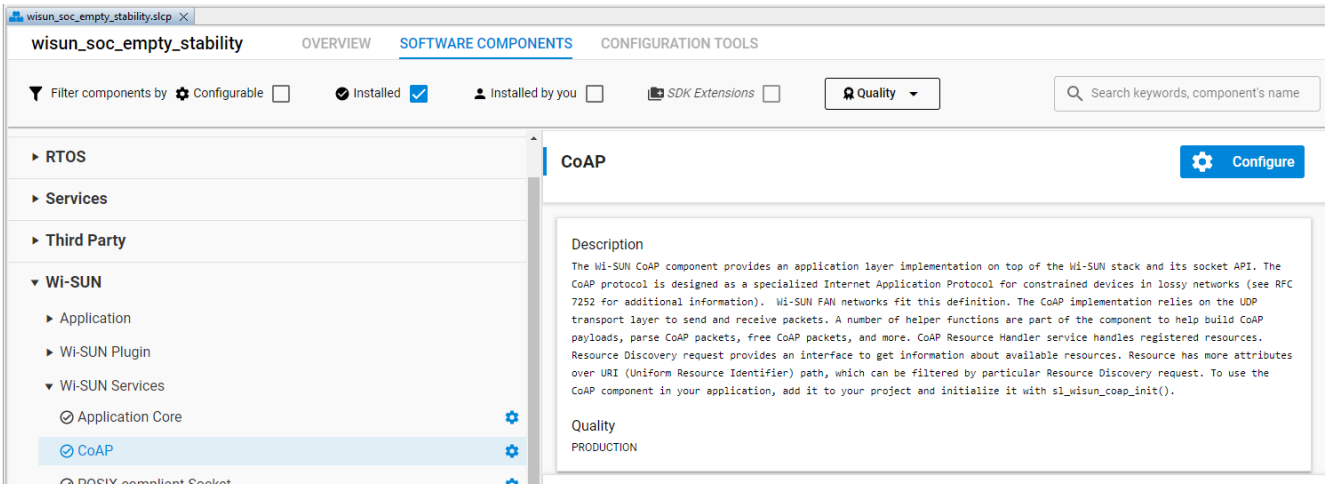
Using the Send status / Retrieve UDP notifications technique, the devices are automatically sending messages to the Border Router (by default, another destination is possible). In a large Wi-SUN network, this may not be convenient, or the `auto_send_sec` value would be set to a very high value, such that these messages act as 'keep alive' messages, used to make sure the devices are still active and connected.

An approach more fitting for large networks consists of adding the capability for the nodes to reply with the status messages to CoAP 'get' messages. With this approach, the network management machine is responsible for regularly checking the node status, potentially infrequently. This is described in the next paragraphs.

If the devices are monitoring critical sensors, such as temperature values, they can be set to send notification messages if the monitored values pass above pre-determined thresholds, or enter critical zones. This is application-specific, out of scope of the current example application.

Add CoAP to the Project

The demonstration code is in `app_coap.c` and `app_coap.h`. It uses the **Wi-SUN CoAP** service, available in **SOFTWARE COMPONENTS > Wi-SUN > Wi-SUN Services > CoAP**.



Because the demonstration application uses a lot of CoAP resources, the `Maximum capacity of the CoAP Resource Table` is set to `30` in `/wisun_soc_empty_stability/config/sl_wisun_coap_config.h`, instead of the default of `10`.

```
// <o SL_WISUN_COAP_RESOURCE_HND_MAX_RESOURCES> Maximum capacity of the CoAP Resource Table
// <i> Default: 10
#define SL_WISUN_COAP_RESOURCE_HND_MAX_RESOURCES      30U
```

The `Resource handler service socket communication buffer size` is left to its default value of `1024`, declared as `SL_WISUN_COAP_RESOURCE_HND_SOCK_BUFF_SIZE` in `sl_wisun_coap_config.h`. This will limit the length of the JSON text string we can send.

```
// <o SL_WISUN_COAP_RESOURCE_HND_SOCK_BUFF_SIZE> Resource handler service socket communication buffer size
// <i> Default: 1024
#define SL_WISUN_COAP_RESOURCE_HND_SOCK_BUFF_SIZE      1024UL
```

Definition of CoAP Resources

One of the most difficult parts of coding a CoAP application is setting correct names for resources, i.e. names that feel logical for most users, and present a consistent hierarchy for those resources.

In the demonstration application, Silicon Labs used:

- 'info/' for parameters that will not change for a given device:
  - device_tag
  - chip
  - board
  - 'info/all' returns a json string with all of the above
- 'status/' for parameters varying with time and not part of the statistics
  - running
  - parent (which can change if selecting a different parent)
  - neighbor (access to neighbor count if no payload, otherwise adding `-e <n>` returns info for neighbor `n`)
  - connected (not accumulated connection time)
  - 'status/all' returns a json string with all of the above
- 'statistics/' for values accumulated by the application, or retrieved from the stack
  - 'statistic/app/'
    - join_states_sec
    - disconnected_total
    - connections
    - connected_total
    - availability
    - 'statistics/app/all' returns a json string with all of the above. Adding `-e reset` resets these statistics.
  - 'statistics/stack/'
    - phy (statistics from sl-wisun-statistics-phy-t. Adding `-e reset` resets these statistics)
    - mac (statistics from sl-wisun-statistics-mac-t. Adding `-e reset` resets these statistics)
    - fhss (statistics from sl-wisun-statistics-fhss-t. Adding `-e reset` resets these statistics)
    - wisun (statistics from sl-wisun-statistics-wisun-t. Adding `-e reset` resets these statistics)
    - network (statistics from sl-wisun-statistics-network-t. Adding `-e reset` resets these statistics)
    - regulation (statistics from sl-wisun-statistics-regulation-t. Adding `-e reset` resets these statistics)
    - No 'all' option here, because some of the resources already return very long strings. Adding all together would overflow the json string limit.
- 'settings/' for parameters of the application we want to change via CoAP
  - auto_send (returns the current `auto_send_sec` value if no payload, otherwise adding `-e <s>` sets `auto_send_sec` to `s`))

Declare a CoAP Callback Function for each CoAP Resource

The application needs to know which function to execute when a CoAP `get` request is received. There is **one function per resource**. We will only document some of them. Adding more is straightforward once the first ones are clearly explained.

CoAP Callback Prototype

Each CoAP callback function has the same return type and parameters:

- return type `sl_wisun_coap_packet_t *`, a pointer to the response packet
- parameter `const sl_wisun_coap_packet_t *const req_packet`, the CoAP request packet, defined in `sl_wisun_coap.h` as:

```
/**
 * \brief Main CoAP message struct
 */
typedef struct sn_coap_hdr_ {
  uint8_t              token_len;          /**< 1-8 bytes. */

  sn_coap_status_e     coap_status;        /**< Used for telling to User special cases when parsing message */
  sn_coap_msg_code_e   msg_code;           /**< Empty: 0; Requests: 1-31; Responses: 64-191 */

  sn_coap_msg_type_e   msg_type;           /**< Confirmable, Non-Confirmable, Acknowledgment or Reset */
  sn_coap_content_format_e content_format; /**< Set to COAP_CT_NONE if not used */

  uint16_t             msg_id;             /**< Message ID. Parser sets parsed message ID, builder sets message ID of built coap message */
  uint16_t             uri_path_len;       /**< 0-255 bytes. Repeatable. */
  uint16_t             payload_len;        /**< Must be set to zero if not used */

  uint8_t             *token_ptr;          /**< Must be set to NULL if not used */
  uint8_t             *uri_path_ptr;       /**< Must be set to NULL if not used. E.g: temp1/temp2 */
  uint8_t             *payload_ptr;        /**< Must be set to NULL if not used */

  /* Here are not so often used Options */
  sn_coap_options_list_s *options_list_ptr;    /**< Must be set to NULL if not used */
} sn_coap_hdr_s;
...
typedef sn_coap_hdr_s sl_wisun_coap_packet_t;
```

The above shows that there is a possible payload, corresponding to `payload_ptr` with a size of `payload_len`. The code can therefore check the payload to act differently. This payload corresponds to the `-e <payload>` the user can add to each CoAP request.

The `settings` resource is interesting to look at, since it can be used to **retrieve** the `auto_send_sec` value and also to **change** it.

CoAP Reply Function

To more easily reply to a CoAP request with a text string as the result, the `app_coap_reply()` function is added to `app_coap.c` :

```
sl_wisun_coap_packet_t * app_coap_reply(char *response_string,
                const sl_wisun_coap_packet_t *const req_packet) {

  sl_wisun_coap_packet_t* resp_packet = NULL;
  // Prepare CoAP response packet with default response string
  resp_packet = sl_wisun_coap_build_response(req_packet, COAP_MSG_CODE_RESPONSE_BAD_REQUEST);
  if (resp_packet == NULL) {
    return NULL;
  }

  resp_packet->msg_code      = COAP_MSG_CODE_RESPONSE_CONTENT;
  resp_packet->content_format = COAP_CT_TEXT_PLAIN;
  resp_packet->payload_ptr    = (uint8_t *)response_string;
  resp_packet->payload_len    = (uint16_t)sl_strnlen(response_string, COAP_MAX_RESPONSE_LEN);

  return resp_packet;
}
```

The CoAP component provides the `sl_wisun_coap_build_response()` function, which can be used to prepare most of the reply. Then the payload is set to the `response_string` input string.

CoAP Resource Handler

The CoAP Resource Handler is the only part of the CoAP component you need to interact with. It will then perform the following tasks whenever a CoAP request is received:

- Check that there is a known resource `uri_path` matching the request's `uri_path_ptr`.
  - If yes, call the resource's `auto_response` function.
  - If the `auto_response` function is coded to send a reply, the reply will be sent to the IPv6 address at the origin of the request.
- If the request is `".well-known/core"`, return a JSON array with all resource set as `discoverable`

`settings/auto_send` Example

The `settings/auto_send` resource is interesting to look at, since it can be used to **retrieve** the `auto_send_sec` value and also to **change** it.

`settings/auto_send` CoAP Callback Function

```
sl_wisun_coap_packet_t * coap_callback_auto_send (
    const  sl_wisun_coap_packet_t *const req_packet)  {
  int sec = 0;
  int res;
  if (req_packet->payload_len) {
    res = sscanf((char *)req_packet->payload_ptr, "%d", &sec);
    if (res) { auto_send_sec = sec; }
  }
  snprintf(coap_response, COAP_MAX_RESPONSE_LEN, "auto_send_sec: %d", auto_send_sec);
return app_coap_reply(coap_response, req_packet); }
```

The one-before-last line of the function is formatting the text string corresponding to the reply's payload (`req_packet->payload_ptr`), then `app_coap_reply()` is called to send the reply to the sender. The reply is in the form `auto_send_sec: 60` .

Before this, the request payload has been checked, and if it contains a decimal value, this is used as the new `auto_send_sec` if it exists, so the reply will always contain the current value.

`settings/auto_send` CoAP Resource Registration

To have the CoAP component handle this new resource, a part of the `app_coap_resources_init()` function provides a corresponding `sl_wisun_coap_rhnd_resource_t` to the CoAP Resource Handler:

```
uint8_t app_coap_resources_init() {
  sl_wisun_coap_rhnd_resource_t coap_resource = { 0 };
  uint8_t count = 0;

  // Add CoAP resources (one per item)
...
  coap_resource.data.uri_path = "/settings/auto_send";
  coap_resource.data.resource_type = "sec";
  coap_resource.data.interface = "settings";
  coap_resource.auto_response = coap_callback_auto_send;
  coap_resource.discoverable = true;
  assert(sl_wisun_coap_rhnd_resource_add(&coap_resource) == SL_STATUS_OK);
    count++;
...
  printf("  %d/%d CoAP resources added to CoAP Resource handler\n", count, SL_WISUN_COAP_RESOURCE_HND_MAX_RESOURCES);
  return count;
}
```

A check has been added at the end of `app_coap_resources_init()` on the number of CoAP resources registered with the CoAP Resource Handler. This allows you to check if `SL_WISUN_COAP_RESOURCE_HND_MAX_RESOURCES` is correctly set, and if it needs to be increased.

CoAP Resources Initialization by the Application

The `app_coap_resources_init()` function is called once at startup from `app-init.c/app_init()`, adding all CoAP resources to the CoAP Resource Handler:

```
void app_init(void)
{
 app_coap_resources_init();

 /* Creating App main thread */
...
```

With the above, from the Border Router console, you can:

- Discover the available CoAP resources on the device, using

`coap-client -m get -N -B 3 coap://[<device_IPv6>]:5683/.well-known/core`

- Retrieve a CoAP resource using

`coap-client -m get -N -B 3 coap://[<device_IPv6>]:5683/<resource>`

Details about the possibilities are provided in the `app_coap.c/print_coap_help()` function once connected, because at this moment the device has obtained an IPv6 address:

```
void  print_coap_help (char* device_global_ipv6_string, char* border_router_ipv6_string) {
 printf("\n");
 printf("To start a CoAP server on the linux Border Router:\n");
 printf("  coap-server -A %s -p %d -d 10\n", border_router_ipv6_string, 5685);
 printf("CoAP discovery:\n");
 printf("  coap-client -m get -N -B 3 coap://[%s]:5683/.well-known/core\n", device_global_ipv6_string);
 printf("CoAP GET requests:\n");
 printf("  coap-client -m get -N -B 3 coap://[%s]:5683/<resource>, for the following resources:\n", device_global_ipv6_string);
 sl_wisun_coap_rhnd_print_resources();
 printf("  '/settings/auto_send'        returns the current notification duration in seconds\n");
 printf("  '/settings/auto_send' -e <d>' changes the notification duration to d seconds\n");
 printf("  '/status/neighbor'           returns the neighbor_count\n");
 printf("  '/status/neighbor -e <n>'    returns the neighbor information for neighbor at index n\n");
 printf("  '/statistics/stack/<group> -e reset' clears the Stack statistics for the selected group\n");
 printf("  '/statistics/app/all      -e reset' clears all statistics\n");
 printf("\n");
}
```

`status/all` Example

The `status/all` resource is used to retrieve the `running`, `connected`, and `parent` information.

`status/all` CoAP Callback Function

```
sl_wisun_coap_packet_t * coap_callback_all_statuses (
    const  sl_wisun_coap_packet_t *const req_packet)  {
 #define JSON_ALL_STATUSES_FORMAT_STR     \
  "{\n"                           \
  "  \"running\": \"%s\",\n"          \
  "  \"connected\": \"%s\"\n"         \
  "  \"parent\": \"%s\",\n"           \
  "}\n"

 char running_str[40];
 char connected_str[40];

 snprintf(running_str  , 40, dhms(now_sec()) );
 snprintf(connected_str, 40, dhms(now_sec() - connection_time_sec) );

 snprintf(coap_response, COAP_MAX_RESPONSE_LEN, JSON_ALL_STATUSES_FORMAT_STR,
         running_str,
         connected_str,
         parent_tag
 );
 return app_coap_reply(coap_response, req_packet);
}
```

`status/all` CoAP Resource Registration

To have the CoAP component handle this new resource, a part of the `app_coap_resources_init()` function provides a corresponding `sl_wisun_coap_rhnd_resource_t` to the CoAP Resource Handler:

```
uint8_t app_coap_resources_init() {
 sl_wisun_coap_rhnd_resource_t coap_resource = { 0 };
 uint8_t count = 0;

 // Add CoAP resources (one per item)
...
 coap_resource.data.uri_path = "/status/all";
 coap_resource.data.resource_type = "json";
 coap_resource.data.interface = "node";
 coap_resource.auto_response = coap_callback_all_statuses;
 coap_resource.discoverable = true;
 assert(sl_wisun_coap_rhnd_resource_add(&coap_resource) == SL_STATUS_OK);
 count++;
...
 printf("  %d/%d CoAP resources added to CoAP Resource handler\n", count, SL_WISUN_COAP_RESOURCE_HND_MAX_RESOURCES);
 return count;
}
```

As visible above, there are very few differences between the block of code used to register `"/settings/auto_send"` or `"/status/all"`, so adding more is easy.

Before looking at adding CoAP to the Border Router, below is an example of the output you can get on the Border Router when monitoring multiple devices:

`status/all` in use

```
./coap_all /status/all
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a51c]:5683/status/all : { "running": " 8-05:37:43", "connected": " 0:01:49:55"
"parent": "2527", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a51d]:5683/status/all : { "running": " 0-23:00:55", "connected": " 0:22:59:53"
"parent": "a901", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a527]:5683/status/all : { "running": " 8-05:37:35", "connected": " 0:02:05:35"
"parent": "333a", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a8ff]:5683/status/all : { "running": " 8-05:37:38", "connected": " 0:21:05:04"
"parent": "333a", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a901]:5683/status/all : { "running": " 1-02:44:11", "connected": " 1:02:43:16"
"parent": "333a", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2191]:5683/status/all : { "running": " 1-05:50:51", "connected": " 1:05:49:47"
"parent": "333a", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2527]:5683/status/all : { "running": " 1-05:50:50", "connected": " 1:05:49:37"
"parent": "a8ff", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2853]:5683/status/all : { "running": " 1-05:50:53", "connected": " 0:05:10:54"
"parent": "333a", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8486]:5683/status/all : { "running": " 1-05:43:48", "connected": " 0:00:47:53"
"parent": "333a", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8493]:5683/status/all : { "running": " 1-02:45:43", "connected": " 0:10:37:17"
"parent": "2853", }
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8503]:5683/status/all : { "running": " 1-02:43:29", "connected": " 0:07:51:09"
"parent": "333a", }
```

> NOTE: Remember that timestamping is in `dd-hh:mm:ss` format, meaning that in the example above `" 8-05:37:43` means more than 8 days of connection time, corresponding to a group of 3 devices, while others have been started only a day ago, with about 3 hours between them. This shows how interesting it is to use this time stamping format.

Retrieve UDP Notifications

Retrieve UDP Notifications on the Border Router

Note that **this part is executed on the Linux Border Router**.

If the application flashed on the Wi-SUN devices is sending notification messages to the Border Router, a convenient way to retrieve them from the Border Router is via a python script such as `udp_notification_receiver.py`.

Call the Script

```
python udp_notification_receiver.py 1237 " "
```

- The first parameter is the UDP port to listen to, `rcv_port` in the script.
- The second parameter is an optional `newline` parameter used to replace the newline characters from the incoming message. It's useful to display one message per line, since otherwise they would be displayed in json format on multiple lines. You can use a different character as a separator.
- The script is also suppressing all spaces from the incoming message, for a more compact display.

Script Output

```
python udp_notification_receiver.py 1237 " "
Receiving on ::/1237...
[2023-08-18 14:07:28] Rx 1237:  { "device":"a901", "chip":"xG25", "parent":"333a", "running":"0:22:47:30", "connected":"0:22:46:35",
"disconnected":"no", "connections":"1", "availability":"100.00", "connected_total":"0:22:46:35", "disconnected_total":"0:00:00:00" }
[2023-08-18 14:07:44] Rx 1237:  { "device":"8486", "chip":"xG25", "parent":"333a", "running":"1:01:47:22", "connected":"1:01:46:40",
"disconnected":"no", "connections":"1", "availability":"100.00", "connected_total":"1:01:46:40", "disconnected_total":"0:00:00:00" }
[2023-08-18 14:07:46] Rx 1237:  { "device":"a51d", "chip":"xG25", "parent":"a901", "running":"0:19:04:32", "connected":"0:19:03:30",
"disconnected":"no", "connections":"1", "availability":"100.00", "connected_total":"0:19:03:30", "disconnected_total":"0:00:00:00" }
[2023-08-18 14:07:58] Rx 1237:  { "device":"a51c", "chip":"xG25", "parent":"333a", "running":"8:01:41:33", "connected":"1:20:54:38",
"disconnected":"no", "connections":"6", "availability":"97.34", "connected_total":"7:20:32:26", "disconnected_total":"0:05:08:36" }
[2023-08-18 14:08:03] Rx 1237:  { "device":"2191", "chip":"xG25", "parent":"333a", "running":"1:01:54:45", "connected":"1:01:53:41",
"disconnected":"no", "connections":"1", "availability":"100.00", "connected_total":"1:01:53:41", "disconnected_total":"0:00:00:00" }
[2023-08-18 14:08:04] Rx 1237:  { "device":"2853", "chip":"xG25", "parent":"333a", "running":"1:01:54:47", "connected":"0:01:14:48",
"disconnected":"no", "connections":"2", "availability":"99.88", "connected_total":"1:01:51:49", "disconnected_total":"0:00:01:52" }
```

Code Walkthrough

The script is as follows:

```
# Used to receive UDP notifications strings
# Call with
# python udp_notification_receiver.py 1237 " "

import socket
import sys
import datetime

HOST_IP = "::" # Host own address (tun0 IPv6 address)

rcv_port = int(sys.argv[1])
newline = " "

if (len(sys.argv) > 2):
  newline = sys.argv[2]
  space = ""

PORT = rcv_port # Port used by the peer

sock = socket.socket(socket.AF_INET6, socket.SOCK_DGRAM)
sock.bind((HOST_IP, PORT))

print(f"Receiving on {HOST_IP}/{PORT}...")

while True:
  data, addr = sock.recvfrom(2048) # buffer size is 2048 bytes
  now = datetime.datetime.now()
  now_str = str(now.strftime('%Y-%m-%d %H:%M:%S'))

  try:
    message_string = data.decode("utf-8").replace(" ", space).replace("\n", newline)
  except Exception as e:
    print(f"Exception {e} (from {addr})")

  print (f"[{now_str}] Rx {PORT}: {newline}", message_string)
```

It goes through:

- Import of the required python packages
- Retrieving the `rcv_port` value from the first argument
- Retrieving the `newline` replacement, if any, from the second argument
- Opening a socket to listen to the `rcv_port` on all IPv6 addresses
- Infinite loop with:
  - Reception from the UDP socket
  - Formatting the current date and time
  - Removing all spaces from the received message
  - Replacing all `\n` from the received message by `newline`
  - Printing the modified string

Retrieve Device Information

Retrieve the Device Information from the Border Router using CoAP

Install `libcoap2-bin` on the Border Router

- The application requires libcoap binaries to be installed on the Border Router, so install the `-bin` package.
- If using CoAP notifications instead of UDP notifications, `coap-server` will be used, and requires the `-d` option to define the max number of resources the coap-server can listen to. This is only available in `libcoap` as from `libcoap2`, so install `libcoap` as follows:

`sudo apt-get install libcoap2-bin`

Check the installed binaries using:

`ls -al /usr/bin/coap*`

```
-rwxr-xr-x 1 root root 26236 Nov  9  2019 /usr/bin/coap-client
-rwxr-xr-x 1 root root 26236 Nov  9  2019 /usr/bin/coap-client-gnutls
-rwxr-xr-x 1 root root 26236 Nov  9  2019 /usr/bin/coap-client-openssl
-rwxr-xr-x 1 root root 18016 Nov  9  2019 /usr/bin/coap-rd
-rwxr-xr-x 1 root root 18016 Nov  9  2019 /usr/bin/coap-rd-gnutls
-rwxr-xr-x 1 root root 18016 Nov  9  2019 /usr/bin/coap-rd-openssl
-rwxr-xr-x 1 root root 22124 Nov  9  2019 /usr/bin/coap-server
-rwxr-xr-x 1 root root 22124 Nov  9  2019 /usr/bin/coap-server-gnutls
-rwxr-xr-x 1 root root 22124 Nov  9  2019 /usr/bin/coap-server-openssl
```

The example uses `coap-client` and optionally `coap-server` (if using COAP notifications).

List Connected Devices' IPv6 Addresses

`wsbrd_cli status` returns the connected devices' MAC addresses, while CoAP requires the IPv6 addresses.

The dbus resources related to the Wi-SUN Border Router are available to dbus as `com.silabs.Wisun.BorderRouter` :

```
busctl introspect  SERVICE                    OBJECT                      [INTERFACE]
busctl introspect  com.silabs.Wisun.BorderRouter  /com/silabs/Wisun/BorderRouter  com.silabs.Wisun.BorderRouter
```

```
NAME                            TYPE      SIGNATURE  RESULT/VALUE                    FLAGS
com.silabs.Wisun.BorderRouter   interface -          -                               -
.leCustomClear                  method    -          -                               -
.leCustomInsert                 method    yyayay     -                               -
.InstallGtk                     method    ay         -                               -
.InstallLgtk                    method    ay         -                               -
.JoinMulticastGroup             method    ay         -                               -
.LeaveMulticastGroup            method    ay         -                               -
.RevokeGroupKeys                method    ayay       -                               -
.RevokePairwiseKeys             method    ay         -                               -
.SetModeSwitch                  method    ayi        -                               -
.SetSlotAlgorithm               method    y          -                               -
.Gaks                           property  aay        4 16 54 182 30 129 250 196 136 180 3 24... emits-change
.Gtks                           property  aay        4 16 0 16 32 48 64 80 96 112 128 144 16... emits-change
.HwAddress                      property  ay         8 144 253 159 255 254 0 51 58           -
.Lgaks                          property  aay        3 16 159 29 31 16 211 121 13 120 93 104... emits-change
.Lgtks                          property  aay        3 16 205 198 155 180 84 17 14 130 116 6... emits-change
.Nodes                          property  a(aya{sv}) 3 8 144 253 159 255 254 0 51 58 3 "is_b... emits-invalidation"
.WisunChanPlanId                property  u          3                               const
.WisunClass                     property  u          0                               const
.WisunDomain                    property  s          "BZ"                            const
.WisunFanVersion                property  y          2                               const
.WisunMode                      property  u          0                               const
.WisunNetworkName               property  s          "Linux_BZ_3_8"                  const
.WisunPanId                     property  q          59418                           const
.WisunPhyModeId                 property  u          8                               const
.WisunSize                      property  s          "CERT"                          const
org.freedesktop.DBus.Introspectable interface -      -                               -
.Introspect                     method    -          s                               -
org.freedesktop.DBus.Peer       interface -          -                               -
.GetMachineId                   method    -          s                               -
.Ping                           method    -          -                               -
org.freedesktop.DBus.Properties interface -          -                               -
.Get                            method    ss         v                               -
.GetAll                         method    s          a{sv}                           -
.Set                            method    ssv        -                               -
.PropertiesChanged              signal    sa{sv}as   -                               -
```

What you are looking for is the `Nodes` property, which you can 'get' using:

```
busctl introspect com.silabs.Wisun.BorderRouter /com/silabs/Wisun/BorderRouter com.silabs.Wisun.BorderRouter Nodes
```

```
a(aya{sv}) 3 8 144 253 159 255 254 0 51 58 3 "is_border_router" b true "node_role" y 0 "ipv6" aay 2 16 254 128 0 0 0 0 0 0 146 253 159 255 254 0 51 58 16 253 0 97 114 109 0 0 0 146 253 159 255 254 0 51 58 8 96 164 35 255 254 55 168 255 8 "is_authenticated" b true "node_role" y 1 "parent" ay 8 144 253 159 255 254 0 51 58 "is_neighbor" b true "rssi" i -46 "rsl" i -41 "rsl_adv" i -37 "ipv6" aay 2 16 254 128 0 0 0 0 0 0 96 164 35 255 254 55 168 255 16 253 0 97 114 109 0 0 0 98 164 35 255 254 55 168 255 8 96 164 35 255 254 55 169 1 4 "is_authenticated" b true "node_role" y 1 "parent" ay 8 96 164 35 255 254 55 165 29 "ipv6" aay 2 16 254 128 0 0 0 0 0 0 96 164 35 255 254 55 169 1 16 253 0 97 114 109 0 0 0 98 164 35 255 254 55 169 1
```

The underlying structure is detailed in wsbrd-br-linux/DBUS.md:

- `ay`: EUI64
- `a{sv}`: list of properties identified by a string, as described in the
  following table. Not all properties are guaranteed to be present per node
  (ex: a node without parent has no `parent` field)

Key	Signature	Comment
`is_border_router`	`b`	Deprecated. Use `node_role` instead.
`node_role`	`y`	Semantics from Wi-SUN (`0`: BR, `1`: FFN-FAN1.1, `2`: LFN, none: FFN-FAN1.0)
`ipv6`	`aay`	Array of IPv6 addresses (usually link-local and GUA)
`parent`	`ay`	EUI-64 of the preferred parent
`is_authenticated`	`b`	
`is_neighbor`	`b`	Only nodes that use direct unicast traffic to the border router are listed
`rssi`	`i`	Received Signal Strength Indication (RSSI) of the last packet received in dBm (neighbor only)
`rsl`	`i`	Exponentially Weighted Moving Average (EWMA) of the Received Signal Level (RSL) in dBm (neighbor only)
`rsl_adv`	`i`	EWMA of the RSL in dBm advertised by the node in RSL-IE (neighbor only)

Although it's possible, using the above to retrieve IPv6 addresses is quite complex, and is better done using the `pydbus` Python package and a short script.

Install `pydbus` Python Package

```
pip3 install pydbus
```

`get_nodes_ipv6_address.py` Python Script

```
from pydbus import SystemBus
import re

"""
get_nodes_ipv6_address.py
Prints the IPv6 address of each node connected to the running WSBRD instance.

USAGE:
   python3 get_nodes_ipv6_address.py

NOTES:
   - This script can only be used with the WSBRD + RCP setup and must be executed on the host.
"""

bus = SystemBus()
proxy = bus.get("com.silabs.Wisun.BorderRouter", "/com/silabs/Wisun/BorderRouter")
nodes = proxy.Nodes


def sliceIPv6(source):
   return [source[i : i + 4] for i in range(0, len(source), 4)]


for node in nodes:
   if len(node[1]["ipv6"]) != 2:
      continue
   if "parent" not in node[1] and (
      "node_role" not in node[1] or node[1]["node_role"] != 2
   ):
      continue

   ipv6 = bytes(node[1]["ipv6"][1]).hex()
   ipv6 = ":".join(sliceIPv6(ipv6))
   ipv6 = re.sub("0000:", ":", ipv6)
   ipv6 = re.sub(":{2,}", "::", ipv6)
   print(ipv6)
```

For a more convenient use from bash, this Python script can be called from the `ipv6s` bash script.

```
#!/bin/bash
# usage: listing IPv6 addresses of all nodes currently connected
# ./ipv6s

ipv6s=$(python /home/pi/get_nodes_ipv6_address.py)
for ipv6 in $ipv6s
do
   echo $ipv6
done
```

This script needs to be set as an executable using `chmod a+x ipv6s` .

Use `ipv6s`

`./ipv6s`

```
fd00:6172:6d00::62a4:23ff:fe37:a8ff
fd00:6172:6d00::62a4:23ff:fe37:a901
```

> NOTE: The IPv6 addresses obtained from wsbrd via DBus reflect the vision of the network from wsbrd.

It takes a while for a device lifetime to expire, so even after a device is switched off, it may still appear as part of the network from wsbrd's perspective. If a device is off for a while and then turned on in a relatively short term, it may even reconnect before wsbrd considers it as disconnected. With the initial connection message in the demonstration application, you will know this because:

- You will receive a notification when the device will reconnect.
- You can ask via CoAP for the device status and statistics, which will tell you when the reconnection occurred.

Typical CoAP 'get' Request

`coap-client` uses port `5683` by default. This is the default CoAP UDP port, and is set in the application code.

An abstract of `coap-client --help` with the options used in the example is:

```
coap-client v4.2.1 -- a small CoAP implementation
Copyright (C) 2010-2019 Olaf Bergmann <bergmann@tzi.org> and others

TLS Library: None

Usage: coap-client [-e text] [-m method] [-N]  [-B seconds] URI

      URI can be an absolute URI or a URI prefixed with scheme and host

General Options
      -e text         Include text as payload (use percent-encoding for non-ASCII characters)
      -m method       Request method (get|put|post|delete|fetch|patch|ipatch), default is 'get'
      -N              Send NON-confirmable message
      -B seconds      Break operation after waiting given seconds (default is 90)
```

The typical `coap-client` request for the example application takes the following form:

`coap-client -m get -N -B 3 coap://[<IPV6>]:5683<COAP_URI> [-e <PAYLOAD>]`

where `COAP_URI` is the CoAP resource, starting with `/`

Discover CoAP Resources

When a device supports CoAP, it generally supports discovery using the `/.well-known/core` URI:

```
coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a901]:5683/.well-known/core
```

```
</settings>;ct=40,</statistics>;ct=40,</status>;ct=40,</info>;ct=40,</settings/auto_send>;rt="sec";if="settings",
</statistics/stack/regulation>;rt="json";if="regulation",</statistics/stack/network>;rt="json";if="network",
</statistics/stack/wisun>;rt="json";if="wisun",</statistics/stack/fhss>;rt="json";if="fhss",</statistics/stack/mac>;rt="json";if="mac",
</statistics/stack/phy>;rt="json";if="phy",</history>;rt="text";if="node",</statistics/app/availability>;rt="ratio";if="node",
</statistics/app/connected_total>;rt="dhms";if="node",</statistics/app/connections>;rt="int";if="node",
</statistics/app/disconnected_total>;rt="dhms";if="node",</statistics/app/join_states_sec>;rt="array";if="node",
</statistics/app/all>;rt="json";if="node",</status/connected>;rt="dhms";if="node",</status/neighbor>;rt="json";if="node",
</status/parent>;rt="tag";if="node",</status/running>;rt="dhms";if="node",</status/all>;rt="json";if="node",</info/version>;rt="text";if="node",
</info/application>;rt="text";if="node",</info/chip>;rt="tag";if="node",</info/device>;rt="tag";if="node",</info/all>;rt="json";if="node"
```

The above shows four groups of resources (splitting on `,`, with `ct=40`):

```
</settings>;ct=40,
</statistics>;ct=40,
</status>;ct=40,
</info>;ct=40,
```

And the resources you can access for each group (splitting the remaining text on `,`):

```
</settings/auto_send>;rt="sec";if="settings",
</statistics/stack/regulation>;rt="json";if="regulation",
</statistics/stack/network>;rt="json";if="network",
</statistics/stack/wisun>;rt="json";if="wisun",
</statistics/stack/fhss>;rt="json";if="fhss",
</statistics/stack/mac>;rt="json";if="mac",
</statistics/stack/phy>;rt="json";if="phy",
</history>;rt="text";if="node",
</statistics/app/availability>;rt="ratio";if="node",
</statistics/app/connected_total>;rt="dhms";if="node",
</statistics/app/connections>;rt="int";if="node",
</statistics/app/disconnected_total>;rt="dhms";if="node",
</statistics/app/join_states_sec>;rt="array";if="node",
</statistics/app/all>;rt="json";if="node",
</status/connected>;rt="dhms";if="node",
</status/neighbor>;rt="json";if="node",
</status/parent>;rt="tag";if="node",
</status/running>;rt="dhms";if="node",
</status/all>;rt="json";if="node",
</info/version>;rt="text";if="node",
</info/application>;rt="text";if="node",
</info/chip>;rt="tag";if="node",
</info/device>;rt="tag";if="node",
</info/all>;rt="json";if="node"
```

These match what the application provides, since it's filled by the CoAP Resource Manager based on registered CoAP resources.

Send a CoAP Request to All Connected Devices

The `coap_all` bash script can be used to send the same request to all devices. It uses `get_nodes_ipv6_addresses.py` to get the IPv6 addresses and traces the CoAP command, such that you can easily copy/paste it for a specific device if needed.

```
./coap_all /settings/auto_send
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a51c]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a51d]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a527]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a8ff]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a901]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2191]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2527]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2853]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8486]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8493]:5683/settings/auto_send : auto_send_sec: 60
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8503]:5683/settings/auto_send : auto_send_sec: 60
```

Based on the `coap_callback_auto_send()` implementation in the Wi-SUN device, `coap_all` can also be used to change the `auto_send` period for all nodes in a row:

```
./coap_all /settings/auto_send -e 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a51c]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a51d]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a527]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a8ff]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::62a4:23ff:fe37:a901]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2191]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2527]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b635:22ff:fe98:2853]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8486]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8493]:5683/settings/auto_send -e 120 : auto_send_sec: 120
  coap-client -m get -N -B 3 coap://[fd00:6172:6d00::b6e3:f9ff:fec5:8503]:5683/settings/auto_send -e 120 : auto_send_sec: 120
```

OTA DFU

Over-The-Air Device Firmware Upgrade

To upgrade an application on a device, you need to:

- Once only (on the Border Router platform):
  - Install libcoap2 & libcoap2-bin
  - Install a TFTP server
  - Start a CoAP notification server (optional, used to monitor the upgrade process)
  - Start a Linux Border Router
    - Add known IPv6 addresses to the Border Router `tun0` interface
- Once per device hardware:
  - Create an OTA-capable bootloader, with the compression algorithm(s) of your choice
- Once per project:
  - Add the OTA DFU component
    - This will add the CoAP and TFTP components
  - Set the Wi-SUN network to auto-connect to the Border Router (Optional. If not, you will need to trigger connection on the device)
- Once per device:
  - Flash the OTA-capable bootloader to the device, with the compression algorithm(s) of your choice
  - Flash an initial OTA-capable application to the device
  - Have the device connect to the Wi-SUN network
- For every upgrade:
  - Build the project
  - Create a `.gbl` file from the `.s37` file (optionally using compression)
  - Copy the `.gbl` to the expected name in the TFTP server folder
  - Trigger the OTA upgrade using a coap-client 'post' method
  - Check file download results
  - Trigger the reboot on the new firmware (the initial version of the OTA DFU component uses auto-reboot)

The following explains how to perform these steps based on the 'Wi-SUN SoC Empty' example application. OTA DFU can be applied using the same steps to all Wi-SUN example projects.

It is recommended to set an initial startup message in the application to show that the upgrade worked.

Prerequisites

`libcoap2` Installation

`libcoap2` is used for:

- The (optional) CoAP notification server
- The CoAP client used to trigger and monitor the firmware update

Both can be installed on separate machines, as long as they can be reached using IPv6. For the sake of simplicity, use the Linux platform supporting `wsbrd` for both functions. You will add a separate IPv6 address to the Border Router Wi-SUN interface to demonstrate this capability.

```
sudo apt-get install libcoap2
```

```
sudo apt-get install libcoap2-bin
```

TFTP Server Installation on the Linux Border Router

On the selected file server, install a TFTP server through the following steps. Here also you use the Linux platform used as the Border Router for this purpose, with a separate IPv6 address.

TFTP Daemon Installation

```
sudo apt-get install tftpd-hpa tftp-hpa
```

TFTP Service Configuration

```
sudo nano /etc/default/tftpd-hpa

# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/srv/tftp"
TFTP_ADDRESS=":69"
TFTP_OPTIONS="--secure"
```

(optional) Set the `TFTP_USERNAME` and `TFTP_DIRECTORY` to match you setup, if it differs from the defaults.

Restart the TFTPD Service

```
/etc/init.d/tftpd-hpa restart
```

From this point, files stored under the `TFTP_DIRECTORY` are accessible using TFTP.

Start a Linux Border Router

Start a Border Router set to allows connecting your devices. The devices should connect before adding OTA DFU to their application.

Add known IPv6 addresses to the Border Router (acting as CoAP server and Notification Server)

What you need here is:

- A TFTP server, such that your devices can send TFTP requests for the `.gbl` files.
- A (optional, highly recommended) CoAP Notification Server such that your devices can send progress notifications during OTA DFU download.

The TFTP and CoAP servers may not be on the Border Router platform, as long as they can be reached by your Wi-SUN devices via the Border Router. For the sake of simplicity in this first setup, use the same Linux platform as for `wsbrd` (i.e, your Wi-SUN Border Router) for both uses.

Using separate IPv6 addresses makes it clear that the three entities (Border Router, TFTP server, CoAP Notification Server) can be on different platforms. Since the Border Router application is 'wsbrd', you have control over the IPv6 prefix used by your Wi-SUN network (set in your `wsbrd.conf` file).

`ipv6_prefix = fd00:6172:6d00::/64`

You can then add a first IPv6 address to the `tun0` interface using this IPv6 prefix for the 'OTA DFU HOST' role:

```
sudo ip -6 address add fd00:6172:6d00::1/64 dev tun0
```

Add a second IPv6 address to the `tun0` interface 'OTA DFU NOTIFY HOST' role:

```
sudo ip -6 address add fd00:6172:6d00::2/64 dev tun0
```

Then check that you now have two IPv6 addresses for the 'tun0' interface:

```
ip address show tun0 | grep global'
```

```
  inet6 fd00:6172:6d00::1/64 scope global
  inet6 fd00:6172:6d00::2/64 scope global
  inet6 fd00:6172:6d00:0:92fd:9fff:fe00:333a/64 scope global
```

You will later set the IPv6 `OTA_DFU_HOST` and `OTA_DFU_NOTIFY_HOST` addresses in the OTA component code to match these fixed (`fd00:6172:6d00::1` and `fd00:6172:6d00::2` , respectively) addresses.

Start the CoAP Notification Server

The CoAP Notification server is optional, but it's useful to follow the upgrade process. Silicon Labs highly recommends using it.

Checking `coap-server` Options

Use `coap-server --help` to get the help. Below is an abstract with the version info and options used.

```
coap-server: invalid option -- '-'
coap-server v4.2.1 -- a small CoAP implementation
(c) 2010,2011,2015-2018 Olaf Bergmann <bergmann@tzi.org> and others

TLS Library: None

Usage: coap-server [-d max] [-g group] [-l loss] [-p port] [-v num]
            [-A address] [-N]
            [[-k key] [-h hint]]
            [[-c certfile][-C cafile] [-n] [-R root_cafile]]
General Options
      -d max         Allow dynamic creation of up to a total of max
                     resources. If max is reached, a 4.06 code is returned
                     until one of the dynamic resources has been deleted
      -g group       Join the given multicast group
      -p port        Listen on specified port
      -v num         Verbosity level (default 3, maximum is 9). Above 7,
                     there is increased verbosity in GnuTLS logging
      -A address     Interface address to bind to
      -N             Make "observe" responses NON-confirmable. Even if set
                     every fifth response will still be sent as a confirmable
                     response (RFC 7641 requirement)
```

Starting the COAP Notification Server

Start the CoAP notification server with `-d 10` to allow dynamic resource creation:

```
coap-server -A fd00:6172:6d00::2 -p 5685 -d 10
```

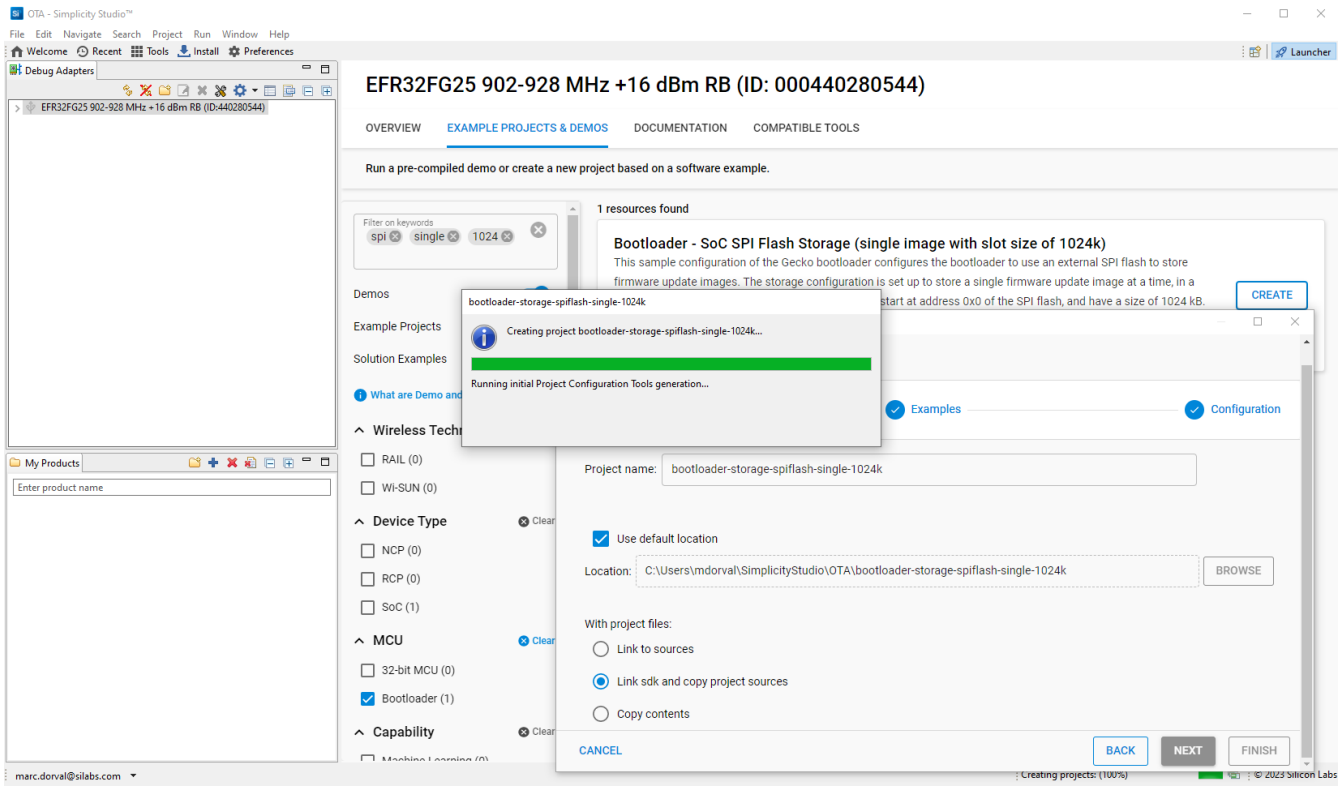
Bootloader Application

You need a Bootloader Application using SPI Flash for a single image of max 1024k bytes. This is to be manually flashed **once** to the part to allow OTA.

> TIP:If you use the **Erase** option before **Program** when flashing the Wi-SUN application, you will need to re-flash the bootloader.

In Simplicity Studio:

1. Select the board you want to flash the bootloader to in the **Debug Adapter** window.
2. Open the **Launcher** perspective.
3. Select the **EXAMPLE PROJECTS & DEMOS** tab.
4. On the projects filtering section, under **MCU**, check **Bootloader** to reduce the list of available projects.
5. Further reduce by filtering on SPI, single and 1024. Enter these strings in the **Filter on keywords** box, pressing **Enter** after each string.
6. 'Create' a **Bootloader - SoC SPI Flash Storage (single image with slot size of 1024k)** project.



Bootloader Configuration

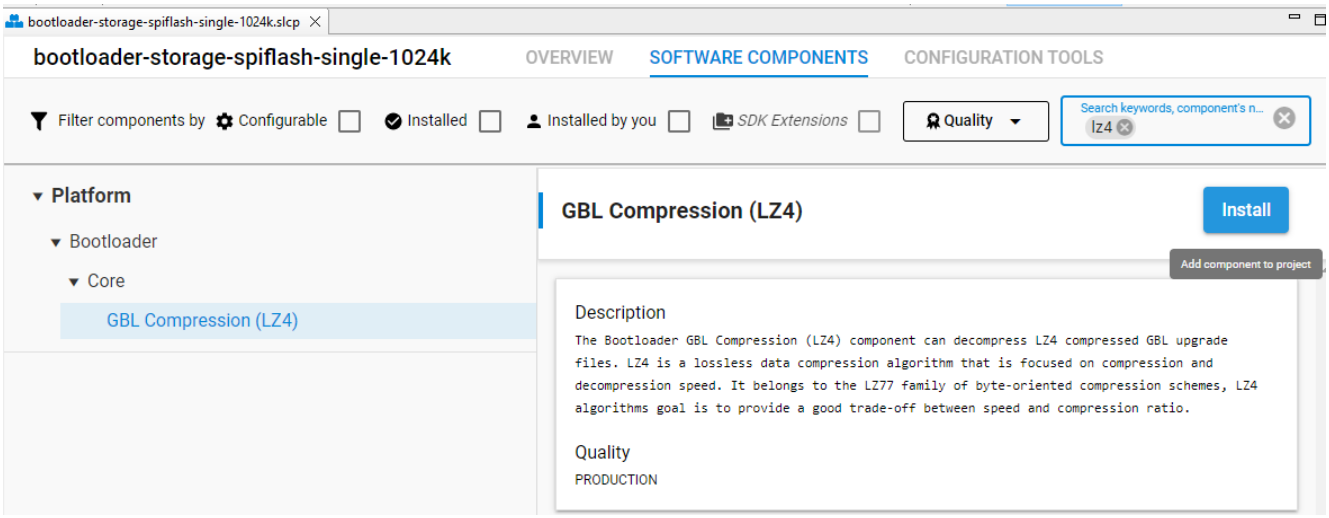
The Bootloader settings are defined in `config/btl_storage_cfg.h`. No need to change anything here, the default values are fine.

- The Base Address is defined as `BTL_STORAGE_BASE_ADDRESS`.
- The memory slots to store the images are defined as well.
- Only `SLOT0` is enabled, with a `SLOT0_SIZE` of `1048576` = `1024 * 1024` = 1024 kBytes, starting at `SLOT0_START 0`.

Add LZ4 Compression to Bootloader

By default, the bootloader projects don't include the LZ4 compression algorithm, so install it if you want to use it.

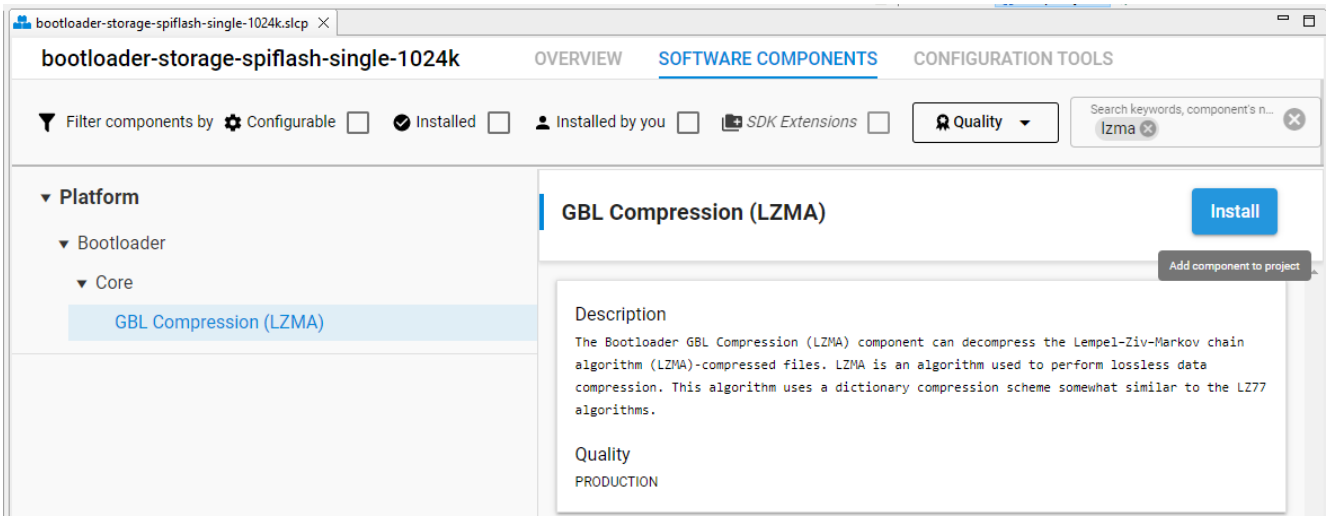
> TIP: Use lzma instead of lz4, lzma has better compression performance.



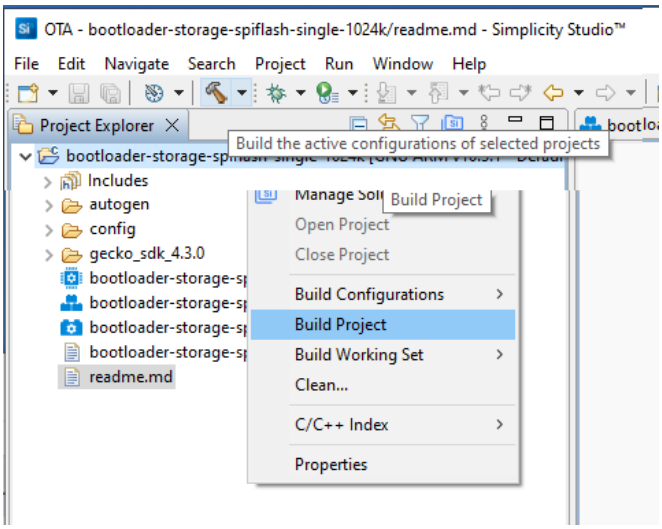
Add LZMA Compression to Bootloader

By default, the bootloader projects don't include the most efficient LZMA compression algorithm, so install it if you want to use it.

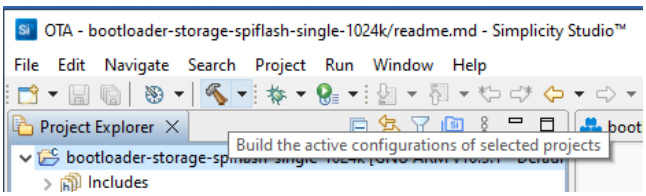
> TIP: Using lzma to compress `.gbl` files to save transmission time and therefore power when transmitting data.



Build the Bootloader Project



or

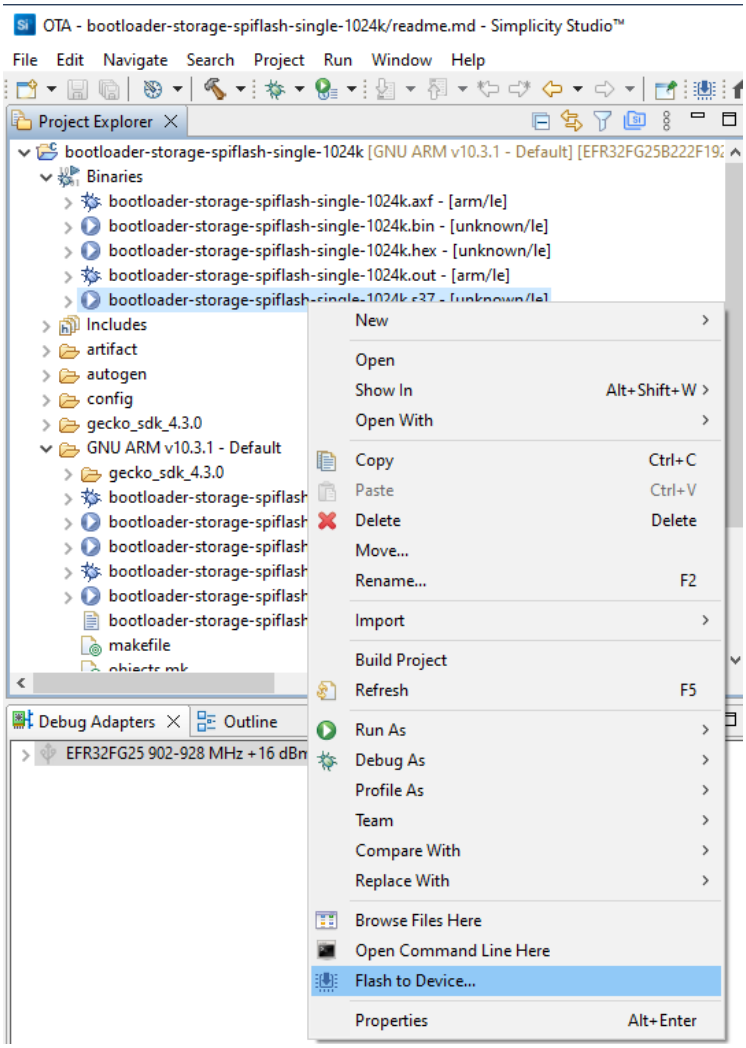


See Simplicity Studio 5 Users Guide for details.

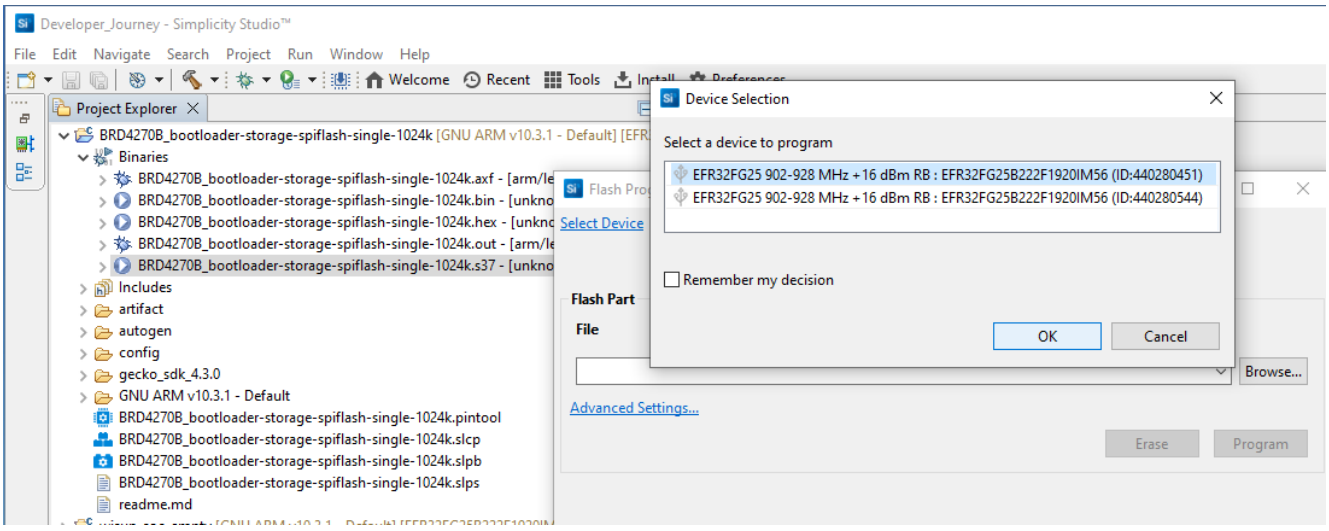
Flash the Bootloader Binary to the Debug Adapter

A new `Binaries` folder appeared in the project. This corresponds to the binaries present under `GNU ARM v10.3.1 - Default`.

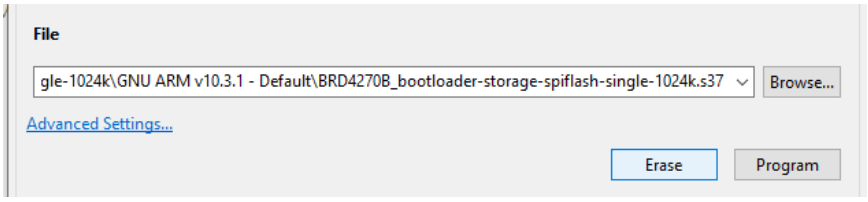
- On Series 1 (xG12, for example), select the `<projectname>-combined.s37` binary (see [UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher](https://www.silabs.com/documents/public/user-guides/ug489-gecko-bootloader-user-guide-gsdk-4.pdf) section 6, comments on Series 1 and 2 for details).
- On Series 2 (xG25 or xG28, for example), select the `<projectname>.s37` binary.



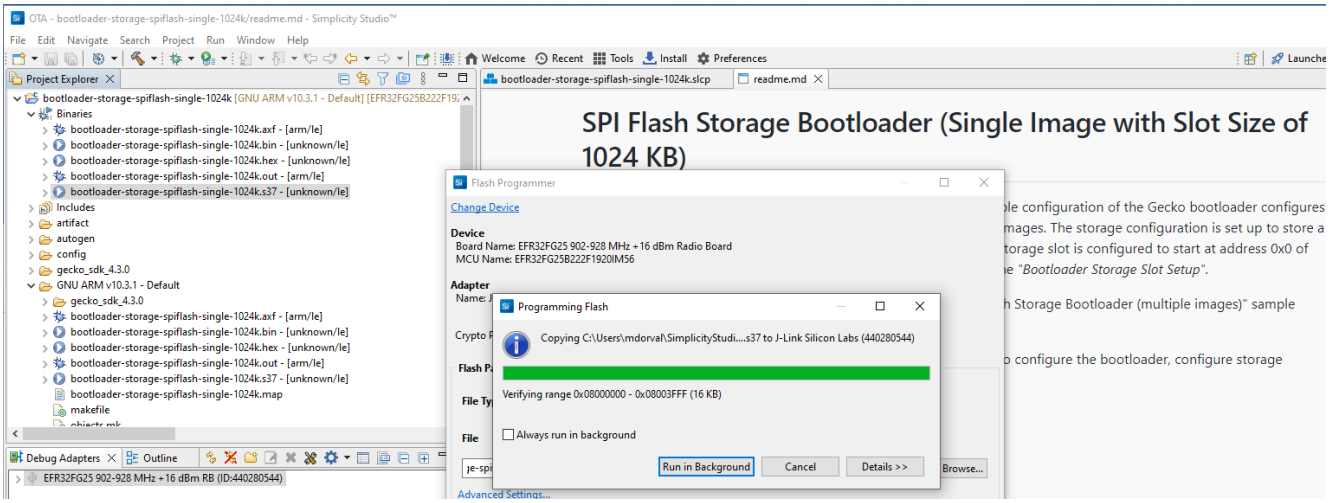
If you have several devices in your **Debug Adapters** window, you will be asked to select the device.



> TIP: It's always better to erase the part before flashing a bootloader. As opposed to this, do not erase before flashing the Wi-SUN application; otherwise, you will need to re-flash the bootloader.



Finally, click the **Program** button.



Application Project Creation (if not already existing)

If you have no previous application to flash to the device for testing, you can create a new 'Wi-SUN SoC Empty' project as described in Building and Connecting to a Wi-SUN Network. Otherwise, use your working Wi-SUN project.

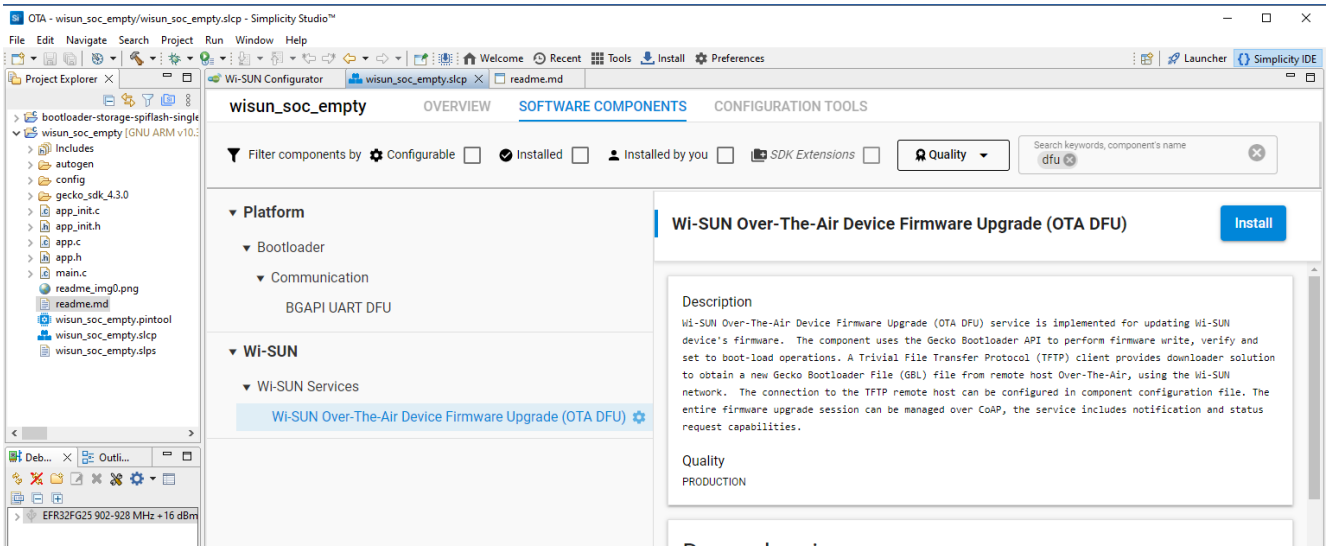
Adding the 'Wi-SUN Over-The-Air Device Firmware Upgrade (OTA DFU)' Component

To perform OTA upgrades:

- The bootloader on your device must support OTA DFU.
- The application running on your device must support OTA DFU.

In the **Simplicity IDE** perspective, open `wisun_soc_empty.slcp` select the **SOFTWARE COMPONENTS** tab, filter on 'DFU', and look for the **Wi-SUN Over-The-Air Device Firmware Upgrade (OTA DFU)** Component.

Install the OTA DFU Component



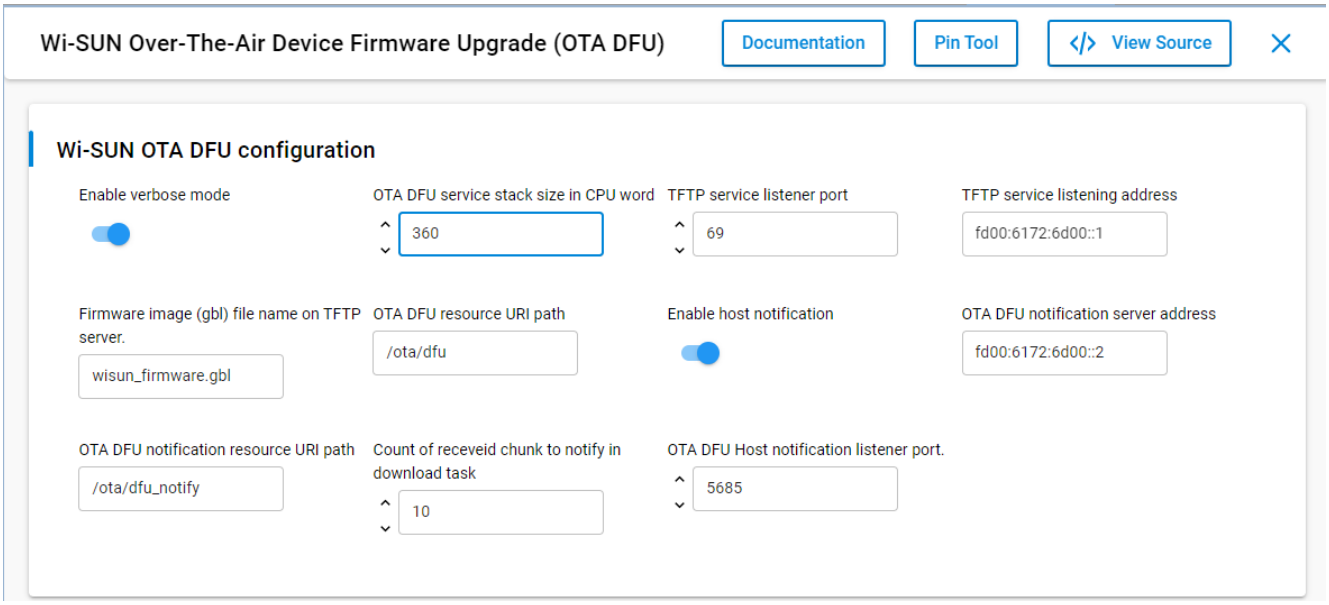
Check the OTA DFU Settings

> NOTE: It will take ~30 seconds for the **Configure** button to be active after installing the OTA DFU component.

There are two ways to change the OTA DFU component settings. The first way is through the **SOFTWARE COMPONENTS** view.

1. In the **SOFTWARE COMPONENTS** view, click **Configure** to access the corresponding GUI. Changes you make here will be reflected in `config/sl_wisun_ota_dfu_config.h` .
   - Verbose mode is enabled by default to follow the upgrade process in the device's console.
   - Host notifications are enabled by default to follow the upgrade process from the Border Router.
   - The **OTA DFU service stack size in CPU word** is set to 360 words (1440 bytes) to avoid stack overflow.
2. Set the **TFTP service listening address** to match the TFTP server IPv6.
3. Set the **OTA DFU notification server address** to match the notification server IPv6.

Optionally, you can change the **Firmware image (gbl) file name on TFTP server** to match your hardware. This can make it easier to manage various devices in the future. For the time being, keep using `wisun_firmware.gbl` .



- The OTA DFU GUI reflects in `config/sl_wisun_ota_dfu_config.h`. You can open it from the GUI, using the **View Source** button.

sl_wisun_ota_dfu_config.h

```
/* @file
#ifndef SL_WISUN_OTA_DFU_CONFIG_H
#define SL_WISUN_OTA_DFU_CONFIG_H

/*******************************************************************************//**
 * @defgroup SL_WISUN_OTA_DFU_CONFIG Configuration
 * @ingroup SL_WISUN_OTA_DFU_API
 * @{
 ******************************************************************************/

// <<< Use Configuration Wizard in Context Menu >>>

// <h>Wi-SUN OTA DFU configuration

// <q SL_WISUN_OTA_DFU_VERBOSE_MODE_ENABLED> Enable verbose mode
// <i> Default value: 0
#define SL_WISUN_OTA_DFU_VERBOSE_MODE_ENABLED       1U

// <o SL_WISUN_OTA_DFU_STACK_SIZE_WORD> OTA DFU service stack size in CPU word
// <i> Default: 256
#define SL_WISUN_OTA_DFU_STACK_SIZE_WORD            360UL

// <o SL_WISUN_OTA_DFU_TFTP_PORT> TFTP service listener port
// <i> Default: 69
// <i> This is the port number where TFTP service is listening
// <1-65536>
#define SL_WISUN_OTA_DFU_TFTP_PORT                  69U

// <s SL_WISUN_OTA_DFU_HOST_ADDR> TFTP service listening address
#define SL_WISUN_OTA_DFU_HOST_ADDR                  "fd00:6172:6d00::1"

// <s SL_WISUN_OTA_DFU_GBL_FILE> Firmware image (gbl) file name on TFTP server.
#define SL_WISUN_OTA_DFU_GBL_FILE                   "wisun_firmware.gbl"

// <s SL_WISUN_OTA_DFU_URI_PATH> OTA DFU resource URI path
#define SL_WISUN_OTA_DFU_URI_PATH                   "/ota/dfu"

// <q SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED> Enable host notification
// <i> Default value: 1
#define SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED        1U

#if SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED
// <s SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR> OTA DFU notification server address
#define SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR           "fd00:6172:6d00::2"

// <s SL_WISUN_OTA_DFU_NOTIFY_URI_PATH> OTA DFU notification resource URI path
#define SL_WISUN_OTA_DFU_NOTIFY_URI_PATH            "/ota/dfu_notify"

// <o SL_WISUN_OTA_DFU_NOTIFY_DOWNLOAD_CHUNK_CNT> Count of receveid chunk to notify in download task
// <i> Default value: 10
#define SL_WISUN_OTA_DFU_NOTIFY_DOWNLOAD_CHUNK_CNT  10U

// <o SL_WISUN_OTA_DFU_NOTIFY_PORT> OTA DFU Host notification listener port.
// <i> Default: 5685
#define SL_WISUN_OTA_DFU_NOTIFY_PORT                5685U
#endif

// </h>
// <<< end of configuration section >>>

/** @}*/

#endif
```

The settings must match between:

- The TFTP server IPv6 (`fd00:6172:6d00::1`)
- The `.gbl` file name
- The CoAP Notification server IPv6 (`fd00:6172:6d00::2`)
- The OTA DFU component
- The `coap-server` command
- The `coap-client` commands

> TIP: Use the OTA DFU cheat sheet to check your settings.

Enable OTA DFU Verbose Mode

- In `config/sl_wisun_ota_dfu_config.h`, `SL_WISUN_OTA_DFU_VERBOSE_MODE_ENABLED` is set to `1U` to get messages traced in the device's console during download. Once you get familiar with OTA DFU, you can disable verbose mode.

```
[wisun-btl] (0) Storage info: version: 196608, capabilities: 0, storageType: 0, numStorageSlots: 1
[wisun-btl] (7) Firmware upgrade started
[wisun-btl] (17) TFTP init done
[wisun-btl] (27) TFTP download started: tftp://[fd00:6172:6d00::1]:69/wisun_firmware.gbl
[wisun-btl] (429) download: received chunk 1, offset: 0x00000000
[wisun-btl] (853) download: received chunk 2, offset: 0x00000200
[wisun-btl] (1077) download: received chunk 3, offset: 0x00000400
[wisun-btl] (1302) download: received chunk 4, offset: 0x00000600
[wisun-btl] (1526) download: received chunk 5, offset: 0x00000800
[wisun-btl] (1949) download: received chunk 6, offset: 0x00000a00
[wisun-btl] (2174) download: received chunk 7, offset: 0x00000c00
```

Enable OTA DFU Notifications

- In `config/sl_wisun_ota_dfu_config.h`, `SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED` is set to `1U` to get notification CoAP messages sent by the device to the notification server during download, every `SL_WISUN_OTA_DFU_NOTIFY_DOWNLOAD_CHUNK_CNT` chunks (default `10U`). This can be changed later on to a larger value to reduce the amount of notification messages. Setting it to a very large number will avoid all intermediate notifications while still keeping the final notification message, which is important to indicate file download completion and the validity of the new firmware.
  - The notification server doesn't need to be the Border Router. It can be any other machine. Use a dedicated IPv6 address to make this clear.

Version 1 Application with OTA DFU Support

Now that you have OTA DFU configured, you can set your Wi-SUN network to match your Border Router. Do this with the **Wi-SUN Configurator**.

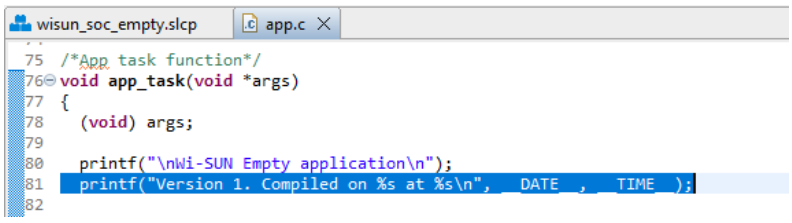
Set the Wi-SUN Configuration

Set the Wi-SUN Configuration to match the Border Router setup.

Add a Startup Text Indicating the Version of the Application

Here, add the following to `app.c/app_task()` to get minimal information on your application and make sure you can tell which version is running through the startup messages that you'll get in the Serial 1 console when you reset the device.

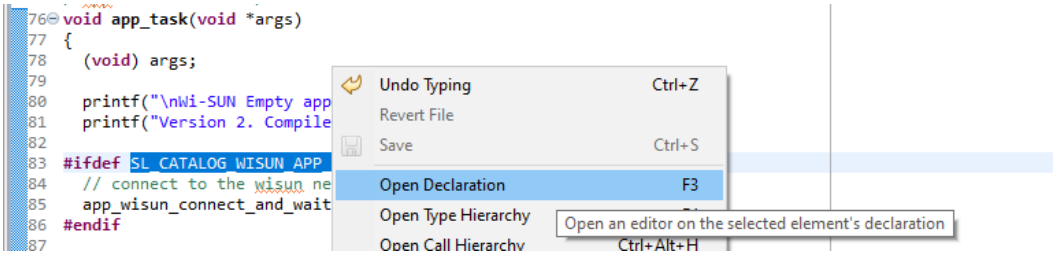
```
printf("Version 1. Compiled on %s at %s\n", __DATE__, __TIME__);
```



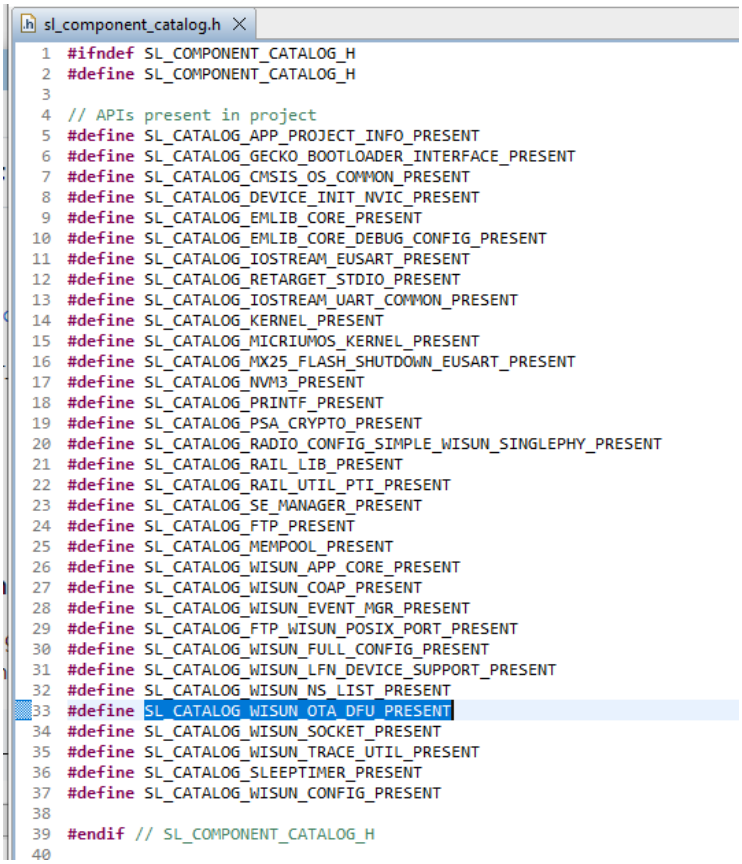
(Optional) Add a Startup Text Indicating OTA DFU Support

Since it's interesting to know if OTA DFU is supported, add a second message if OTA DFU is supported.

To locate where active components are declared, select in `app.c/app_task()` the `SL_CATALOG_WISUN_APP_CORE_PRESENT` text on line 83, right-click, and select **Open Declaration**.



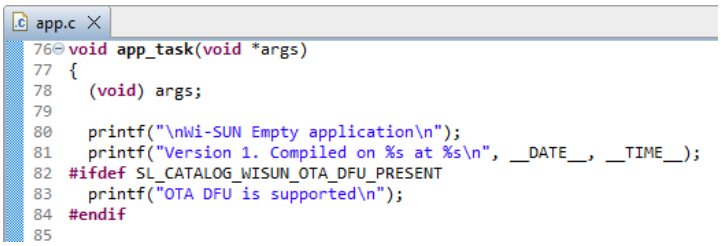
You end up in `autogen/sl_component_catalog.h`, where `SL_CATALOG_WISUN_OTA_DFU_PRESENT` is declared on line 33.



> TIP: You also see that `SL_CATALOG_WISUN_COAP_PRESENT` is declared on line 27, indicating that the CoAP component is also present. CoAP is a dependency for the OTA DFU component, so it has been installed automatically when you added OTA DFU (if not previously installed).

Back to `app.c/app_task()`, add the following code below your previous message:

```
#ifdef SL_CATALOG_WISUN_OTA_DFU_PRESENT
 printf("OTA DFU is supported\n");
#endif
```



(Optional) Add More Information on OTA DFU Once Connected

Once the device is connected to the Wi-SUN network the application can retrieve it's IPv6 global address and display information on how OTA DFU can be used.

This is interesting to add for the first OTA DFU tests, to get you familiar with the commands and the parameters.

In `app.c`, add the following code in the `#include` area:

```
#include "sl_wisun_ota_dfu_config.h"
```

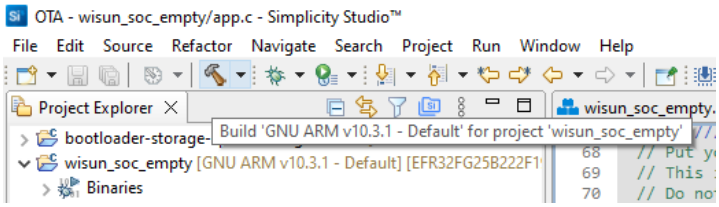
and the following code before the `while (1)` loop in `app_task()`

```
#ifdef SL_CATALOG_WISUN_OTA_DFU_PRESENT
  sl_wisun_ip_address_t global_ipv6;

  printf("OTA DFU will download chunks of '<TFTP_DIRECTORY>/%s' from %s/%d\n",
         SL_WISUN_OTA_DFU_GBL_FILE,
         SL_WISUN_OTA_DFU_HOST_ADDR,
         SL_WISUN_OTA_DFU_TFTP_PORT
        );

  sl_wisun_get_ip_address(SL_WISUN_IP_ADDRESS_TYPE_LINK_LOCAL, &global_ipv6);
  printf("OTA DFU 'start' command:\n");
  printf(" coap-client -m post -N -B 10 -t text coap://[%s]:%d%s -e \"start\"\n",
         app_wisun_trace_util_get_ip_str(&global_ipv6),
         5683,
         SL_WISUN_OTA_DFU_URI_PATH
        );
  printf("Follow OTA DFU progress (from node, intrusive) using:\n");
  printf(" coap-client -m get -N -B 10 -t text coap://[%s]:%d%s\n",
      app_wisun_trace_util_get_ip_str(&global_ipv6),
      SL_WISUN_COAP_RESOURCE_HND_SERVICE_PORT,
      SL_WISUN_OTA_DFU_URI_PATH
  );

  if (SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED) {
     printf("OTA DFU notifications enabled (every %d chunks)\n",
       SL_WISUN_OTA_DFU_NOTIFY_DOWNLOAD_CHUNK_CNT
     );
     printf("OTA DFU notifications will be POSTed to notification server coap://[%s]:%d%s\n",
         SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR,
         SL_WISUN_OTA_DFU_NOTIFY_PORT,
         SL_WISUN_OTA_DFU_NOTIFY_URI_PATH
     );
    printf("Follow OTA DFU progress (from notification server) using:\n");
    printf(" coap-client -m get -N -B 1 -t text coap://[%s]:%d%s\n",
      SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR,
      SL_WISUN_OTA_DFU_NOTIFY_PORT,
      SL_WISUN_OTA_DFU_NOTIFY_URI_PATH
    );
  }
#endif
```

app.c

```
#include "sl_wisun_api.h"
#ifdef SL_CATALOG_WISUN_APP_CORE_PRESENT
  #include "sl_wisun_app_core.h"
#endif
#include "sl_wisun_ota_dfu_config.h"

// -----------------------------------------------------------------------------
//                              Macros and Typedefs
// -----------------------------------------------------------------------------

// -----------------------------------------------------------------------------
//                          Static Function Declarations
// -----------------------------------------------------------------------------

// -----------------------------------------------------------------------------
//                                Global Variables
// -----------------------------------------------------------------------------

// -----------------------------------------------------------------------------
//                                Static Variables
// -----------------------------------------------------------------------------

// -----------------------------------------------------------------------------
//                          Public Function Definitions
// -----------------------------------------------------------------------------
#ifndef SL_CATALOG_WISUN_EVENT_MGR_PRESENT // Event Manager also defines this handler
/*Wi-SUN event handler*/
void sl_wisun_on_event(sl_wisun_evt_t *evt)
#endif

/*App task function*/
void app_task(void *args)
{
  (void) args;

  printf("\nWi-SUN Empty application\n");
  printf("Version 3. Compiled on %s at %s\n", __DATE__, __TIME__);
#ifdef SL_CATALOG_WISUN_OTA_DFU_PRESENT
  printf("OTA DFU is supported\n");
#endif
#ifdef SL_CATALOG_WISUN_APP_CORE_PRESENT
  // connect to the wisun network
  app_wisun_connect_and_wait();
#endif

#ifdef SL_CATALOG_WISUN_OTA_DFU_PRESENT
  sl_wisun_ip_address_t global_ipv6;

  printf("OTA DFU will download chunks of '<TFTP_DIRECTORY>/%s' from %s/%d\n",
         SL_WISUN_OTA_DFU_GBL_FILE,
         SL_WISUN_OTA_DFU_HOST_ADDR,
         SL_WISUN_OTA_DFU_TFTP_PORT
        );

  sl_wisun_get_ip_address(SL_WISUN_IP_ADDRESS_TYPE_LINK_LOCAL, &global_ipv6);
  printf("OTA DFU 'start' command:\n");
  printf(" coap-client -m post -N -B 10 -t text coap://[%s]:%d%s -e \"start\"\n",
         app_wisun_trace_util_get_ip_str(&global_ipv6),
         5683,
         SL_WISUN_OTA_DFU_URI_PATH
        );

  if (SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED) {
      printf("OTA DFU notifications enabled (every %d chunks)\n",
        SL_WISUN_OTA_DFU_NOTIFY_DOWNLOAD_CHUNK_CNT
      );
      printf("OTA DFU notifications will be POSTed to notification server coap://[%s]:%d%s\n",
         SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR,
         SL_WISUN_OTA_DFU_NOTIFY_PORT,
         SL_WISUN_OTA_DFU_NOTIFY_URI_PATH
      );
    printf("Follow OTA DFU progress (from notification server) using:\n");
    printf(" coap-client -m get -N -B 1 -t text coap://[%s]:%d%s\n",
       SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR,
       SL_WISUN_OTA_DFU_NOTIFY_PORT,
       SL_WISUN_OTA_DFU_NOTIFY_URI_PATH
    );
  }
#endif

  while (1) {
    ////////////////////////////////////////////////////////////////////////////
    // Put your application code here!                                        //
    ////////////////////////////////////////////////////////////////////////////
    osDelay(1);
  }
}
```

Rebuild the 'Version 1' Application



Unfortunately, with GSDK 4.3.0, this will fail because of an assert from `<gecko_sdk>/app/wisun/component/ftp_posix_port/sl_wisun_ftp_posix_port.c` line `160`. The root cause is not the assert. The assert is here to stop you at this point such that you correct the underlying code and safely compile with no runtime issue to expect. If you don't solve the assert, you may get issues when running your code.

The point is that the TFTP code is using a data block size of 512 bytes by default, while the socket buffer is using 128 bytes per default. As a consequence, the TFTP packets won't fit in the socket buffer, so the socket buffer needs to be bigger.

With GSDK 4.3.0, setting this was not done automatically in the generation flow when adding TFTP and the OTA DFU component, so this change needs to be done manually. This is improved in later GSDK versions using the following condition in the `.slcp` file:

```
  - name: "SL_SOCKET_BUFFER_SIZE"
    value: "532"
    condition:
      - "sl_wisun_ota_dfu"
```

With GSDK 4.3.0, you will experience this error:

```
"gecko_sdk_4.3.0/app/wisun/component/coap/mbed-coap/sn_coap_builder.o"
"C:/Users/mdorval/SimplicityStudio/SDKs/gecko_sdk_8/app/wisun/component/coap/mbed-coap/sn_coap_builder.c"
Finished building: C:/Users/mdorval/SimplicityStudio/SDKs/gecko_sdk_8/app/wisun/component/ftp/sl_ftp.c
C:/Users/mdorval/SimplicityStudio/SDKs/gecko_sdk_8/app/wisun/component/ftp_posix_port/sl_wisun_ftp_posix_port.c:160:1:
error: static assertion failed: "TFTP: Not enough socket buffer size (increase SL_SOCKET_BUFFER_SIZE to be >=
SL_TFTP_DATA_BLOCK_SIZE + 4)"
  160 | _Static_assert(SL_SOCKET_BUFFER_SIZE >= (SL_TFTP_DATA_BLOCK_SIZE + sizeof(uint16_t) * 2UL),
      | ^~~~~~~~~~~~~~
make: *** [gecko_sdk_4.3.0/app/wisun/component/ftp_posix_port/subdir.mk:20:
gecko_sdk_4.3.0/app/wisun/component/ftp_posix_port/sl_wisun_ftp_posix_port.o] Error 1
make: *** Waiting for unfinished jobs....

Finished building: C:/Users/mdorval/SimplicityStudio/SDKs/gecko_sdk_8/app/wisun/component/mempool/sl_mempool.c
Finished building: C:/Users/mdorval/SimplicityStudio/SDKs/gecko_sdk_8/app/wisun/component/socket/socket.c
```

As the error message indicates, the solution consists of increasing `SL_SOCKET_BUFFER_SIZE` to be higher than `SL_TFTP_DATA_BLOCK_SIZE + 4`.

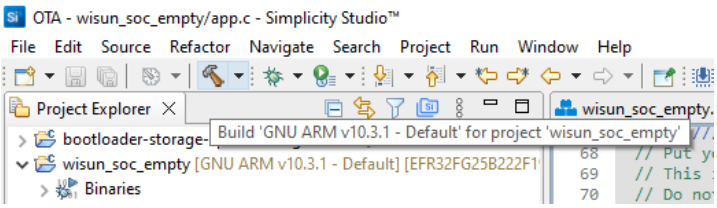
- You find the value of `SL_TFTP_DATA_BLOCK_SIZE` using **Open Declaration (F3)**.

```
#define SL_TFTP_DATA_BLOCK_SIZE      512UL
```

- Go to the `SL_SOCKET_BUFFER_SIZE` declaration and set it to 512 + 4 = 516.

```
#define SL_SOCKET_BUFFER_SIZE                  516U
```

Now you can compile without errors.

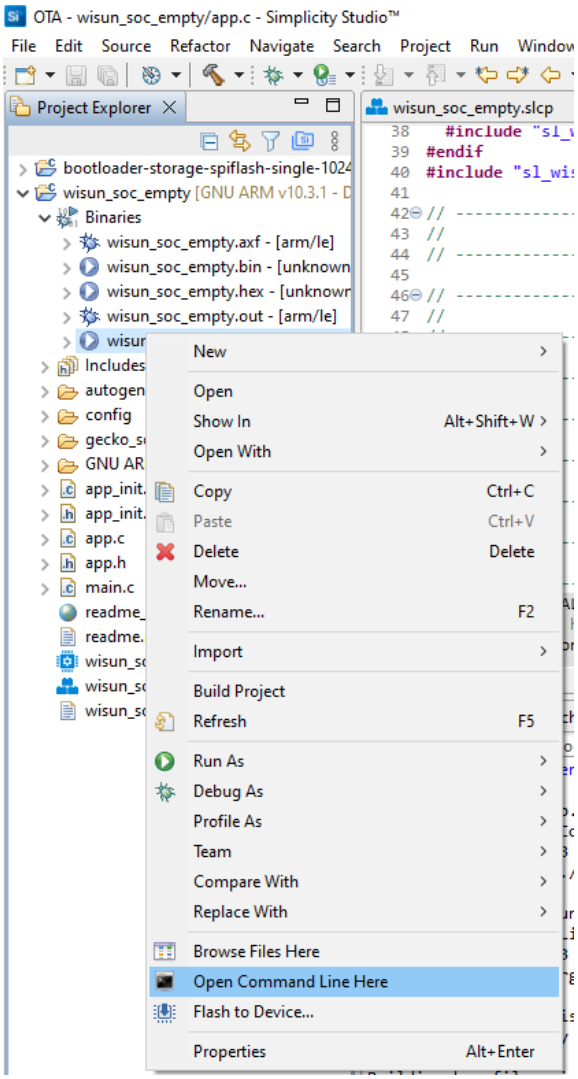



```
Building s37 file: wisun_soc_empty.s37
arm-none-eabi-objcopy -O srec "wisun_soc_empty.axf" "wisun_soc_empty.s37"

Running size tool
arm-none-eabi-size "wisun_soc_empty.axf" -A
wisun_soc_empty.axf  :
section              size        addr
.text              624092   134242304
.ARM.exidx              8   134866396
.copy.table            12   134866404
.zero.table             0   134866416
.stack               4096   536870912
.data                5288   536875008
.bss                36752   536880296
.heap               54272   536917048
.nvm                40960   134866416
.ARM.attributes        54           0
.comment               73           0
.debug_info       3099194           0
.debug_abbrev      262623           0
.debug_loc        1269246           0
.debug_aranges      46856           0
.debug_ranges       99312           0
.debug_macro       467069           0
.debug_line       1600370           0
.debug_str        2019085           0
.debug_frame       153332           0
Total             9782694


15:00:46 Build Finished. 0 errors, 0 warnings. (took 5s.675ms)
```

Manually Install the 'Version 1' Application

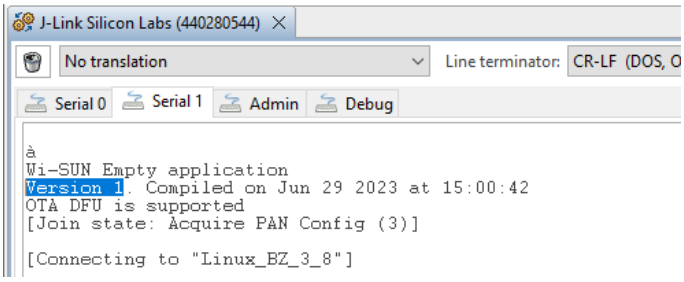
1. Flash it manually (if it's the first OTA-capable application you flash, it needs to be flashed manually, since resources to flash it using OTA DFU are not present yet).

> NOTE: If you use the **Erase** Feature, you will need to re-flash the bootloader then flash the application.



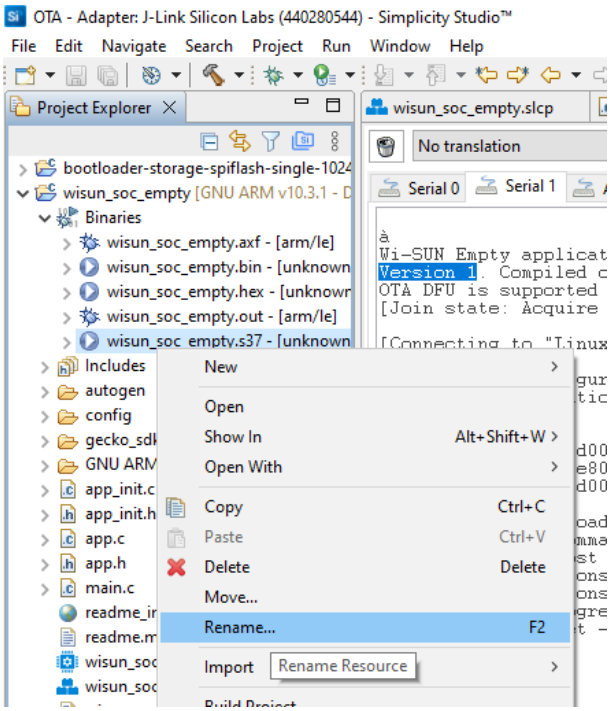
2. Open a console on the device.
3. Select **Serial 1** and press **Enter** to connect.
4. Reset the device (using the **RESET** button on the WSTK/WPK).
    - The startup message indicates that:
        - You're running 'Version 1'
        - OTA DFU is supported

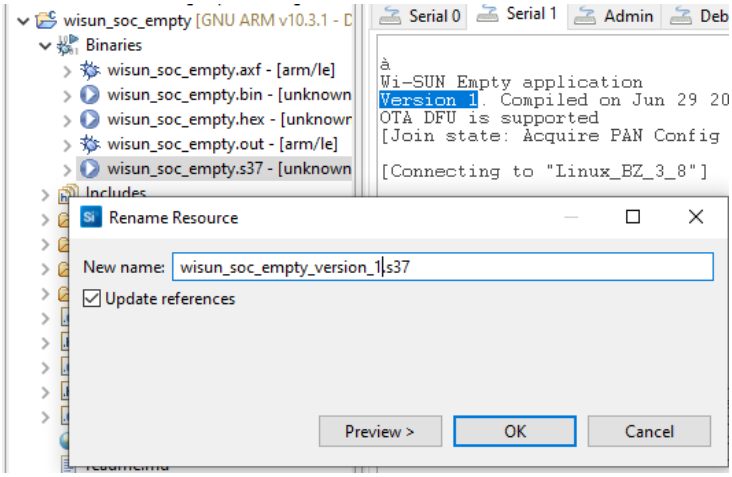
The 'Wi-SUN SoC Empty' example application is set for auto-connection to the Wi-SUN Network, so you see it starting to connect.

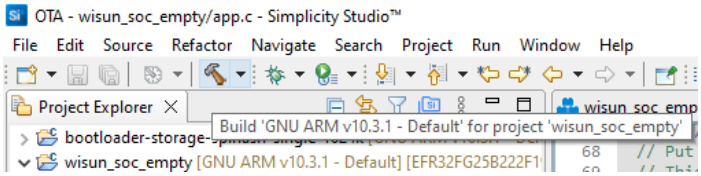


Rename the 'Version 1'Binary to Keep It

Add `_version_1` to the file name to clearly identify it.







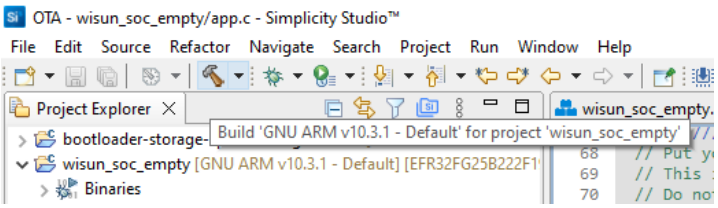
Version 2 Application with OTA DFU Support

Change the Startup Text for 'Version 2'

```
printf("Version 2. Compiled on %s at %s\n", __DATE__, __TIME__);
```

```
app.c
76 void app_task(void *args)
77 {
78   (void) args;
79
80   printf("\nWi-SUN Empty application\n");
81   printf("Version 2. Compiled on %s at %s\n", __DATE__, __TIME__);
82 #ifdef SL_CATALOG_WISUN_OTA_DFU_PRESENT
83   printf("OTA DFU is supported\n");
84 #endif
```

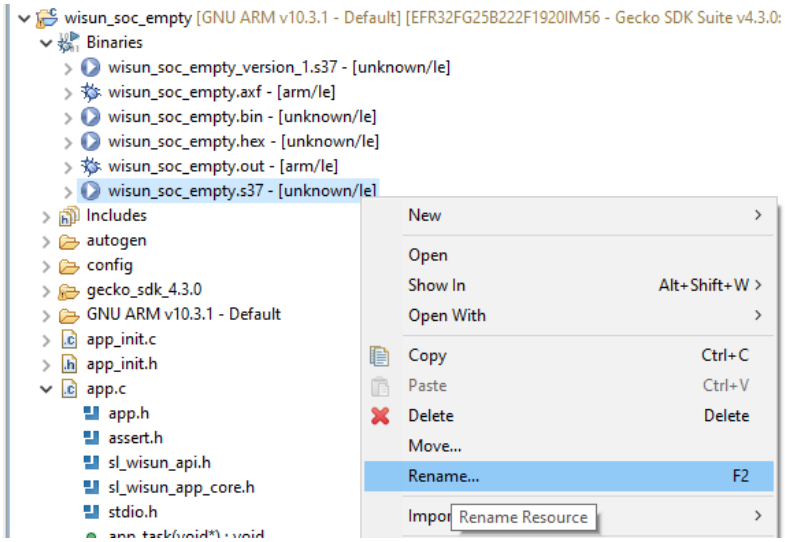
Rebuild the 'Version 2' Application



The `wisun_soc_empty.s37` file is back in the `Binaries` folder. It now corresponds to your 'Version 2' application.

Rename the 'Version 2' Binary to Keep It

Add `_version_2` to the file name to clearly identify it.



Convert the 'Version 2' Binary to GBL Format

The 'Version 2' binary needs to be converted to the GBL (Gecko BootLoader) format using Simplicity Commander. It will have a `.gbl` extension after conversion.

On Windows platforms, Simplicity Commander is by default installed under `C:\SiliconLabs\SimplicityStudio\v5\developer\adapter_packs\commander`.

> TIP: To access Simplicity Commander easily, it's a good option to add the path to the `commander.exe` in your environment variables. Remember that command line windows only check environment variables upon starting,

> so you need to re-open them to benefit from the change.

- Open a command window in the `C:\Users\username\SimplicityStudio\v5_workspace\wisun_soc_empty\GNU ARM v10.3.1 - Default` folder where binaries are stored.
- Check that the `.s37` files are present:

```
C:\Users\username\SimplicityStudio\v5_workspace\wisun_soc_empty\GNU ARM v10.3.1 - Default>dir *.s37

29/06/2023  15:00         1 888 290 wisun_soc_empty_version_1.s37
29/06/2023  15:12         1 888 290 wisun_soc_empty_version_2.s37
```

- Create the GBL file for the application to transfer over OTA DFU
  - `commander gbl create --app wisun_soc_empty_version_2.s37 wisun_soc_empty_version_2.gbl`
  - To reduce the size of the file to be transferred, you can compress the resulting file with either `--compress lz4` or `--compress lzma`
    - `commander gbl create --app wisun_soc_empty_version_2.s37 wisun_soc_empty_version_2_lz4.gbl --compress lz4`
    - `commander gbl create --app wisun_soc_empty_version_2.s37 wisun_soc_empty_version_2_lzma.gbl --compress lzma`

> CAUTION: Only compress with the algorithms supported by your bootloader, otherwise the verification step will fail once downloaded.

```
C:\Users\username\SimplicityStudio\OTA\wisun_soc_empty\GNU ARM v10.3.1 - Default>commander gbl create --app
wisun_soc_empty_version_2.s37 wisun_soc_empty_version_2.gbl
Parsing file wisun_soc_empty_version_2.s37...
Initializing GBL file...
Adding application to GBL...
Writing GBL file wisun_soc_empty_version_2.gbl...
DONE

C:\Users\username\SimplicityStudio\OTA\wisun_soc_empty\GNU ARM v10.3.1 - Default>commander gbl create --app
wisun_soc_empty_version_2.s37 wisun_soc_empty_version_2_lz4.gbl --compress lz4
Parsing file wisun_soc_empty_version_2.s37...
Initializing GBL file...
Adding application to GBL...
Compressing using lz4...
Writing GBL file wisun_soc_empty_version_2_lz4.gbl...
DONE

C:\Users\username\SimplicityStudio\OTA\wisun_soc_empty\GNU ARM v10.3.1 - Default>commander gbl create --app
wisun_soc_empty_version_2.s37 wisun_soc_empty_version_2_lzma.gbl --compress lzma
Parsing file wisun_soc_empty_version_2.s37...
Initializing GBL file...
Adding application to GBL...
Compressing using lzma...
Writing GBL file wisun_soc_empty_version_2_lzma.gbl...
DONE
```

As you can see, the best compression method is `lzma` :

Compression	Size (bytes)	None/Compressed ratio
None	629476	100 %
lz4	544424	86 %
lzma	385996	61 %

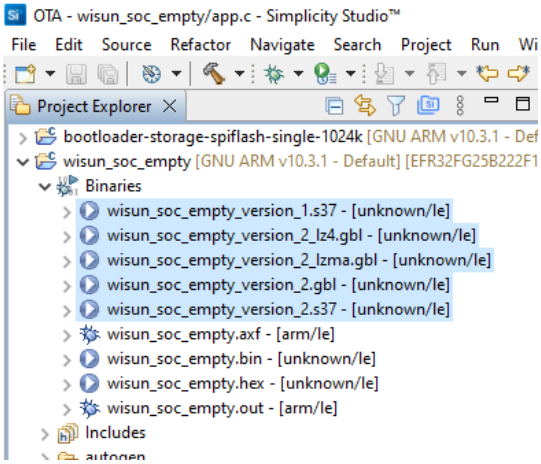
```
C:\Users\username\SimplicityStudio\v5_workspace\wisun_soc_empty\GNU ARM v10.3.1 - Default>dir *version_2*.gbl
```

```
29/06/2023  15:17        629476 wisun_soc_empty_version_2.gbl
29/06/2023  15:18        544424 wisun_soc_empty_version_2_lz4.gbl
29/06/2023  15:18        385996 wisun_soc_empty_version_2_lzma.gbl
```

At this point, you have all the resources you need to start using OTA DFU to upgrade from 'Version 1' (manually flashed) to 'Version 2' (using OTA DFU).

- A running TFTP server on the Linux Border Router.
- A bootloader with OTA DFU support, flashed to your device.
- A 'Version 1' application with OTA DFU support, flashed to your device.
- A 'Version 2' file in GBL format (you'll use the lzma-compressed file).
  - The 'Version 2' binary also has OTA DFU support, to get ready to accept future upgrades to a 'Version 3' application.

And you can follow the process using the notifications, which will be sent to the notification server by the device.



Transfer the 'Version 2' `.gbl` Files to the Linux Border Router

From the machine where the `.gbl` files were created, use `scp` to copy the files to the Linux Border Router host. You can use your preferred file copy method.

```
scp wisun_soc_empty_*.gbl <linux_user>@<linux_hostname>:/tmp/
```

> NOTE: The rest of the operations will occur on the Linux Border Router.

Check 'Version 1' Connection to the Border Router

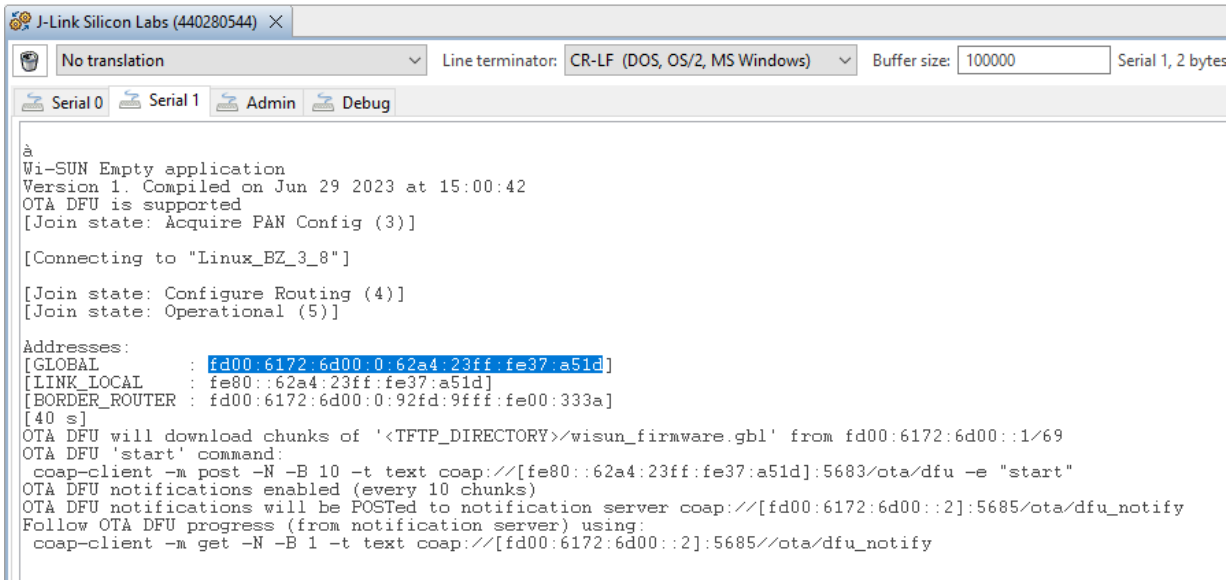
On the Border Router

In `wsbrd_cli status`, the MAC address of the device should be visible:

```
network_name: Linux_BZ_3_8
fan_version: FAN 1.1
domain: BZ
phy_mode_id: 8
chan_plan_id: 3
panid: 0xd4ec
size: CERT
GAK[0]: 36:b6:1e:81:fa:c4:88:b4:03:f7:b4:9b:38:b3:41:47
GAK[1]: 79:8b:b2:98:7f:ef:d3:55:4a:b8:cf:23:d4:a2:8f:5b
GAK[2]: 79:8b:b2:98:7f:ef:d3:55:4a:b8:cf:23:d4:a2:8f:5b
GAK[3]: 79:8b:b2:98:7f:ef:d3:55:4a:b8:cf:23:d4:a2:8f:5b
GTK[0]: 00:10:20:30:40:50:60:70:80:90:a0:b0:c0:d0:e0:f0
GTK[1]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
GTK[2]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
GTK[3]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
LGAK[0]: 62:80:53:11:03:45:b6:f5:16:67:e6:14:65:21:e4:99
LGAK[1]: 79:8b:b2:98:7f:ef:d3:55:4a:b8:cf:23:d4:a2:8f:5b
LGAK[2]: 79:8b:b2:98:7f:ef:d3:55:4a:b8:cf:23:d4:a2:8f:5b
LGTK[0]: 66:70:4e:08:8c:ce:82:c9:d2:aa:c7:62:83:18:97:b9
LGTK[1]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
LGTK[2]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
90:fd:9f:ff:fe:00:33:3a
 `- 60:a4:23:ff:fe:37:a5:1d
```

On the Device

- Check the content of 'Serial 1'. By now, it should be connected if the Border Router and the device settings match and both are working as expected.



- Since it's now connected, the information strings you prepared are now visible, providing hints to the user.
  - Information on TFTP settings
    - `OTA DFU will download chunks of '<TFTP_DIRECTORY>/wisun_firmware.gbl' from fd00:6172:6d00::1/69`
  - Information on the OTA start command (to be executed on the Border Router)
    - `coap-client -m post -N -B 10 -t text coap://[fe80::62a4:23ff:fe37:a51d]:5683/ota/dfu -e "start"`
  - Information on the notification settings
    - `OTA DFU notifications enabled (every 10 chunks)`
    - `OTA DFU notifications will be POSTed to notification server coap://[fd00:6172:6d00::2]:5685/ota/dfu_notify`
  - Information on how to follow the OTA progress
    - `Follow OTA DFU progress (from notification server) using:`

- `coap-client -m get -N -B 1 -t text coap://[fd00:6172:6d00::2]:5685/ota/dfu_notify`

Store the Device's IPv6 GLOBAL Address

Copy the `GLOBAL` IPv6 address either from the **Addresses** `[GLOBAL :<IPv6>]` section, or from the above info on the "start" command. This is the device's IPv6 address you will need to use for OTA DFU as the `WISUN_NODE_IPV6_ADDR` value.

On the Border Router, set a command line variable to make it easier to repeat the OTA process for other devices:

`WISUN_NODE_IPV6_ADDR=fd00:6172:6d00:0:62a4:23ff:fe37:a51d` (in the example)

> TIP: You need to do this for each bash window you open.

Store the Border Router's IPv6 Address

Copy the `BORDER_ROUTER` IPv6 address from the **Addresses** `[BORDER_ROUTER :<IPv6>]` section.

On the Border Router, set a command line variable to make it easier to repeat the OTA process for other devices:

`WISUN_BR_IPV6_ADDR=fd00:6172:6d00:0:92fd:9fff:fe00:333a` (in the example)

> TIP: You need to do this for each bash window you open.

OTA DFU on the Linux Border Router

Copy the `.gbl` file to the TFTP Server Directory

- Check the TFTP server directory from its config file:

```
sudo cat /etc/default/tftpd-hpa
```

```
# /etc/default/tftpd-hpa

TFTP_USERNAME="tftp"
TFTP_DIRECTORY="/srv/tftp"
TFTP_ADDRESS=":69"
TFTP_OPTIONS="--secure"
```

- Copy the `.gbl` files in the `TFTP_DIRECTORY` folder:

```
sudo cp /tmp/*.gbl /srv/tftp/.
```

```
ls -l /srv/tftp/*.gbl
```

```
-rw-r--r-- 1 root root 629476 Jun 29 15:38 /srv/tftp/wisun_soc_empty_version_2.gbl
-rw-r--r-- 1 root root 544424 Jun 29 15:38 /srv/tftp/wisun_soc_empty_version_2_lz4.gbl
-rw-r--r-- 1 root root 385996 Jun 29 15:38 /srv/tftp/wisun_soc_empty_version_2_lzma.gbl
```

- Copy the `.gbl` file of your choice to the name set as `SL_WISUN_OTA_DFU_GBL_FILE` in `config/sl_wisun_ota_dfu_config.h`.
- First, start testing with the un-compressed file.

```
sudo cp /srv/tftp/wisun_soc_empty_version_2.gbl /srv/tftp/wisun_firmware.gbl
```

- Check the presence of the `.gbl` file:

```
ls -al /srv/tftp/*.gbl
```

```
-rw-r--r-- 1 root root 629476 Jun 29 15:41 /srv/tftp/wisun_firmware.gbl
-rw-r--r-- 1 root root 629476 Jun 29 15:38 /srv/tftp/wisun_soc_empty_version_2.gbl
-rw-r--r-- 1 root root 544424 Jun 29 15:38 /srv/tftp/wisun_soc_empty_version_2_lz4.gbl
-rw-r--r-- 1 root root 385996 Jun 29 15:38 /srv/tftp/wisun_soc_empty_version_2_lzma.gbl
```

See that the file sizes for `wisun_firmware.gbl` and `wisun_soc_empty_version_2.gbl` match, indicating that you're using the uncompressed file for the time being.

Check that you can Ping the Border Router

```
ping $WISUN_BR_IPV6_ADDR -c 1
```

```
PING fd00:6172:6d00:0:92fd:9fff:fe00:333a(fd00:6172:6d00:0:92fd:9fff:fe00:333a) 56 data bytes
64 bytes from fd00:6172:6d00:0:92fd:9fff:fe00:333a: icmp_seq=1 ttl=64 time=0.214 ms

--- fd00:6172:6d00:0:92fd:9fff:fe00:333a ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 0.214/0.214/0.214/0.000 ms
```

Successfully pinging the Border Router means that it's started and the IPv6 address that you retrieved from the device is correct.

Check that you can Ping the TFTP Server

```
ping fd00:6172:6d00::1
```

Successfully pinging the TFTP server means that it's IPv6 has been added to `tun0` and it is accessible from your devices.

Check that you can Ping the Notification Server

```
ping fd00:6172:6d00::2
```

Successfully pinging the notification server means that it's IPv6 has been added to `tun0` and it is accessible from your devices and from the Border Router.

- Your devices will use coap 'POST' messages to store their OTA status.
- You'll use coap 'GET' messages to check the devices status.
- Note that:
  - You're not directly asking the devices for status, because:
    - Doing so, you reduce the traffic on your Wi-SUN network.
    - If a device reboots after the upgrade, it will not be able to respond anymore to CoAP requests, while the notification server will be able to tell if OTA was successful, since the last notification message will show the last notification received from the device.

Check that you can Ping the Node

```
ping $WISUN_NODE_IPV6_ADDR -c 1
```

```
PING fd00:6172:6d00:0:62a4:23ff:fe37:a51d(fd00:6172:6d00:0:62a4:23ff:fe37:a51d) 56 data bytes
64 bytes from fd00:6172:6d00:0:62a4:23ff:fe37:a51d: icmp_seq=1 ttl=63 time=74.9 ms

--- fd00:6172:6d00:0:62a4:23ff:fe37:a51d ping statistics ---
1 packets transmitted, 1 received, 0% packet loss, time 0ms
rtt min/avg/max/mdev = 74.892/74.892/74.892/0.000 ms
```

Successfully pinging the device means that it's connected to the Wi-SUN network and is working as expected. You can now test OTA DFU.

Check that CoAP is Running on the Node

Send your `coap-client` GET method with `-v 6` to get some level of debug information.

```
coap-client -m get coap://[$WISUN_NODE_IPV6_ADDR]:5683/ota/dfu -v 6
```

> TIP: Here add `-v 6` to add verbosity and get more details on the CoAP command. Once you get used to the procedure, you may not want to keep using this method.

```
v:1 t:CON c:GET i:ded0 {} [ Uri-Path:ota, Uri-Path:dfu ]
v:1 t:ACK c:2.01 i:ded0 {} [ Content-Format:application/json ] :: '{\x0A"elapsed_t":"00:23:44",\x0A"downl_bytes":0,\x0A"flags":
"0×00000000",\x0A"fw_update_started": 0,\x0A"fw_downloaded": 0,\x0A"fw_verified": 0,\x0A"fw_set": 0,\x0A"fw_stopped":
0,\x0A"fw_download_error": 0,\x0A"fw_verify_error": 0,\x0A"fw_set_error": 0\x0A}\x0A'
{
"elapsed_t":"00:23:44",
"downl_bytes":0,
"flags": "0×00000000",
"fw_update_started": 0,
"fw_downloaded": 0,
"fw_verified": 0,
"fw_set": 0,
"fw_stopped": 0,
"fw_download_error": 0,
"fw_verify_error": 0,
"fw_set_error": 0
}
```

> TIP: The `"elapsed_t"` value is the time since connection to the Wi-SUN network, as long as OTA DFU hasn't been started.

On the device console, you see the CoAP 'Received packet' and 'Response packet'.

```
[CoAP-RHND-Service: Received packet]
{
  "token_len": 0,
  "coap_status": 0,
  "msg_code": 1,
  "msg_type": 0,
  "content_format": 4294967295,
  "msg_id": 57040,
  "payload_len": 0,
  "uri_path_len": 7,
  "token": "n/a",
  "uri_path": "ota/dfu",
  "payload": "n/a",
}
[CoAP-RHND-Service: Response packet]
{
  "token_len": 0,
  "coap_status": 0,
  "msg_code": 65,
  "msg_type": 32,
  "content_format": 50,
  "msg_id": 57040,
  "payload_len": 224,
  "uri_path_len": 0,
  "token": "n/a",
  "uri_path": "n/a",
  "payload":
{
"elapsed_t":"00:23:44",
"downl_bytes":0,
"flags": "0×00000000",
"fw_update_started": 0,
"fw_downloaded": 0,
"fw_verified": 0,
"fw_set": 0,
"fw_stopped": 0,
"fw_download_error": 0,
"fw_verify_error": 0,
"fw_set_error": 0
}

}
```


> TIP: You can check that the message index `i:ded0` in `v:1 t:CON c:GET i:ded0` matches the `"msg_id": 57040` in the Device's console (`57040 = 0xded0`).

Start OTA DFU for the Wi-SUN Device

At this point, you can trigger an OTA DFU for the device.

Checking `coap-client` Options

Use `coap-client --help` to get the help. Below is an abstract with the version info and options used.

```
coap-client --help
```

```
coap-client v4.1.2 -- a small CoAP implementation
(c) 2010-2015 Olaf Bergmann <bergmann@tzi.org>

usage: coap-client [-A type...] [-t type] [-b [num,]size] [-B seconds] [-e text]
            [-m method] [-N] [-o file] [-P addr[:port]] [-p port]
            [-s duration] [-O num,text] [-T string] [-v num] [-a addr] [-U] URI

      URI can be an absolute or relative coap URI,
      -B seconds      break operation after waiting given seconds
                      (default is 90)
      -e text         include text as payload (use percent-encoding for
                      non-ASCII characters)
      -m method       request method (get|put|post|delete), default is 'get'
      -N              send NON-confirmable message
      -v num          verbosity level (default: 3)
```

Starting Firmware Download using `coap-client`

```
coap-client -m post -N -B 3 -t text coap://[$WISUN_NODE_IPV6_ADDR]:5683/ota/dfu -e "start" -v 6
```

OTA DFU Starting on the Border Router

```
v:1 t:NON c:POST i:4186 {} [ Uri-Path:ota, Uri-Path:dfu, Content-Format:text/plain ] :: 'start'
v:1 t:NON c:2.01 i:0000 {} [ Content-Format:application/json ] :: '{\x0A"elapsed_t":"00:00:00",\x0A"downl_bytes":0,\x0A"flags":
"0×00000001",\x0A"fw_update_started": 1,\x0A"fw_downloaded": 0,\x0A"fw_verified": 0,\x0A"fw_set": 0,\x0A"fw_stopped":
0,\x0A"fw_download_error": 0,\x0A"fw_verify_error": 0,\x0A"fw_set_error": 0\x0A}\x0A'
{
"elapsed_t":"00:00:00",
"downl_bytes":0,
"flags": "0×00000001",
"fw_update_started": 1,
"fw_downloaded": 0,
"fw_verified": 0,
"fw_set": 0,
"fw_stopped": 0,
"fw_download_error": 0,
"fw_verify_error": 0,
"fw_set_error": 0
}
```

The `"fw_update_started": 1` flag is set!

OTA DFU Starting on the Device

```
[CoAP-RHND-Service: Received packet]
{
  "token_len": 0,
  "coap_status": 0,
  "msg_code": 2,
  "msg_type": 16,
  "content_format": 0,
  "msg_id": 16774,
  "payload_len": 5,
  "uri_path_len": 7,
  "token": "n/a",
  "uri_path": "ota/dfu",
  "payload": "start"}
[CoAP-RHND-Service: Response packet]
{
  "token_len": 0,
  "coap_status": 0,
  "msg_code": 65,
  "msg_type": 16,
  "content_format": 50,
  "msg_id": 0,
  "payload_len": 224,
  "uri_path_len": 0,
  "token": "n/a",
  "uri_path": "n/a",
  "payload":
{
"elapsed_t":"00:00:00",
"downl_bytes":0,
"flags": "0×00000001",
"fw_update_started": 1,
"fw_downloaded": 0,
"fw_verified": 0,
"fw_set": 0,
"fw_stopped": 0,
"fw_download_error": 0,
"fw_verify_error": 0,
"fw_set_error": 0
}

}
[wisun-btl] (0) Storage info: version: 196608, capabilities: 0, storageType: 0, numStorageSlots: 1
[wisun-btl] (7) Firmware upgrade started
[wisun-btl] (11) notify: coap:///[fd00:6172:6d00::2]:5685/ota/dfu_notify
[wisun-btl] (18) TFTP init done
[wisun-btl] (21) notify: coap:///[fd00:6172:6d00::2]:5685/ota/dfu_notify
[wisun-btl] (27) TFTP download started: tftp:///[fd00:6172:6d00::1]:69/wisun_firmware.gbl
[wisun-btl] (442) download: received chunk 1, offset: 0×00000000
[wisun-btl] (866) download: received chunk 2, offset: 0×00000200
[wisun-btl] (1090) download: received chunk 3, offset: 0×00000400
[wisun-btl] (1315) download: received chunk 4, offset: 0×00000600
[wisun-btl] (1739) download: received chunk 5, offset: 0×00000800
[wisun-btl] (1964) download: received chunk 6, offset: 0×00000a00
[wisun-btl] (2189) download: received chunk 7, offset: 0×00000c00
[wisun-btl] (2414) download: received chunk 8, offset: 0×00000e00
[wisun-btl] (2920) download: received chunk 9, offset: 0×00001000
[wisun-btl] (3128) notify: coap:///[fd00:6172:6d00::2]:5685/ota/dfu_notify
[wisun-btl] (3153) download: received chunk 10, offset: 0×00001200
[wisun-btl] (3377) download: received chunk 11, offset: 0×00001400
[wisun-btl] (3802) download: received chunk 12, offset: 0×00001600
```

- The `"ota/dfu"` `"start"` command has been received
- The OTA DFU process started

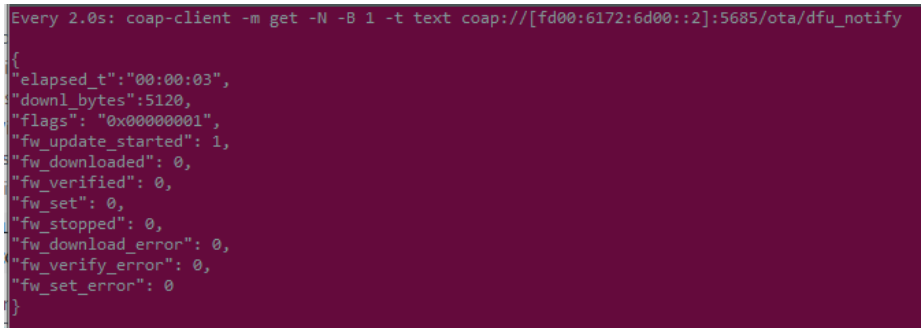
- The first chunks of `wisun_firmware.gbl` have been received
- The first two notifications have been sent to the notification server's IPV6

Monitoring OTA DFU Progress on Notification Server

```
coap-client -m get -N -B 1 -t text coap://[fd00:6172:6d00::2]:5685/ota/dfu_notify
```

> TIP: A more convenient way to follow the process is to use `watch` as follows:

`watch --interval 2 coap-client -m get -N -B 1 -t text coap://[fd00:6172:6d00::2]:5685/ota/dfu_notify`



- The `"elapsed_t"` and `"downl_bytes"` will increase during image download
- The `"fw_update_started"` flag is `1` from the start
- The `"fw_downloaded"` flag will be `1` when download is complete (several minutes at 50 kbps/1 Hop)
- The `"fw_verified"` flag will be `1` when verification is complete (this can take an additional ~20 sec)
- The `"fw_set"` flag will be `1` when reboot is about to be triggered (using the NVIC_SystemReset() low-level function)
  - Note that the device will send its last notification with `"fw_set"` then reboot, so whatever you retrieve from the notification server is this message until you start a new upgrade.

Monitoring OTA DFU Progress on Device (intrusive!)

You can also check the progress using a CoAP GET method on the device, but because this is using the Wi-SUN network, it has an impact on the OTA DFU duration if used frequently. The device will also stop responding for a while while rebooting and reconnecting, then respond with `"fw_update_started": 0` once reconnected.

```
coap-client -m get coap://[$WISUN_NODE_IPV6_ADDR]:5683/ota/dfu
```

```
{
"elapsed_t":"00:01:59",
"downl_bytes":204800,
"flags": "0×00000001",
"fw_update_started": 1,
"fw_downloaded": 0,
"fw_verified": 0,
"fw_set": 0,
"fw_stopped": 0,
"fw_download_error": 0,
"fw_verify_error": 0,
"fw_set_error": 0
}
```

> TIP: Using `watch --interval 10 coap-client -m get coap://[$WISUN_NODE_IPV6_ADDR]:5683/ota/dfu` , you can get the request sent to the device every 10 seconds. This is a convenient way to follow the upgrade process, but it's

> even more intrusive than not using `watch`.

Checking OTA DFU Success on the Device

In the device console, check the startup message, looking for '**Version 2**'.

```
Wi-SUN Empty application
Version 2. Compiled on Jul  6 2023 at 13:27:37
OTA DFU is supported
[Join state: Acquire PAN Config (3)]

[Connecting to "Linux_BZ_3_8"]

[Join state: Configure Routing (4)]
[Join state: Operational (5)]

Addresses:
[GLOBAL      : fd00:6172:6d00:0:b635:22ff:fe98:2527]
[LINK_LOCAL  : fe80::b635:22ff:fe98:2527]
[BORDER_ROUTER : fd00:6172:6d00:0:92fd:9fff:fe00:333a]
[25 s]
```

OTA DFU Cheat Sheet

Below are all items used for OTA DFU that need to match between various parts of the setup. Check these in case of issues.

- The TFTP server
- The `.gbl` file
- The CoAP Notification server
- The OTA DFU component
- The `coap-server` command
- The `coap-client` commands
- The bootloader

Below are the relationships between the settings:

Configuration	Setting on Linux Host	Setting on Node project
TFTP Server IPv6 address	Command line: `sudo ip address add fd00:6172:6d00::2/64 dev tun0`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR "fd00:6172:6d00::2"`
TFTP Server Port	TFTP Server configuration file: `/etc/default/tftpd-hpa` `TFTP_ADDRESS=":69"`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_TFTP_PORT "69U"`
TFTP Directory	TFTP Server configuration file: `/etc/default/tftpd-hpa` `TFTP_DIRECTORY="/srv/tftp"`	Location of `.gbl` files: `/srv/tftp/`
`.gbl` file name	Command line: `ls -l /srv/tftp/*.gbl`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_GBL_FILE "wisun_firmware.gbl"`
OTA DFU Notifications	Command line: `coap-server...` started	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED 1U`

Configuration	Setting on Linux Host	Setting on Node project
Notification server IPv6 address	Command line: `sudo ip -6 address add fd00:6172:6d00::2/64 dev tun0`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR "fd00:6172:6d00::2"`
OTA DFU notify URI path	Command line: `coap-client -m get ... coap://[<node_ipv6>]:<port>/ota/dfu_notify`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_NOTIFY_URI_PATH "/ota/dfu_notify"`
CoAP Notification server IPv6	Command line: `coap-server -A fd00:6172:6d00::2 ...`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR "fd00:6172:6d00::2"`
CoAP Notification server UDP port	Command line: `coap-server ... -p 5685 ...`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_NOTIFY_PORT 5685U`
CoAP Notification server dynamic resources count	Command line: `coap-server ... -d 10` (dynamic resources)	Number of devices concurrently performing OTA DFU
OTA DFU control port	Command line: `coap-client -m post ... coap://[<node_ipv6>]:5683...`	CoAP Configuration file: `config/sl_wisun_coap_config.h` `SL_WISUN_COAP_RESOURCE_HND_SERVICE_PORT 5683U`
OTA DFU control URI path	Command line: `coap-client -m post ... coap://[<node_ipv6>]:<port>/ota/dfu ...`	OTA DFU Configuration file: `config/sl_wisun_ota_dfu_config.h` `SL_WISUN_OTA_DFU_URI_PATH "/ota/dfu"`
OTA DFU start command	Command line: `coap-client -m post ... -e "start"`	OTA DFU Source file: `<GSDK>\app\wisun\component\ota_dfu\sl_wisun_ota_dfu.c` `SL_WISUN_OTA_DFU_START_PAYLOAD_STR "start"`
OTA DFU stop command	Command line: `coap-client -m post ... -e "stop"`	OTA DFU Source file: `<GSDK>\app\wisun\component\ota_dfu\sl_wisun_ota_dfu.c` `SL_WISUN_OTA_DFU_STOP_PAYLOAD_STR "stop"`
OTA DFU boot command	Command line: `coap-client -m post ... -e "boot"`	OTA DFU Source file: `<GSDK>\app\wisun\component\ota_dfu\sl_wisun_ota_dfu.c` (not implemented in GSDK 4.3.0, uses 'auto-reboot')

Below are other items that need to be selected correctly:

Configuration	Compression component	project `.gbl` file creation
no compression	Bootloader Compression: `none`	`commander gbl create --app <projectname>.s37 <projectname>.gbl`
lz4 compression	Bootloader Component: `GBL Compression (LZ4)`	`commander gbl create --app <projectname>.s37 <projectname>.gbl --compress lz4`
lzma compression	Bootloader Component: `GBL Compression (LZMA)`	`commander gbl create --app <projectname>.s37 <projectname>.gbl --compress lzma`

Configuration	Part	Bootloader Binary
Bootloader binary for Series 1	xG12	`<projectname>-combined.s37`
Bootloader binary for Series 2	xG25 or xG28	`<projectname>.s37`

> NOTE: An application with OTA DFU support will crash during OTA if no OTA-capable bootloader is flashed to the device, because the application expects certain low-level functions to be available in the bootloader. This is a common pitfall when customers experiment with OTA DFU and start flashing their OTA DFU application to

> multiple devices: if they forget to also flash an OTA-capable bootloader they end up having unexpected crashes while their first test device works fine.

Application configuration	Bootloader configuration	Expected behavior
Application with OTA DFU	Bootloader without OTA DFU	Crashes
Application with OTA DFU	Bootloader with OTA DFU	Okay
Application without OTA DFU	Bootloader without OTA DFU	Okay
Application without OTA DFU	Bootloader with OTA DFU	Okay

Overview

Wi-SUN Node

The contents in this section provide information about developing Wi-SUN Node application. Contents include:

- Wi-SUN Developer's Guide (PDF): Covers the (Wi-SUN) stack architecture, application development flow, steps to configure the application Wi-SUN radio settings and advanced debug features.
- Wi-SUN Configurator: The Wi-SUN Configurator provides an interface to the Wi-SUN application's main settings through three panels: Application, Security, and Radio. For some examples, the Wi-Sun Configurator only displays the Radio panel. These examples do not have the application and security infrastructure.
- FAN 1.0 Node Certification: Describes how to install the Wi-SUN FAN Certification component and how to change connection settings.
- Wi-SUN Limited Function Nodes (LFN): Describes the Wi-SUN LFN architecture, supported hardware, power management, configuration, and parenting.

Wi-SUN Configurator

Wi-SUN Configurator

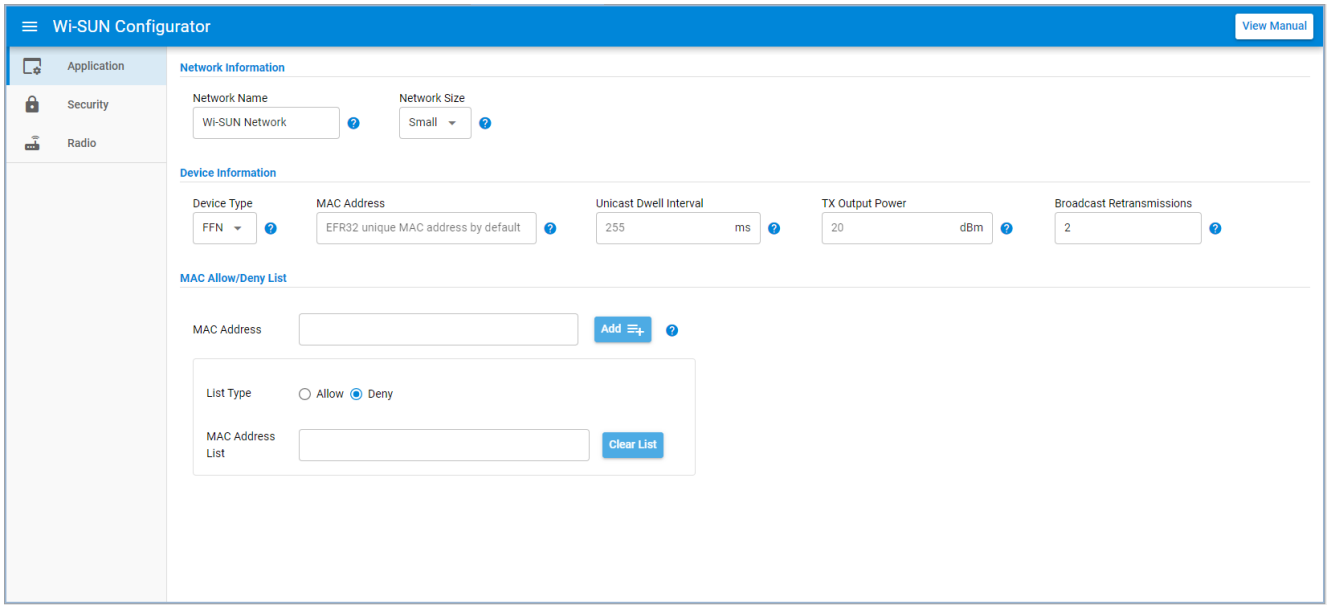
When creating a new Wi-SUN project, a Wi-SUN Configurator is added to it by default. The Wi-SUN Configurator provides an interface to the Wi-SUN application's main settings through three panels: Application, Security, and Radio. For some examples, the Wi-Sun Configurator only displays the Radio panel. These examples do not have the application and security infrastructure.

The Wi-SUN Configurator tab can be displayed by opening **/config/wisun/wisun_settings.wisunconf**.

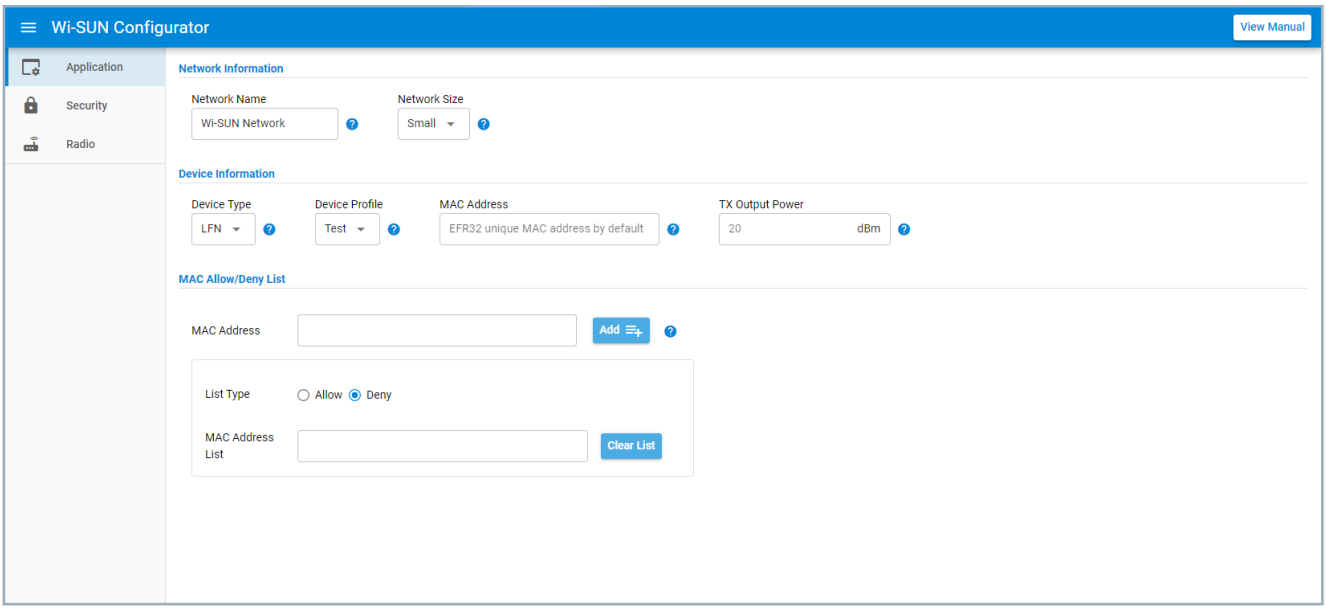
Application Panel

The Application panel exposes multiple Wi-SUN stack settings associated with the application, such as:

- The network name the device will try to connect to
- The network size setting
- The device's TX output power
- The unicast dwell interval



The device type dropdown allows you to set the device to be either an FFN or an LFN. Setting the device type to LFN exposes the device profile dropdown, which adjusts the LFN internal behavior for better performance and battery life.



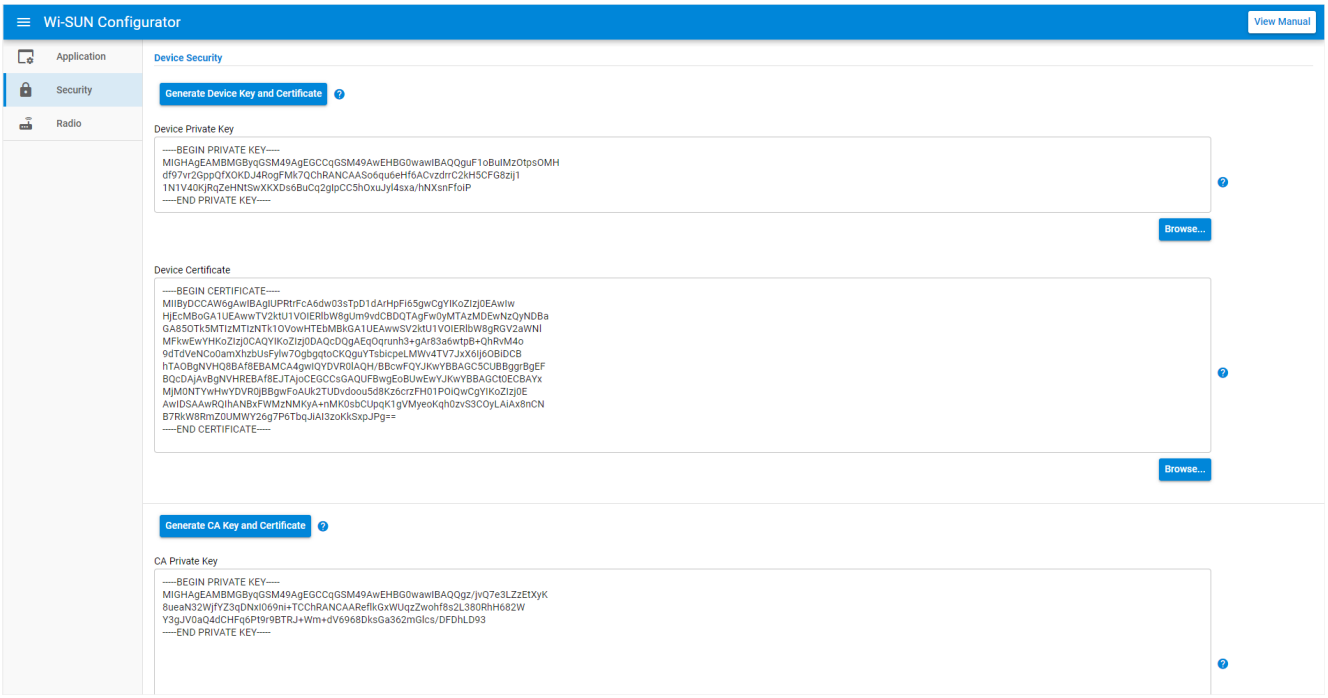
The MAC address filtering feature provides a way to force a topology on a Wi-SUN network. It is a simpler alternative than spacing out the Wi-SUN devices or tweaking the Tx power levels to achieve the desired topology. The device interacts only with other Wi-SUN devices part or not (depending on the allow/deny option) of the list.

Security Panel

The Security panel displays the private key and certificates used by the device to authenticate itself when connecting to a Wi-SUN network. By default, it uses the Silicon Labs demonstration samples. They can be modified to use a distinct certificate infrastructure aligned with the border router or authenticator certificate.

The **Generate Device Key and Certificate** button helps generate a new device key and certificate different from those used by default with the Silicon Labs Wi-SUN sample applications. This button is disabled in case of invalid CA Key or Certificate. CA Key and Certificate can also be generated using the button **Generate CA Key and Certificate**.

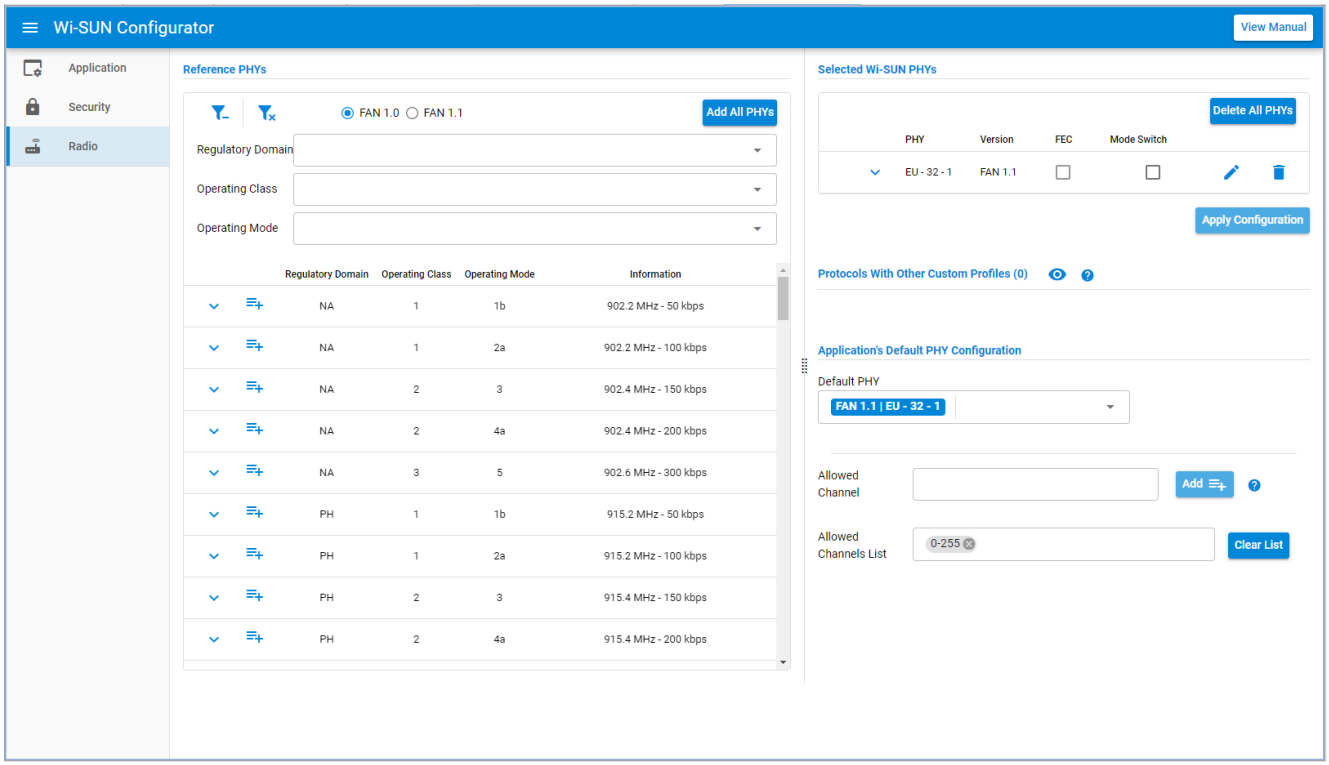
Generating new device key and certificate imply changing the default key and certificate on the Border router with new ones generated with the same CA Key and Certificate used to generate the device key and certificate.



Radio Panel

The Radio panel is an interface to configure the radio profiles included in the application. It provides a user interface to access any specified Wi-SUN FAN 1.0 PHY or FAN 1.1 PHY. A radio button allows the user to choose between the FAN 1.0 or FAN 1.1 context.

The complete list can be filtered to help you find the right PHY configuration. An application can embed several PHYs from different regions. The PHY used by the stack is defined by the `sl_wisun_join()` API call.



The Application's Default PHY input defines the PHY that the application starts with. The default value depends on the EFR32 radio board used to create the project. You can open the dropdown and select another default PHY.

Every selected Wi-SUN PHY can be edited in the Radio Configurator by clicking the pen icon. This action opens the Radio Configurator user interface on the selected Wi-SUN PHY. Moreover, non Wi-SUN FAN PHY are listed under **Other Custom Profile** for information.

The radio panel also embeds a section to modify the list of the device allowed channel. By default, the device operates in all the PHY's channels. Modifying the list does not affect the asynchronous frames. The changes are only applied on unicast and broadcast frames for the border router.

The channel mask will not affect asynchronous frames, as it is only applied on unicast traffic. For broadcast frames, the routers follow the border router's configuration.

FAN 1.0 Node Certification

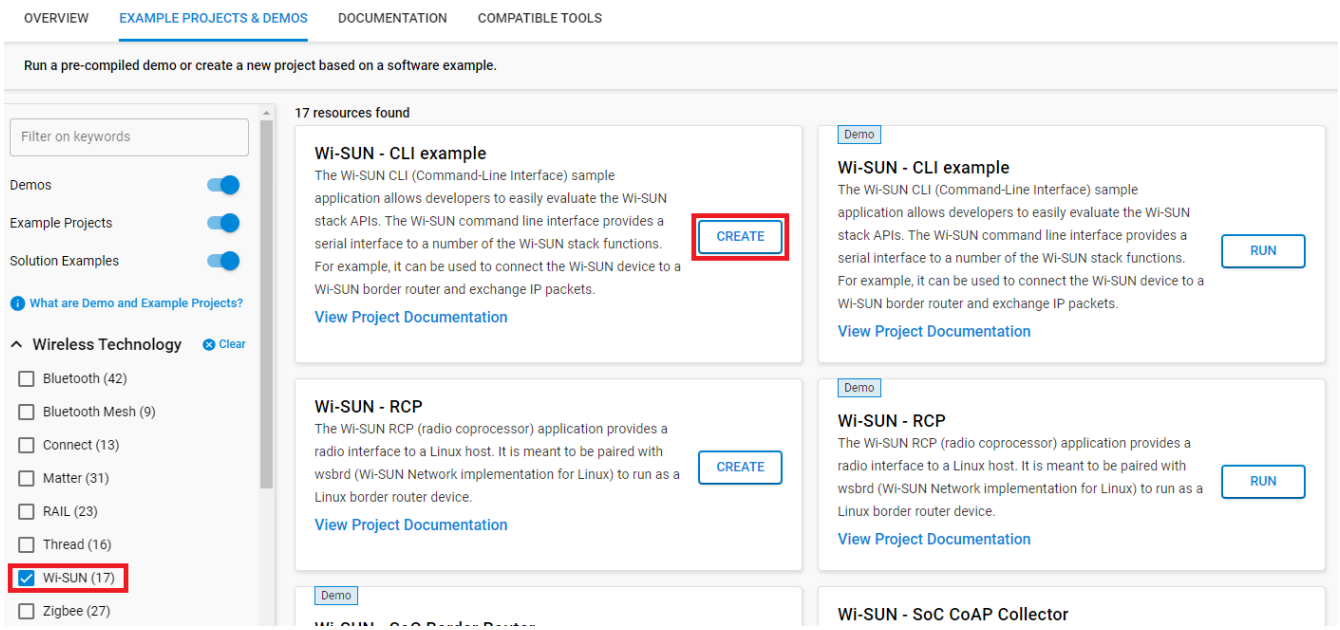
Wi-SUN FAN 1.0 Node Certification

The Silicon Labs Wi-SUN stack provides a component through Simplicity Studio 5 to help our customers certify their Wi-SUN FAN 1.0 routers. The **Wi-SUN FAN Certification** component can only be used with the **Wi-SUN - CLI example**, which sets up the WiSUN Alliance certificates with the certification configuration (Network name, Regulatory domain, Network size, etc).

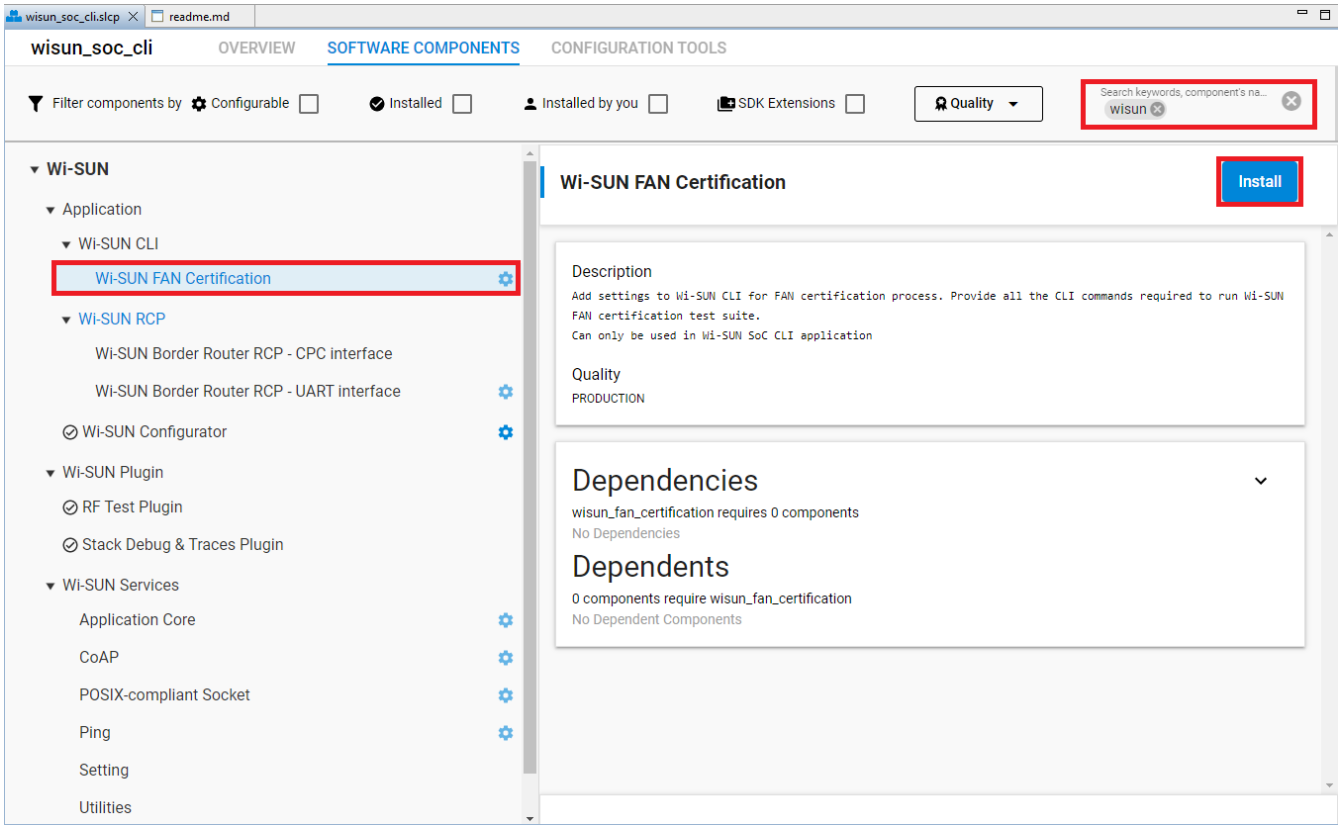
Installing Wi-SUN FAN Certification component

Follow these steps to add the **Wi-SUN FAN Certification** component to your application:

1. Open Simplicity Studio 5 and choose your board in **Debug Adapter**.
2. In the **Launcher** perspective, click **EXAMPLE AND PROJECT**.
3. Filter by *Wi-SUN* under **Wireless Technology** and create the **Wi-SUN - CLI example** app.



4. After creating the project, open *wisun_soc_cli.slcp* and click **Software Component**.
5. Enter *wisun* in the search bar to filter the components and install the **Wi-SUN FAN Certification** component.



6. Build the project and flash it to the board.

Changing the connection settings

The Wi-SUN Conformance Tests specification defines four required group tests to pass the **Wi-SUN FAN 1.0 Certification**, and describes the required settings for each test group.

- 6.2 Startup Phase
- 6.3 Normal Operations
- 6.4 Security
- 6.5 Error Handling

The **Wi-SUN FAN Certification** component sets by default the network name, regulatory domain, operating class, operating mode, certificate chain, and allowed channels to match those required by the '6.2 Startup phase' test group.

```
> wisun get wisun.network_name
wisun.network_name = "WiSUN PAN"
> wisun get wisun.regulatory_domain
wisun.regulatory_domain = NA (1)
> wisun get wisun.operating_class
wisun.operating_class = 1
> wisun get wisun.operating_mode
wisun.operating_mode = 0x1b
> wisun get wisun.certificate_chain
wisun.certificate_chain = certif (1)
> wisun get wisun.allowed_channels
wisun.allowed_channels = "0"
```

For each test group, change the PHY settings, certificate chain, and the allowed channels. The next sections explain how to change those settings.

PHY Configuration

As described in the Wi-SUN Conformance Tests specification, two PHY settings are common between the test groups. The following table provides the PHY settings of each group.

	6.2	6.3 / 6.4 /6.5
Regulatory domain	NA	NA
Operating class	1	2
Operating mode	0x1b	0x3

By default, the settings are set for the group '6.2 Startup phase'. Use the following commands to change the settings for the other test groups.

```
wisun set wisun.operating_class 2
wisun set wisun.operating_mode 0x3
```

Wi-SUN Certificates

The **Wi-SUN FAN Certification** component adds the possibility to change the certificates used by the application from the command line. This allows you to change the certificates without having to reflash a new binary on the DUT, especially when running the test '6.4.2 SEC-TLS-TERMINATE-2', which requires certificates that are different than the Wi-SUN Alliance Test certificates.

By default, the component installs Wi-SUN Alliance Test certificates with the wisun.certification_chain set to 'certif'.

```
> wisun get wisun.certificate_chain
wisun.certificate_chain = certif (1)
```

To change the certificate chain to Silicon Labs certificates, use the following command.

```
wisun set wisun.certificate_chain silabs
```

Allowed Channels

The Startup phase tests require a fixed channel. By default, the allowed channel is 0 but can be changed to align with Test Bed Units. Use the following command to change the allowed channel to channel 10, for example.

```
wisun set wisun.allowed_channels 10
```

The Normal Operations, Security, and Error Handling tests use Direct Hash Channel Function and do not require channel filtering. To reset channel filtering and allow all the channels in the channel plan, use the following command.

```
wisun set wisun.allowed_channels 0-255
```

Restart the DUT

After making your changes and before starting a new test, follow this sequence to reset the DUT.

```
# Disconnect from the Wi-SUN network
> wisun disconnect
# Clear the credential cache
> wisun clear_credential_cache
```

Then use the following command to join the network.

```
# Connect to the Wi-SUN network using FAN1.0 settings
> wisun join_fan10
```

Wi-SUN Limited Function Nodes (LFN)

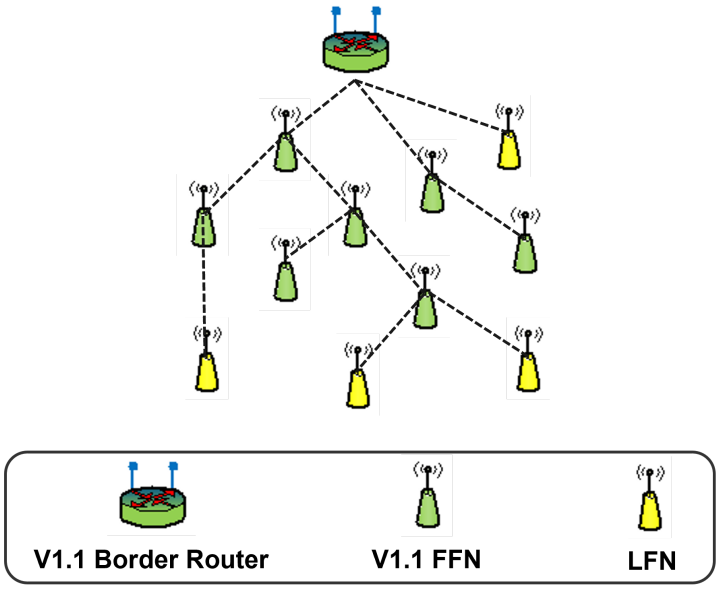
Introduction

The Wi-SUN FAN stack supports Limited Function Node (LFN) functionalities specified in the Wi-SUN FAN Technical Profile Specification version 1.1v06. The Limited Function Nodes as described in the specification must support low power operations:

- The MAC must allow for LFN battery life of 20 years.
- The MAC must support a less than 2 minute response for LFNs.
- Multicast operations must minimize LFN power consumption.

Architecture

LFNs provide minimum capabilities like discovering/joining a PAN and sending/receiving IPv6 packets. LFNs also implement the same communication stack as a Full Function Node (FFN) with a limited listening schedule to conserve power. But it can only operate within a PAN rooted at a FAN 1.1 Border Router (LFN is not part of the FAN 1.0 specification) as a child of a FAN 1.1 Router and not a parent to any other node. See the following figure.



LFN Characteristics

An LFN employs different mechanisms than those used by an FFN to communicate, do routing, and secure the association with the border router.

- An LFN operates as an RPL Unaware Leaf (RUL) node, which differs from an FFN in that an LFN does not implement RPL routing and relies on a parent RPL Router to provide routing capability.
- LFNs employ a sampled listening technique rather than the continuous listening performed by an FFN.
- LFN's security associations with the Border Router are substantially longer lived than those of an FFN because FFNs and LFNs use separate sets of group keys (GTK / LGTK, respectively), with longer lifetimes for the LFN group keys.

Those characteristics help an LFN meet the required low power capabilities described in the Wi-SUN FAN Specification.

Supported Hardware

LFN feature is supported on all the Wi-SUN capable devices listed at Wi-SUN Wireless SoCs. However, the power optimization capabilities are only supported on the **EFR32FG28** (BRD4401C) and **EFR32ZG28** (BRD4400C) as the other Wi-SUN Wireless SoCs have low energy consumption limitations in EM2 DeepSleep energy mode.

LFN in Silicon Labs Wi-SUN Stack

The Wi-SUN Stack implements all the necessary mechanisms for an LFN to be able to join a Wi-SUN Network. Also, it supports Multicast and the power optimization capabilities.

On the latest GSDK, LFNs are supported on the following sample apps by default:

- **Wi-SUN – LFN CLI example**
- **Wi-SUN – CLI example**
- **Wi-SUN – SoC Meter**
- **Wi-SUN – SoC Coap Meter**
- **Wi-SUN – SoC Network Measurement**
- **Wi-SUN – SoC Ping**
- **Wi-SUN – SoC TCP Server**
- **Wi-SUN – SoC UDP Server**
- **Wi-SUN – SoC Empty**

LFN Power Management

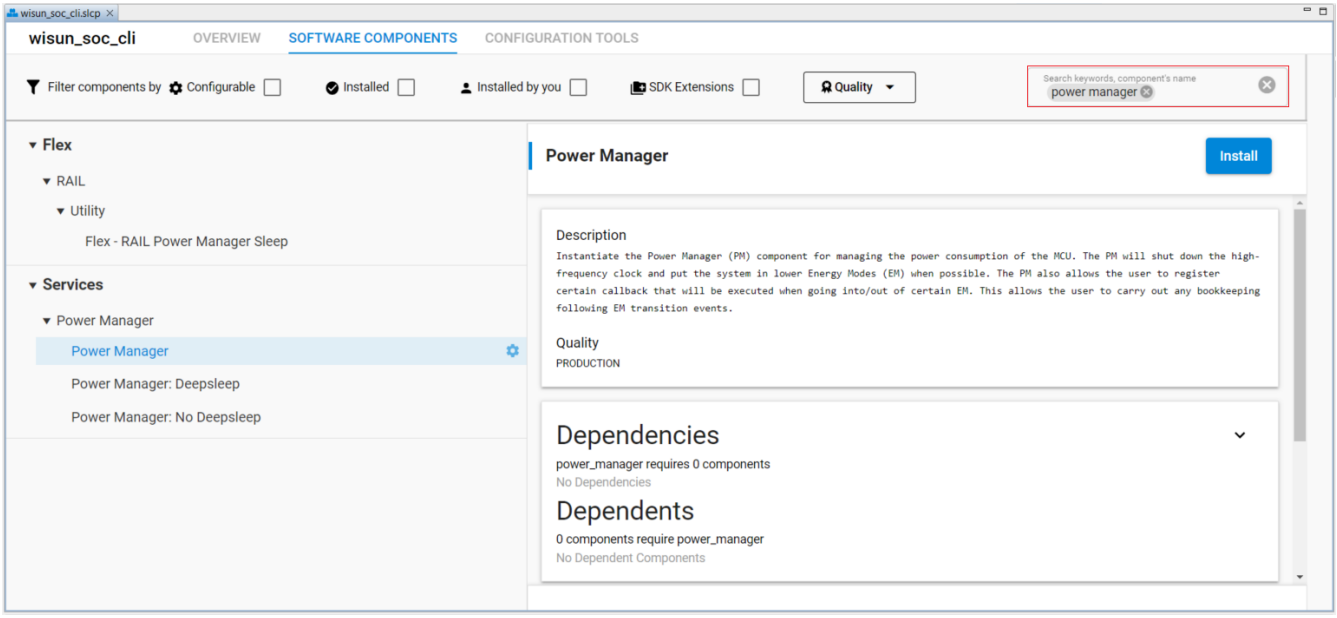
The Wi-SUN stack supports LFN power management capabilities in GSDK 4.3.1 and later. This feature allow the LFNs to go into sleep in EM2 mode, with an average consumption of **5.23 µA** with the **EFR32FG28** (BRD4401C) and **EFR32ZG28** (BRD4400C).

> Note: The EFR32FG28 version BRD4401B and the EFR32ZG28 version BRD4400B have hardware limitations that keep them from entering sleep mode EM2.

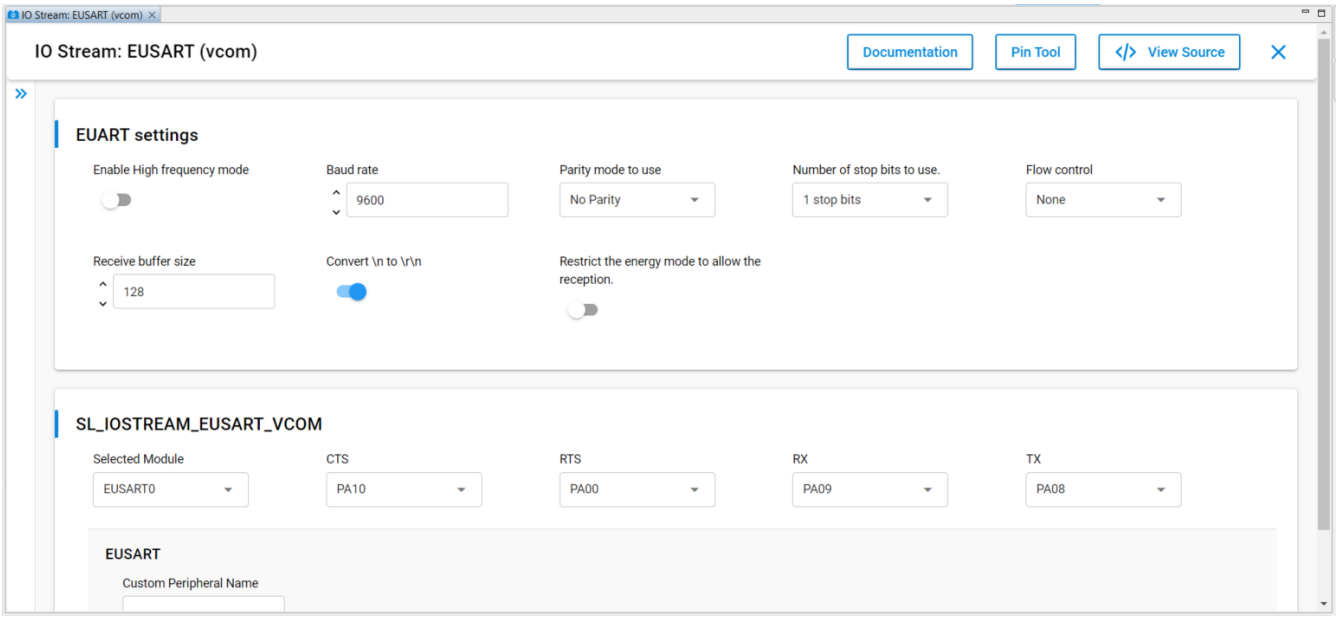
This power optimization level comes with some challenges, especially the CLI UART clock that we use in our sample applications. For the **Wi-SUN – LFN CLI example**, the project configures the CLI UART to use the Low-Frequency clock to be able to use the CLI in a low speed of 9600 Bauds without preventing the LFN from entering the sleep mode EM2.

For the rest of the examples, this configuration has to be done manually on the software component perspective after creating the project. Follow the next steps to disable the UART high frequency clock and enable the power management capabilities:

1. Open the slcp file and click the **SOFTWARE COMPONENTS**. Then use the search bar to look for the **Power Manager** component and click **Install**.



2. Use the search bar to look for **IO Stream: EUSART** and click the on the configuration icon in front of *vcom*:
   - Switch off **Enable High Frequency mode**.
   - Set the Baud rate to 9600.
   - Switch off **Restrict the energy mode to allow the reception** if enabled.



3. Search for the **Micrium OS Kernel** component and click the configuration icon:
   - Switch off **Enable Round-Robin scheduling**.
   - Switch off **Enable statistics gathering task**.
   - Switch off **Enable task profiling instrumentation**.

After configuring the project to enter the sleep mode EM2, configure the WPK to use 9600 vcom baud rate speed following the next steps:

1. On the Debug Adapter perspective, right-click on your board and click **Launch Console**.
2. Click the **Admin** tab of the Console panel.
3. Enter the command `serial vcom config speed 9600` .

> Note: This is a persistent configuration and it might be necessary to restore the previous value if the same WPK is used with any other CLI application. To do that, enter the command `serial vcom config speed 115200` .

LFN Configuration Using the CLI

The **Wi-SUN - SoC CLI example** supports changing the device type and the device profile using the command line interface. By default, the device type is configured as FFN. To change the device type, use the following command that allows setting the device type to either **LFN** or **FFN**:

```
wisun set wisun.device_type LFN
```

> Note: For this command to take effect after a join either as an FFN or an LFN, a board reset is mandatory.

The device profile can also be set through the command line, using the following command that allows choosing between three profiles:

- **Test:** Used for testing the LFNs performance.
- **Balanced:** Balance between LFNs performance and battery life.
- **Eco:** Profile for high battery life performance.

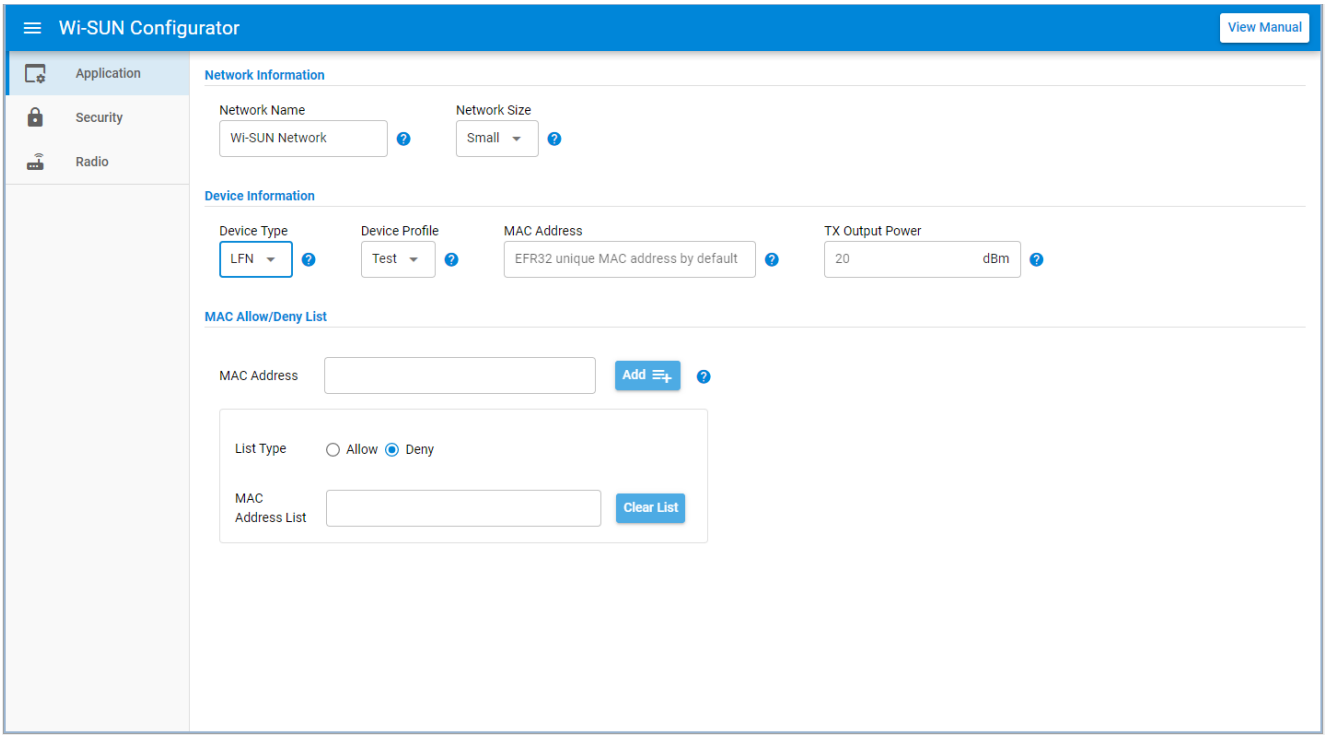
```
wisun set wisun.device_profile balanced
```

> Note: The profiles defined above focus on defining different time slots and intervals for the LFN parameters to optimize performance and battery life. These three predefined modes are just for reference, and it is up to the customer to fine tune the parameters to adapt the LFN behavior to the application use case. The three profiles are defined under protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h.

Enabling LFN Using Wi-SUN Configurator

Except for the **Wi-SUN - LFN CLI example** and **Wi-SUN - SoC CLI example**, the other sample applications supporting LFN, by default, use **Wi-SUN Configurator** to enable LFN device type.

Once the project is created, the device type is set by default to FFN. To change the device type to LFN, go to the **Application** panel on the **Wi-SUN Configurator** and, under the **Device information** section, change the **Device Type** of the node to **LFN**.



LFN Device Profiles

When LFN is selected in the **Device Type** dropdown, a dropdown named **Device Profile** is exposed on the Wi-SUN Configurator Application's panel to allow choosing from three device profiles: **Test**, **Balanced**, and **Eco**.

LFN Parenting

An LFN operates as an RPL Unaware Leaf (RUL) node, which means that an LFN does not implement RPL routing and relies on its parent Border Router or FFN to provide routing capability. In other words, LFNs won't be able to join the Network in the following cases:

- If the Border router doesn't support LFN parenting.
- If the Border router supports LFN parenting and the LFN parent does not support LFN parenting.
- If the Border router doesn't support LFN direct parenting and there is no router that supports LFN parenting which the LFN can join.

Routers LFN Parenting

On the GSDK 4.3 and later, LFN parenting is supported by the sample apps listed above by default.

The number of supported LFNs by each FFN is 10 by default on all the applications supporting LFN parenting. On **Wi-SUN - CLI example**, this number can be changed using the following command:

```
wisun set_lfn_support [Max number of LFNs]
```

The maximum number of LFN nodes that can be supported by an FFN is 10.

Enabling LFN Parenting

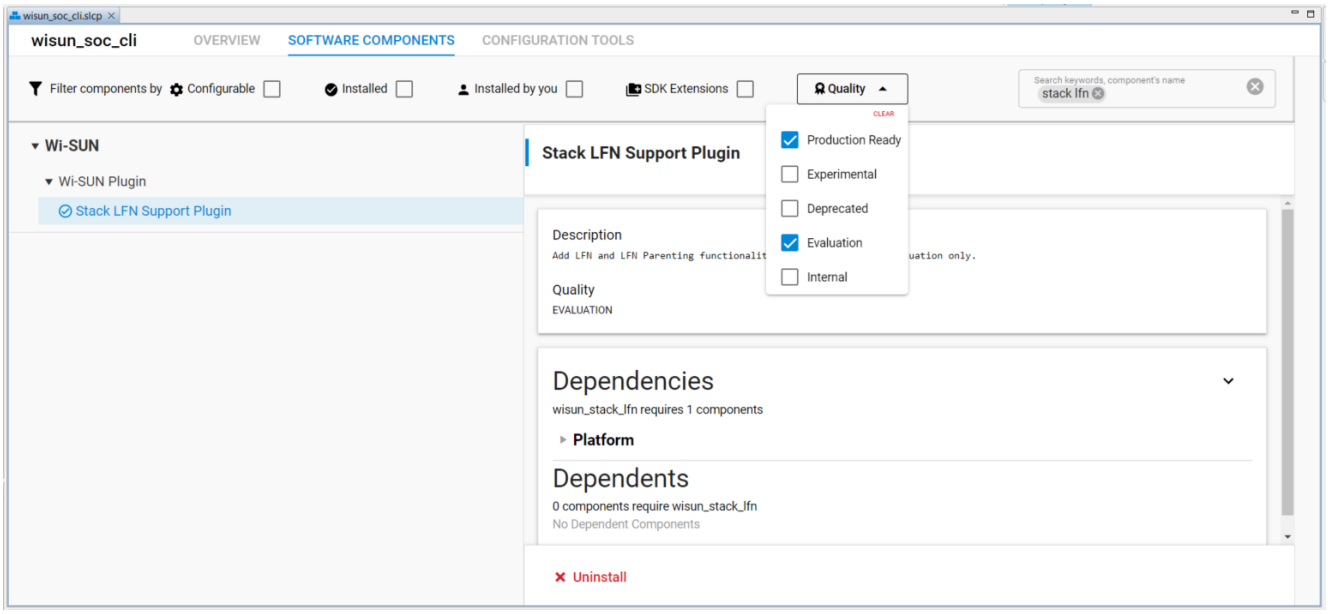
On the sample applications listed below, LFN is not supported by default.

- Wi-SUN – SoC CoAP Collector
- Wi-SUN – SoC Collector
- Wi-SUN – SoC TCP Client

- Wi-SUN – SoC UDP Client

To allow the LFNs to join the devices running those applications as FFNs, it is mandatory to enable LFN parenting.

LFN parenting can be enabled on those applications by installing the component **Stack LFN Support Plugin**. Make sure to check the *Evaluation* box on the quality filter before searching for the component.



Border Routers LFN Parenting

Wi-SUN - SoC Border Router

LFN parenting is supported by default on **Wi-SUN - SoC Border router**, and it is disabled along side the LFN support on the PAN by setting ***lfn_support_pan*** to ***0*** using the following command:

```
wisun set_lfn_support [lfn_limit] [lfn_support_pan]
```

The ***lfn_limit*** argument sets the number of locally supported LFNs by the Border Router SoC. By default, it is set to 10 and the maximum number of LFNs that the Border Router can support is 10.

For a better LFN device battery performance and synchronization with the Border Router SoC, the LFN device profiles have to be set also on the Border Router side using the following command:

```
wisun set wisun.lfn_profile [device profile]
```

Wi-SUN Border Router Linux

The wisun-br-linux version 1.6 and later supports LFN direct parenting by default. And it has no option to set a maximum number of LFN devices that can connect directly to the border router.

LFN Support in The PAN Network

In version 1.7, a new configuration option was added to the configuration file wsbrd.conf named `enable_lfn` to allow enabling or disabling LFN devices support in the PAN Network, hence disabling the LFN parenting.

LFN Device Profiles

Unlike the **Wi-SUN - SoC Border Router** that has predefined LFN device profiles, the **Wi-SUN Border Router Linux** introduces two configuration options, `lfn_broadcast_interval` and `lfn_broadcast_sync_period` in wsbrd.conf to allow

synchronizing the LFN broadcast listening windows and the number of LFN broadcast intervals to optimize the power consumption by the LFN devices.

> Note: The wisun-br-linux supports LFN parenting by default with a limitation of not supporting direct parenting of LFNs on the wisun-br-linux versions 1.5 to 1.5.4. To connect an LFN to wisun-br-linux using one of those versions, an FFN connected to the network and supporting LFN parenting is mandatory.

Overview

Platform Resources

When you develop in the Silicon Labs GSDK, 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. As well as the security features available, this section describes Wi-SUN-specific security design considerations.

Overview

Bootloading Wi-SUN Applications

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.

- 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.

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. As well as the security features available, this section describes Wi-SUN-specific security design considerations.

- 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.
- Wi-SUN FAN Security Concepts and Design Considerations: Discusses security in terms of authentication, the Wi-SUN Router, and the Wi-SUN Border Router.
- 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.

Security Concepts and Design Considerations

Wi-SUN FAN Security Concepts and Design Considerations

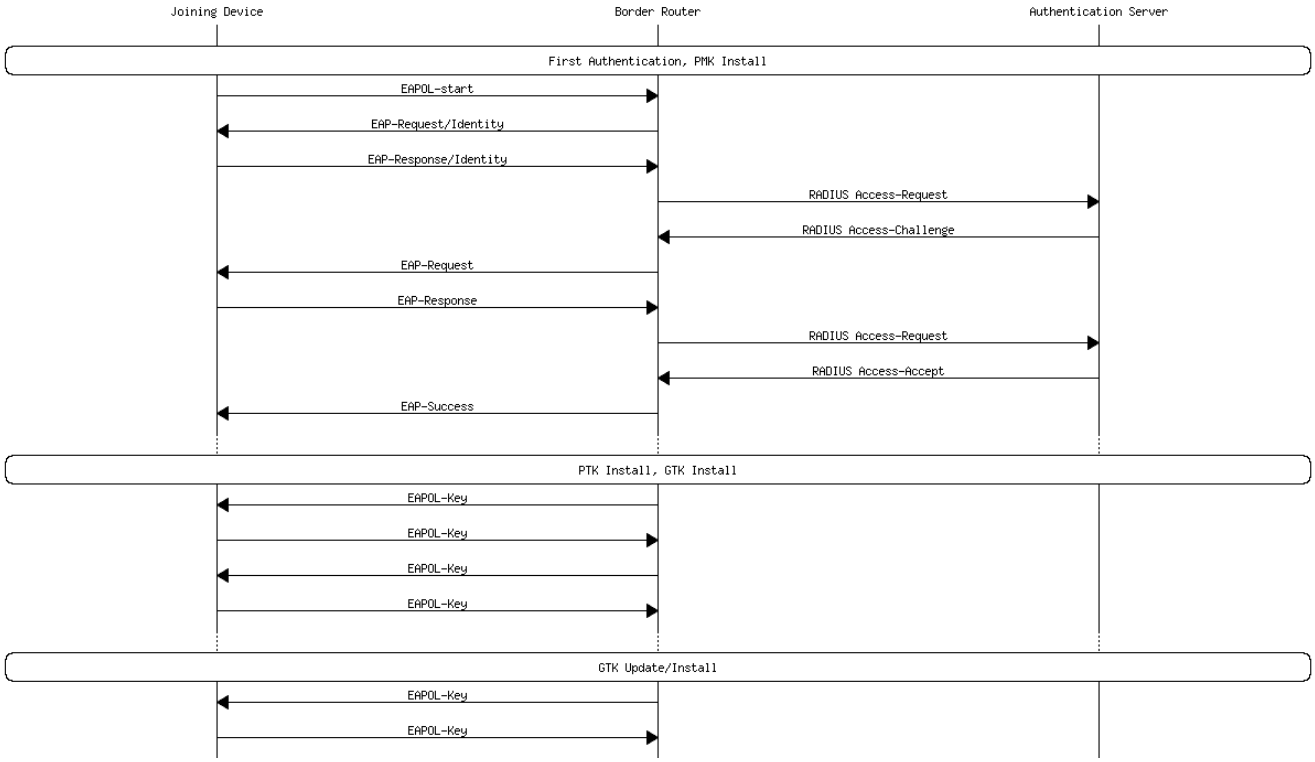
Wi-SUN FAN security mechanisms are built on widely-used industry standards. Access control is based on IEEE 802.1X and IEEE 802.11i concepts, providing mutual authentication and establishment of a security association between the joining node and the Wi-SUN Border Router. Frame security uses AES-CCM* from IEEE 802.15.4, providing both data confidentiality and data authenticity.

This page discusses security in terms of:

Authentication

Wi-SUN authentication is done using the Extensible Authentication Protocol - Transport Layer Security (EAP-TLS) protocol over Extensible Authentication Protocol over LAN (EAPOL). The authentication results in a Pairwise Master Key (PMK), a unique key shared between the border router and the device.

The PMK is used in the 4-way-handshake procedure with the border router. During the authentication, the border router delivers up to four Group Transient Keys (GTKs) to the devices. GTKs are shared between all connected devices in the network. The flow is described in the diagram below.



In the Wi-SUN FAN context, authentication is considered to be an “expensive” operation. This is reflected in the default lifetime of various authentication keys. Silicon Labs' Wi-SUN FAN stack stores the authentication keys and the TX frame counter in non-volatile storage in order to make reconnection faster after a reboot or a loss of connection.

Key	Default Lifetime	Refresh Procedure
Pairwise Master Key (PMK)	4 months	EAP-TLS
Pairwise Transient Key (PTK)	2 months	4-way-handshake
Group Transient Key (GTK)	1 month	4-way-handshake or 2-way-handshake

Wi-SUN Router

Each Wi-SUN device has a unique X.509 certificate (NIST EC P-256) signed by a Certification Authority (CA). A Wi-SUN device must store at least two certificates:

- The device certificate: Used to authenticate the device to an authentication server. The authentication server may run on the border router or on the backhaul network.
- The CA root certificate: Used by the device to verify the authentication server.

Device Certificate

Each Wi-SUN node has a secure identity based on a unique per-device X.509 certificate and its corresponding private key, using the Secure Device Identifier (DevID) concept from IEEE 802.1AR. Wi-SUN device certificates must adhere to the requirements in the table below.

X.509 w/ v3 Extensions

Fields and Extensions	Value
version	v3
serialNumber	an unique serial number

Fields and Extensions	Value
signature	ecdsa-with-SHA256
issuer	copied from issuer's subject field
notBefore	issuing time and date in UTC (GeneralizedTime)
notAfter	99991232235959Z (GeneralizedTime)
subjectPublicKeyInfo	id-ecPublicKey, namedCurve secp256r1
signatureAlgorithm	ecdsa-with-SHA256
keyUsage	digitalSignature, keyAgreement
extendedKeyUsage	clientAuth, id-kp-wisun-fan-device
authorityIdentifier	only the keyIdentifier field
subjectAltName	id-on-hardwareModuleName

id-kp-wisun-fan-device

In addition to clientAuth, all Wi-SUN device certificates must contain the id-kp-wisun-fan-device object identifier.

```
id-kp-wisun-fan-device OBJECT IDENTIFIER ::= { iso(1) identified-organization(3) dod(6) internet(1) private(4) enterprise(1) Wi-SUN (45605) FieldAreaNetwork(1) }
```

id-on-hardwareModuleName

All Wi-SUN device certificates must contain one and only one alternative name of type OtherName of type id-on-hardwareModuleName. The sequence is specified in RFC4108.

```
HardwareModuleName ::= SEQUENCE { hwType OBJECT IDENTIFIER, hwSerialNum OCTET STRING }
```

`hwType` is an object identifier that, at a minimum, identifies the manufacturer's enterprise number (IANA) but may optionally be subtyped to contain manufacturer-specific information, such as the device model.

`hwSerialNum` is the serial number of the hardware module. No particular structure is imposed on the serial number. However, the combination of the hwType and hwSerialNum uniquely identifies the hardware module.

Device Private Key

In addition to a plaintext key in the buffer, the stack also accepts a PSA key reference to the device private key. When using the key reference, the PSA key attributes must be set according to the following table.

Attribute	Value
psa_set_key_usage_flags	PSA_KEY_USAGE_SIGN_HASH
psa_set_key_type	PSA_KEY_TYPE_ECC_KEY_PAIR(PSA_ECC_FAMILY_SECP_R1)
psa_set_key_algorithm	PSA_ALG_ECDSA(PSA_ALG_SHA_256)

Credential Cache

The device maintains a credential cache for the previously connected PAN. If the same PAN is joined again, the device will use the stored credentials to bypass the authentication phase.

NOTE: The credential cache can be cleared before the join attempt by using the API function `sl_wisun_clear_credential_cache()` but it should be used with care. Clearing the cache on the joining node may prevent it from re-joining the same PAN. This is due to AES-CCM* replay protection security mechanism.

Wi-SUN Border Router

In the authentication process, the Wi-SUN border router has the Authenticator role. Depending of the architecture, the authentication server can be hosted on the border router or on another device connected over IP. In the latter case, the authentication server must run RADIUS (Remote Authentication Dial-In User Service). The border router/authenticator acts as a RADIUS client while the external device acts as a RADIUS server.

Create your Own Demo Wi-SUN Certificates

As a prerequisite, install OpenSSL on your machine.

How to View a Certificate

```
openssl x509 -in <certificate pem> -text
```

How to Create the Wi-SUN Demo Certificates

First, retrieve the `openssl-wisun.conf` file in this Community post.

1. Create the certificate database, used to track created certificates.

```
touch certdb.txt
```

1. Generate a Certificate Signing Request (CSR) for the root CA.

```
openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout wisun_root_ca_key.pem -out wisun_root_ca_req.pem -config openssl-wisun.conf
```

1. Self-sign the root CA.

```
openssl ca -selfsign -rand_serial -keyfile wisun_root_ca_key.pem -in wisun_root_ca_req.pem -out wisun_root_ca_cert.pem -notext -extensions v3_root_ca -config openssl-wisun.conf -subj "/CN=Wi-SUN Demo Root CA"
```

1. Generate a CSR for the intermediate 1 CA.

```
openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout wisun_intermediate1_ca_key.pem -out wisun_intermediate1_ca_req.pem -config openssl-wisun.conf
```

1. Sign the intermediate 1 CA with the root CA.

```
openssl ca -rand_serial -cert wisun_root_ca_cert.pem -keyfile wisun_root_ca_key.pem -in wisun_intermediate1_ca_req.pem -out wisun_intermediate1_ca_cert.pem -notext -extensions v3_ca1 -config openssl-wisun.conf -subj "/CN=Wi-SUN Demo Intermediate 1 CA"
```

1. Generate a CSR for the intermediate 2 CA.

```
openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout wisun_intermediate2_ca_key.pem -out wisun_intermediate2_ca_req.pem -config openssl-wisun.conf
```

1. Sign the intermediate 2 CA with the intermediate 1 CA.

```
openssl ca -rand_serial -cert wisun_intermediate1_ca_cert.pem -keyfile wisun_intermediate1_ca_key.pem -in wisun_intermediate2_ca_req.pem -out wisun_intermediate2_ca_cert.pem -notext -extensions v3_ca2 -config openssl-wisun.conf -subj "/CN=Wi-SUN Demo Intermediate 2 CA"
```

1. Generate a CSR for the border router.

```
openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout wisun_br_key.pem -out wisun_br_req.pem -config openssl-wisun.conf
```

1. Sign the border router certificate with the desired CA.

```
openssl ca -rand_serial -cert wisun_root_ca_cert.pem -keyfile wisun_root_ca_key.pem -in wisun_br_req.pem -out wisun_br_cert.pem -notext -extensions v3_br -config openssl-wisun.conf -subj "/CN=Wi-SUN Demo Border Router"
```

1. Generate a CSR for the device.

```
openssl req -new -newkey ec -pkeyopt ec_paramgen_curve:prime256v1 -keyout wisun_device_key.pem -out wisun_device_req.pem -config openssl-wisun.conf
```

1. Sign the device certificate with the desired CA.

```
openssl ca -rand_serial -cert wisun_root_ca_cert.pem -keyfile wisun_root_ca_key.pem -in wisun_device_req.pem -out wisun_device_cert.pem -notext -extensions v3_device -config openssl-wisun.conf -subj "/CN=Wi-SUN Demo Device"
```

Note that the "openssl-wisun.conf" file contains the Silicon Labs enterprise number. You should replace it with your own (cf. https://www.iana.org/assignments/enterprise-numbers/enterprise-numbers).

Overview

Wi-SUN Border Router

A Wi-SUN Border Router connects a Wi-SUN network to other IP-based networks, such as Ethernet®. It also manages the Wi-SUN mesh network connected. The Border Router provides services for devices within the Wi-SUN network, including routing services for off-network operations. Silicon Labs provides a Border Router binary demonstration running on a standalone EFR32 and a production-grade Wi-SUN Linux Border Router.

This section provides additional information about using and configuring a Wi-SUN Border Router.

- Wi-SUN Network Configuration: Introduces the different Wi-SUN Border Router solutions maintained by Silicon Labs. Provides installation instructions for a Wi-SUN Linux Border Router. Describes the Linux Border Router software architecture. Includes the steps to establish a connection between a Wi-SUN network and an external IP network.
- Wi-SUN IP Communication: This section provides guidelines to ping a Wi-SUN device from the Raspberry Pi running wsbrd. These guidelines assume that wsbrd has been configured and is running on the Raspberry Pi.
- Linux Border Router (GitHub): Contains source code and instructions for `wsbrd`, one part of the Linux border router solution.
- Border Router Graphical User Interface: Presents the Wi-SUN Border Router Dashboard, that can be used to configure the Wi-SUN Linux Border Router and visualize the Wi-SUN Network.

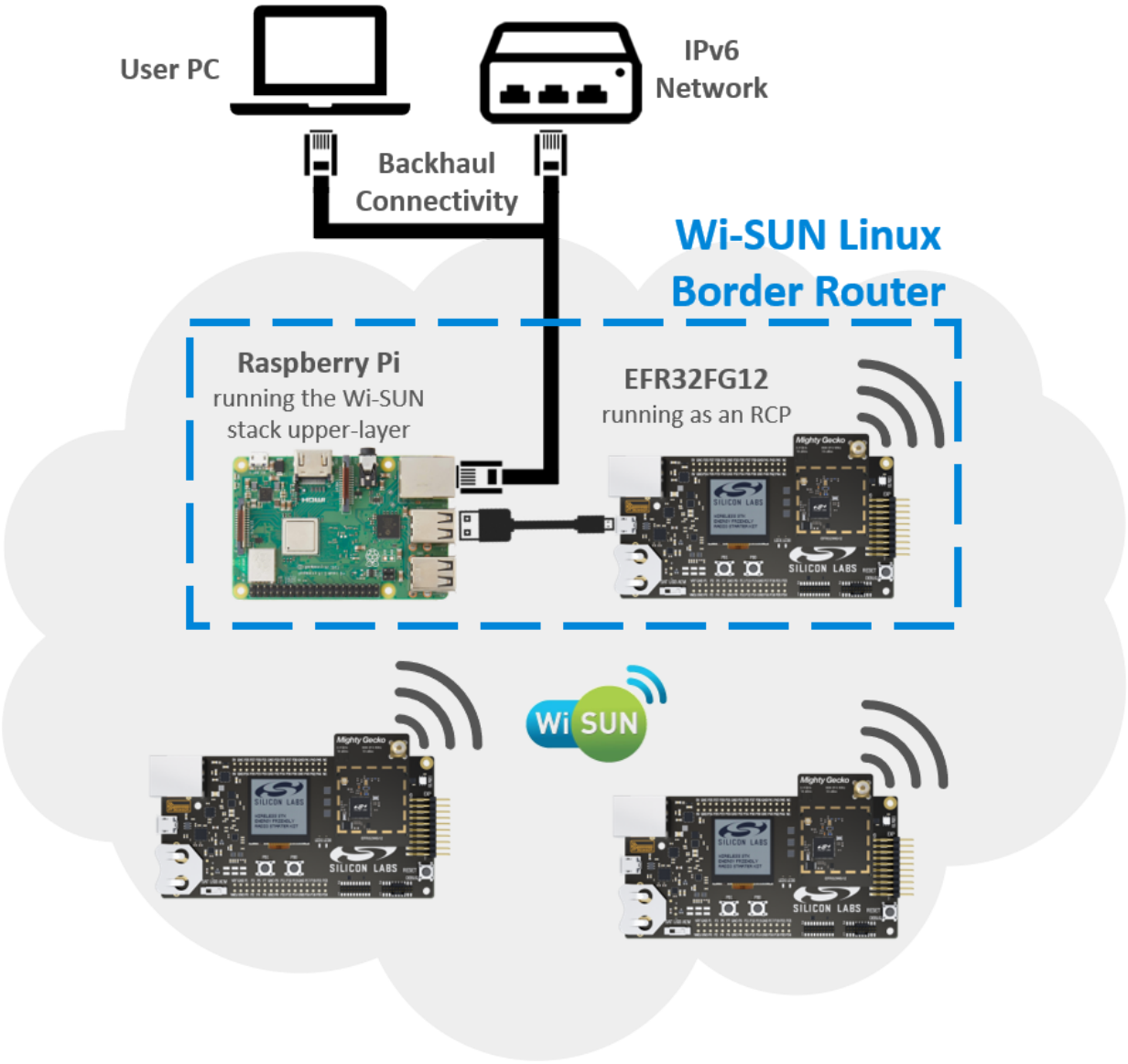
Network Configuration

Network Configuration Introduction

In a Wi-SUN network, the border router oversees the network management (authentication, routing, and so on) and provides internet connectivity to the Wi-SUN devices part of the network.

Because of the standalone RCP connectivity features, the Linux border router solution offers an easy-to-use and scalable solution to address various Wi-SUN network deployments. This implementation relies on an external Silicon Labs EFR32 device flashed with a dedicated Wi-SUN RCP firmware. As a result, the EFR32 is connected to the host with a serial link. The EFR32 runs the time constrained low-level layers while the Linux host handles the memory-intensive computing of the Wi-SUN stack upper-layers.

As an accessible and well-spread Linux platform, the Raspberry Pi is used as a default platform for the solution demonstration (which can also run on a different Linux host), and will run the Wi-SUN Border Router Linux daemon, which is responsible for running the Wi-SUN protocol high-level layers. The demonstration communications and connections are described in the following figure.



As an alternative, Silicon Labs also provides a border router demonstration running on a standalone EFR32. It provides a quicker way to evaluate the solution but does not scale into production.

Hardware Requirements

To create a Wi-SUN Network with a Wi-SUN Linux border router and one Wi-SUN Node, you need the following hardware:

- A Raspberry Pi 3 Model B+ or above with an Internet connection (another Linux host should also work)
- An SD card (4 GB or more) and SD card slot/dongle
- 2 WTSK/WPK boards
- 2 EFR32 radio boards with matching RF bands from the Wi-SUN capable devices listed in this link.

Software Requirements

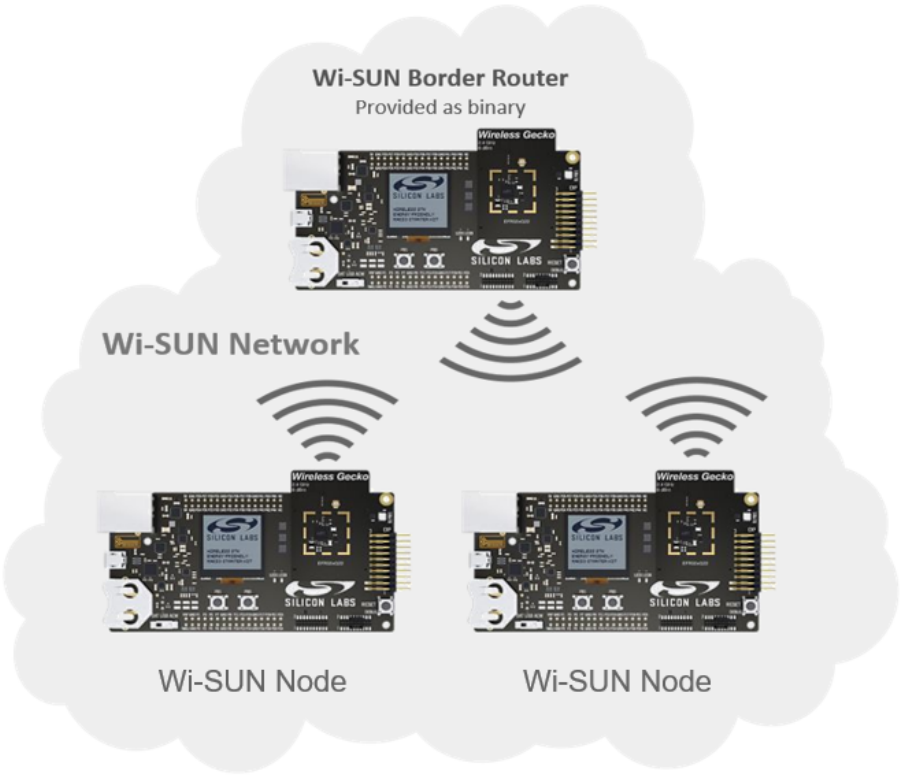
You need the following software to complete the steps in this application note, you will need the following software:

- Simplicity Studio 5
- Raspberry Pi imager
- wsbrd source code repository

Wi-SUN SoC Border Router

Wi-SUN SoC Border Router

The Silicon Labs Wi-SUN SoC Border Router demonstration provides a Wi-SUN Border Router implementation running entirely on the EFR32. It provides an easy and quick means to evaluate the Silicon Labs Wi-SUN stack solution without deploying an expensive and cumbersome production-grade Wi-SUN Border Router. A CLI (Command-Line Interface) is exposed to facilitate the configuration.

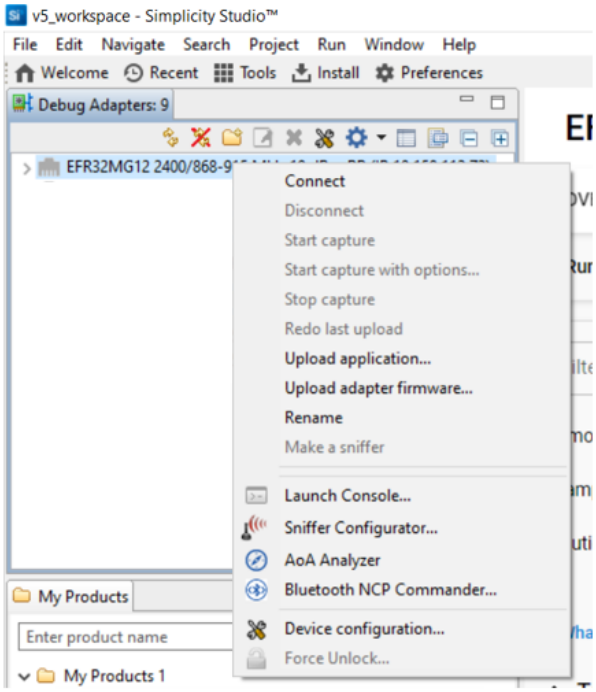


Getting Started with the Solution

The Wi-SUN SoC Border Router Demo creates a Wi-SUN network that the other Wi-SUN nodes can join.

To get started with the demonstration, follow these steps:

1. In the Debug Adapters view, select the device to be used as the Border Router.
2. Navigate to the **EXAMPLE PROJECTS & DEMOS** tab and turn off the Example Projects filter.
3. Click **RUN** next to the **Wi-SUN – SoC Border Router** project.
4. In the Debug Adapter view, right click your chosen device, and click on **Launch Console**.



5. Start the Border Router with default FAN 1.1 PHY using the following command:

```
> wisun start_fan11
```

Or with default FAN 1.0 PHY using the following command:

```
> wisun start_fan10
```

Wi-SUN - SoC Border Router Solution Limitations

The Wi-SUN Border Router demonstration is delivered only in a binary format. The implementation does not scale for a production-grade Border Router maintaining several thousand Wi-SUN nodes.

The Wi-SUN Border Router demonstration is required to use the other Wi-SUN sample applications. The Wi-SUN Border Router creates a Wi-SUN network that the Wi-SUN nodes can join. When part of the same network, the Wi-SUN nodes can exchange IP packets.

Wi-SUN SoC Border Router Configuration

The application provides a command-line interface to control basic configurations. To see the available commands, entre the following command in the console:

```
> wisun help
```

The list of available commands is output on the console with the associated help. Wi-SUN SoC Border Router sample app supports all the PHYs described in Wi-SUN PHY Specification Revision 1V09 and above.

Wi-SUN Linux Border Router

Wi-SUN Linux Border Router

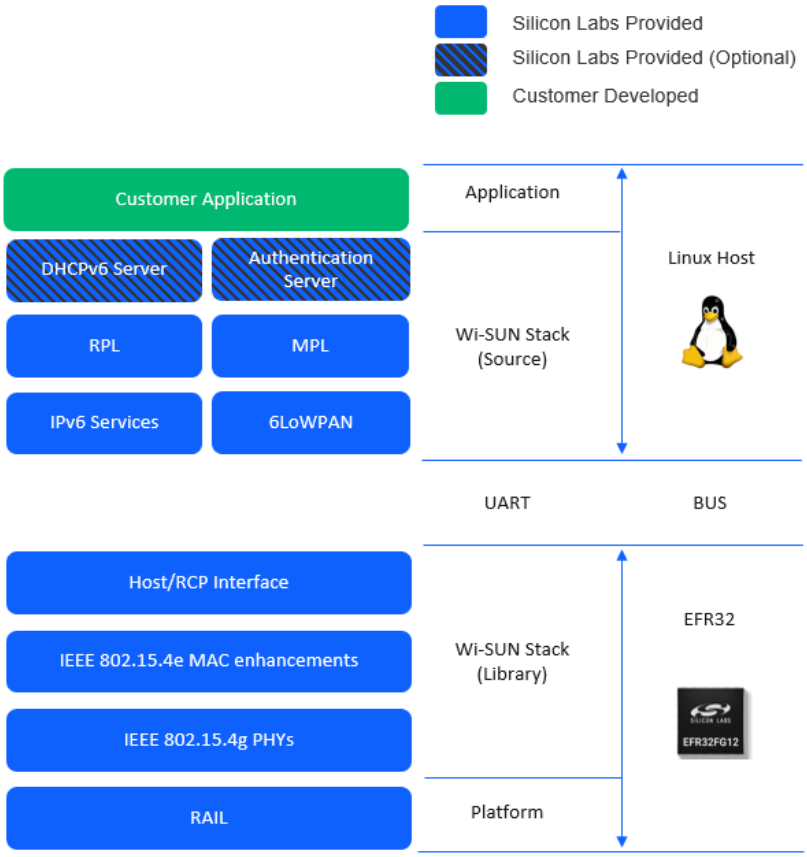
The Silicon Labs Wi-SUN Linux Border Router solution “wsbrd” provides a daemon that is responsible for running the Wi-SUN protocol high-level layers. It is paired with an RF device (RCP) handling the low-level layers and radio activities. The Silicon Labs RCP devices currently supported are listed under Wi-SUN Wireless SoCs.

Software Architecture

The Silicon Labs Linux border router solution has two distinct software components:

- wsbrd: the solution running on a Linux host. It supports high-level Wi-SUN FAN protocols such as 6LoWPAN, RPL, and MPL. The solution is available in source code on GitHub: https://github.com/SiliconLabs/wisun-br-linux
- Wi-SUN RCP: the solution running on an EFR32. It provides a low-level Wi-SUN implementation meant to be paired with a Linux host running wsbrd. Silicon Labs provides the Wi-SUN RCP application in the Gecko SDK and Simplicity Studio 5.

The two solutions exchange messages using a serial link between the Linux host and the EFR32. UART is the currently supported protocol. The solution relies on Linux services for the IPv6 communication with external IP devices. The complete architecture of the solution is provided in the following figure.



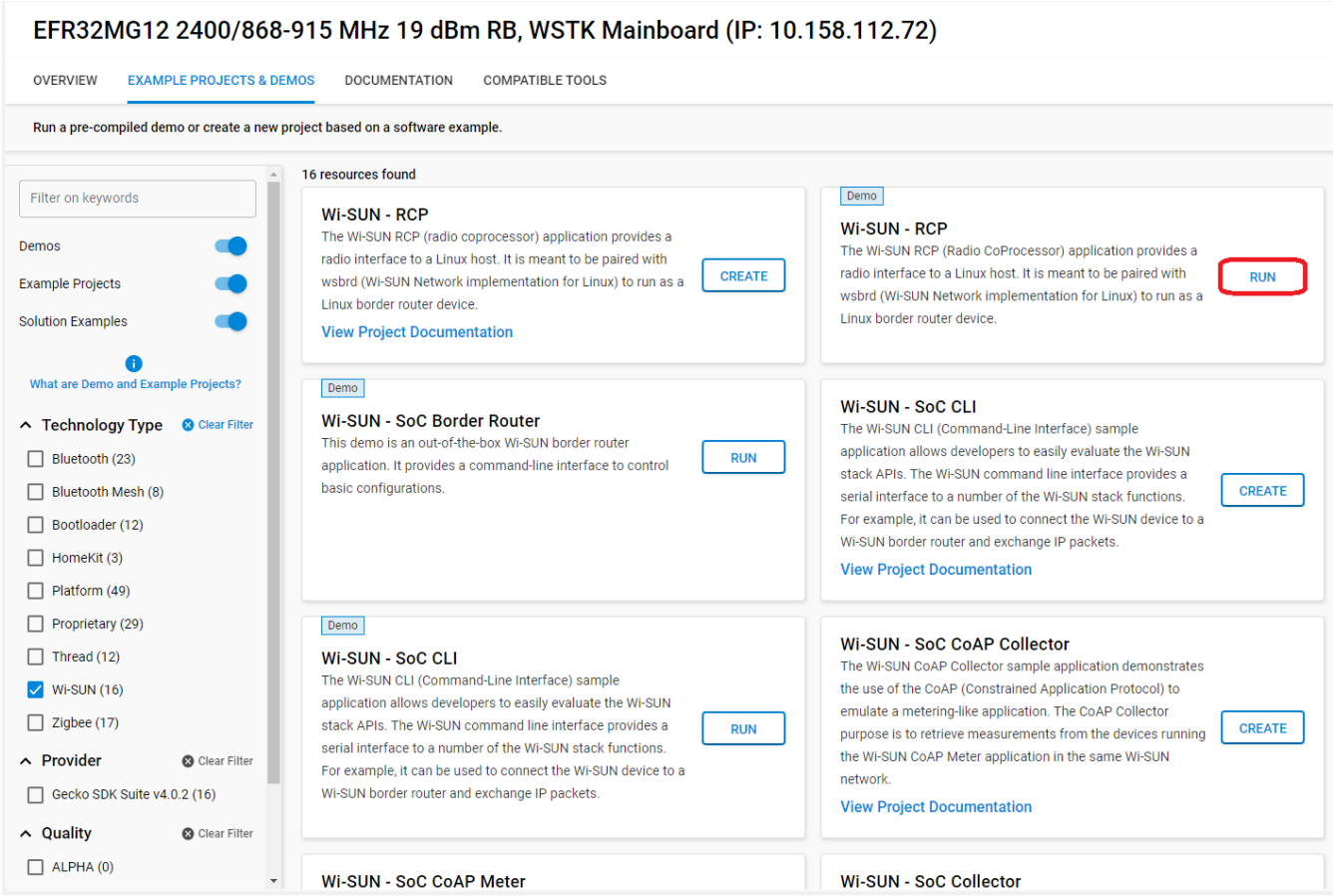
Getting Started with the Solution

To use the Wi-SUN Border Router Linux install wsbrd on a Raspberry Pi (Silicon Labs recommends this platform but you can use another Linux host if desired) and use the solution as it would be in production. This method allows you to access the Wi-SUN devices from the Linux host (Wi-SUN border router and the devices connected to it). However, there is no automatic IPv6 configuration to allow another IP device (other than the Linux host) to access the Wi-SUN network. You must configure the Linux host and the IP routing to match his external IP network configuration.

Flashing the Wi-SUN RCP

A mandatory step to evaluate the Silicon Labs Wi-SUN Linux border router solution is to flash the Wi-SUN RCP image on an EFR32. The first possibility is to use the image or project delivered through Simplicity Studio 5. Create the **Wi-SUN – RCP** project, compile, and flash it in your mainboard. To do so, complete the following steps:

1. In the **Debug Adapters** view, select the device to be used as the Wi-SUN RCP.
2. Navigate to the **EXAMPLE PROJECTS & DEMOS** tab and turn off the **Example Projects** filter.
3. Click **RUN** next to the **Wi-SUN – RCP** project.



wsbrd

To install and run wsbrd on your hardware:

1. Connect the EFR32 flashed with the RCP image to the Raspberry Pi through USB.
2. Download the wsbrd software on your Linux device.
3. Follow the guidelines provided in the repository readme (Wi-SUN Linux Border Router repository).

When starting wsbrd and if it is running as expected, you should see a similar trace on the terminal.

```
$ sudo wsbrd -F examples/wsbrd.conf -u /dev/ttyACM0

Silicon Labs Wi-SUN border router v1.0.0

Connected to RCP "0.12.0" (0.12.0), API 0.12.0

[DBG ][core]: Allocate Root Tasklet

[DBG ][6lo ]: P.Init

Successfully registered to system DBus

[DBG ][core]: NS Root task Init

[DBG ][sck ]: Socket Tasklet Generated

[INFO][wsbs]: WS tasklet init
```

In the traces, the wsbrd, RCP, and API versions are output. The wsbrd automatically checks the RCP and API versions to confirm they are valid and supported by the wsbrd version running.

If the default configuration is used, a new border router is advertising a network with the name “Wi-SUN Network”. On the Raspberry Pi, a new IP interface is created with the name “tun0”. Here is an example of the IP configuration once wsbrd is running.

```
tun0: flags=4305<UP,POINTOPOINT,RUNNING,NOARP,MULTICAST>  mtu 1500

      inet6 fd12:3456::7c3f:f8e7:55c5:e07f  prefixlen 64  scopeid 0×0<global>

      inet6 fe80::d0a6:c144:ad24:9b1d  prefixlen 64  scopeid 0×20<link>

      unspec 00-00-00-00-00-00-00-00-00-00-00-00-00-00-00-00  txqueuelen 500  (UNSPEC)

      RX packets 10  bytes 752 (752.0 B)

      RX errors 0  dropped 0  overruns 0  frame 0

      TX packets 3  bytes 144 (144.0 B)

TX errors 0  dropped 0 overruns 0  carrier 0  collisions 0
```

The “tun0” interface can be natively used to interact with the Wi-SUN devices connected to the Linux border router.

Interact with a Running wsbrd Instance

wsbrd Command Line Interface

wsbrd provides a command line interface to interact with the Linux border router when it is running. The interface can be used to retrieve configurations like the network name, Wi-SUN PHY configuration, PAN ID, GAK/GTK and LGAK/LGTK keys, and the MAC Addresses of the border router and the connected nodes. You can use the “wsbrd_cli status” command to view the list of configurations. For help, use the command “wsbrd_cli help”.

```
$  wsbrd_cli status
network_name: Wi-SUN Network
fan_version: FAN 1.1
domain: NA
phy_mode_id: 2
chan_plan_id: 1
panid: 0×3f48
size: SMALL
GAK[0]: 0f:a7:bc:8e:fa:1f:a3:22:6b:15:30:80:dd:06:6d:e7
GAK[1]: 95:03:54:82:78:c2:01:69:79:96:96:72:9a:f9:08:b9
GAK[2]: 95:03:54:82:78:c2:01:69:79:96:96:72:9a:f9:08:b9
GAK[3]: 95:03:54:82:78:c2:01:69:79:96:96:72:9a:f9:08:b9
GTK[0]: e9:a6:e0:19:00:8d:ae:12:4a:1d:73:93:ee:53:c6:6b
GTK[1]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
GTK[2]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
GTK[3]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
LGAK[0]: a5:01:bf:df:aa:fa:da:84:a5:9d:9c:41:53:31:03:9e
LGAK[1]: 95:03:54:82:78:c2:01:69:79:96:96:72:9a:f9:08:b9
LGAK[2]: 95:03:54:82:78:c2:01:69:79:96:96:72:9a:f9:08:b9
LGTK[0]: 61:ff:86:39:6c:ec:00:52:c9:fd:27:90:f1:db:f7:f5
LGTK[1]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
LGTK[2]: 00:00:00:00:00:00:00:00:00:00:00:00:00:00:00:00
38:39:8f:ff:fe:99:9b:e4
```

DBus Interface

The DBus interface can also interact with the Linux border router when it's running and retrieve static configurations. It is also able to trigger actions like adding or revoking root certificates and revoking a Wi-SUN device access to the network. You can execute the command `busctl introspect com.silabs.Wisun.BorderRouter /com/silabs/Wisun/BorderRouter` to see the list of all the properties that can be retrieved with in the DBus interface and the methods that can be used. For example, to read a DBus property execute the following command:

```
$ busctl get-property com.silabs.Wisun.BorderRouter /com/silabs/Wisun/BorderRouter com.silabs.Wisun.BorderRouter WisunNetworkName

s "Wi-SUN Network"
```

For additional information on the interface usage, refer to the D-Bus API documentation.

Raspberry Pi Adapter BRD8016A

BRD8016A is an excellent solution to build a Wi-SUN Linux Border Router on top of a Raspberry Pi, together with a Wi-SUN Radio Board used as the Wi-SUN device. The Wi-SUN Linux Border Router is ultimately made of 3 boards stacked together:

- Raspberry Pi (Linux host)
- BDR8016A (Adapter Board)

• Radio Board (RCP: Radio Co-Processor)

This setup is an alternative to connecting the Raspberry Pi to the Radio Board over USB via a WSTK/WPK mainboard, as initially described in the Wi-SUN Linux Border Router README.

wsbrd configuration

This setup uses the *Normal UART* instead of *UART over USB* usually used with the WSTK/WPK. For the wsbrd to use the *Normal UART*, change the following option in the wsbrd.conf file:

```
uart_device = /dev/ttyAMA0
```

Raspberry Pi Boot Configuration

To use the *Normal UART* by the Raspberry Pi, change the configuration in `/boot/config.txt`. Because of the way Raspberry Pi 2/3/4 are wired, to get a good UART clock, it is required to disable Bluetooth and use GPIOs 14 and 15 for /dev/ttyAMA0. See the Raspberry Pi documentation for details. To achieve this, the following changes need to be added at the end of `/boot/config.txt` :

```
[all]

dtoverlay=disable-bt

enable_uart=1

gpio=23=op,dh
```

It is required to reboot the Raspberry Pi following such changes:

```
sudo reboot
```

After rebooting your Raspberry Pi, follow these steps to enable shell messages on the serial connection and disable the login shell over serial:

- Open Raspberry Pi configuration GUI:

```
sudo raspi-config
```

- Select `3 Interface Options  Configure connections to peripherals`
- Select `I6 Serial Port  Enable/disable shell messages on the serial connection`
- Answer "**No**" to the question "Would you like a login shell to be accessible over serial?"
- Answer "**Yes**" to the question "Would you like the serial port hardware to be enabled?"
- Choose to reboot your Raspberry Pi

Now that the Raspberry Pi is configured, connect the RCP to the adapter and start wsbrd.

Note that the switch on the adapter (BRD8016A) must be on *High Power (LDO)* to use 5V from the Raspberry Pi to generate a regulated 3.3V supply.

CPCD and wsbrd

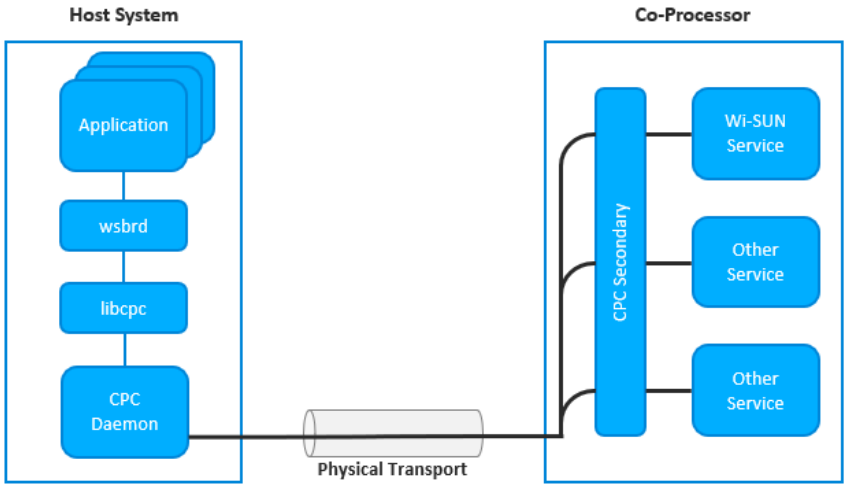
Using Co-Processor Communication Daemon with wsbrd

Overview

Co-Processor Communication (CPC) enables one host system to communicate with a Network co-processor device (NCP), by physical transport (UART, SPI, and so on). In CPC, data transfers between processors are segmented in sequential packets. Transfers are guaranteed to be error-free and sent in order. Multiple applications can send or receive on the same endpoint without worrying about collisions. A CPC daemon (CPCd) is provided to allow applications on Linux to interact with a secondary running CPC.

For the purpose of having Silicon Labs Wi-SUN stack running in harmony with other services such as Bluetooth, ZigBee, or others on the same Co-Processor, Silicon Labs added support of CPCd to wsbrd.

The next sections explain how to install and configure CPCd and wsbrd, and also how to set up an RCP with the CPC Secondary Service component in Simplicity Studio v5. This chapter only covers UART communication between a WSTK and Raspberry Pi.



Setting Up the RCP with CPC Secondary Service

To use the **Wi-SUN – RCP** sample application as CPC secondary device, users should enable the Co-Processor Communication components in the **Wi-SUN – RCP** project in Simplicity Studio v5.

Components Installation

After creating a **Wi-SUN – RCP** project, open the slcp file and select the **Software Component** tab. In the search box, enter ”cpc” to filter the components and install the following components:

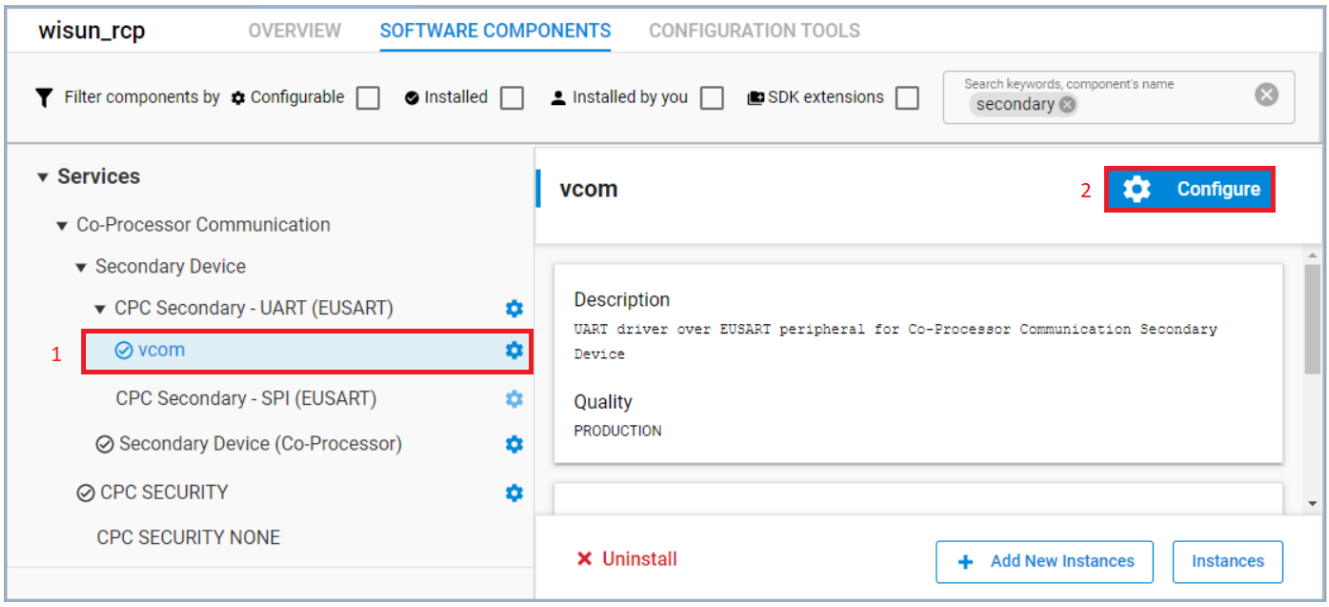
1. Install **CPC Secondary – UART**, enter a name for the component instance in the **Create a Component Instance** pop-up, and click **Done**.

2. Install **Secondary Device (Co-Processor)** if it wasn't installed automatically after installing **CPC Secondary.**
3. **CPC SECURITY** component will be installed automatically after installing the **Secondary Device (Co-Processor)** component. If you wish to disabled security to reduce memory footprint, install **CPC SECURITY NONE**.

Secondary Device Configuration

CPC Secondary - UART Component

After installing all the components, configure the created instance with the desired Flow control and Baud rate. Select your created instance in the slcp perspective, and then click **Configure**.



In the configuration perspective, the recommended **EUSART Baudrate** value is 115200. Concerning the **Flow Control**, it is set by default to **CTS/RTS**. Silicon Labs recommends this default; do not set it to **None**.

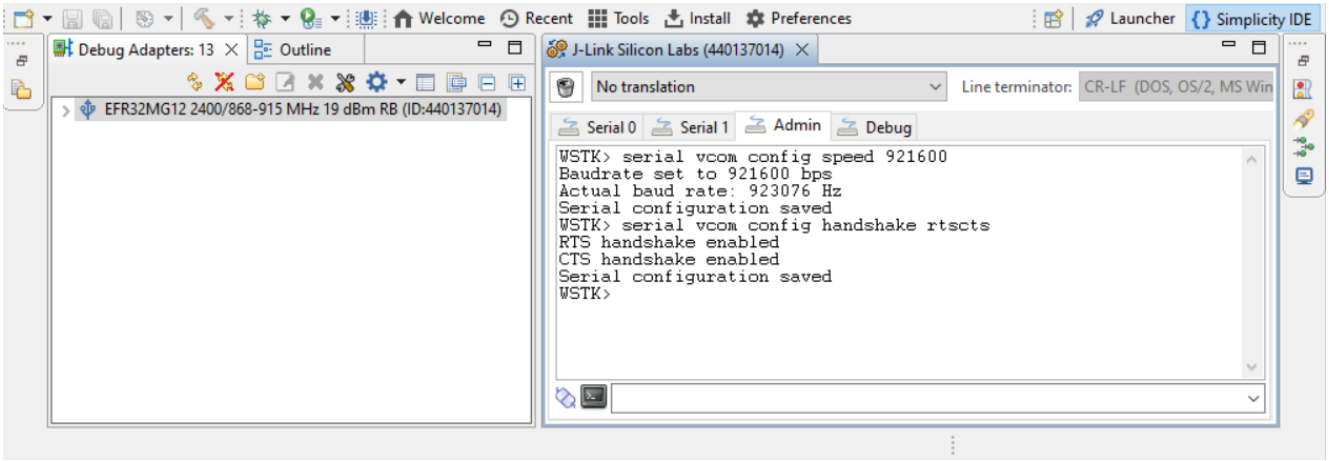
WSTK Configuration

The WSTK boards are factory programmed to support 115200bps with no flow control. To align the WSTK configuration with the Chosen component configuration, follow the next steps:

1. In **Debug Adapters**, right-click your device and click **Launch Console...**
2. On the console, select the **Admin** tab, and enter the following commands:

```
serial vcom config handshake rtscts
```

The following figures shows the resulting console output.



After finishing these steps, you can **Run** your project and flash it to the device.

CPCd Installation and Configuration

CPCd Installation

CPCd is delivered as a CMake project. The minimum version of CMake required to generate the project is 3.10. CPCd requires MbedTLS to encrypt the endpoints. The minimal version required for Mbed-TLS is 2.7.0. libmbedtls-dev must be installed to compile from sources.

Refer to cpc-deamon readme , sections ***Compiling CPCd*** and ***Installing CPCd and CPC library*** to install CPCd.

CPCd Configuration

After installing CPCd, a configuration file can be found under "/usr/local/etc/cpcd.conf". This file can be accessed using the following command:

```
sudo nano /usr/local/etc/cpcd.conf
```

The modifications that need to be applied to this file to configure CPCd for a secured UART communication between a WSTK and Raspberry Pi are:

```
instance_name: cpcd_0

bus_type: UART

uart_device_file: /dev/ttyACM0

uart_device_baud: 921600

disable_encryption: false
```

Note that the uart_device_file and uart_device_baud values above are examples and can be changed to suit your use case.

If the security was disabled on the secondary device by installing the **CPC SECURITY NONE** component, the value of **disable_encryption** must be set to **true**.

wsbrd Configuration

CPC is supported by wsbrd version 1.4 and above. Make sure to update and re-build wsbrd if you are running an older version. In case of reinstalling wsbrd, make sure to delete the file /wisun-br-linux/CMakeCache.txt.

In the **wsbrd.conf** file that can be found under **wisun-br-linux/examples**, comment-out *cpc_instance* and give it the same value as *instance_name* in **cpcd.conf** and comment the option *uart_device*.

Launch CPDC and wsbrd

Before starting CPCd and wsbrd, connect the RCP to your Raspberry Pi. Then you can start the CPCd using the following command:

```
$sudo cpcd
.
..
..
Info : Daemon startup was successful. Waiting for client connections
```

After the line indicating that the deamon startup was successful appears in the logs, use a separate terminal to launch wsbrd using the command:

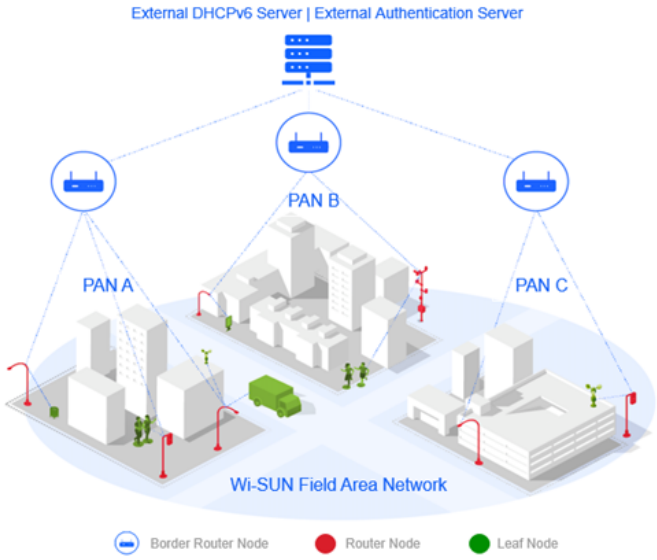
```
sudo wsbrd -F examples/wsbrd.conf
```

External Servers

External Servers

Silicon Labs provides external DHCPv6 and authentication servers with configuration examples to give you the possibility of managing your network at a large scale, and to avoid using the internal default server that is meant for an out-of-the-box experience, and doesn't allow too much flexibility in configuration.

The external DHCPv6 servers can help you unify the IPv6 routing around all your nodes, instead of having different prefixes set by each of your border routers. And having an external authentication server, will let you manage authentication on all your network using one server.



Refer to Wi-SUN Linux Border Router repository Readme and follow the guidelines to use the configuration examples.

IP Communication

IP Communication

This section provides guidelines to ping a Wi-SUN device from the Raspberry Pi running wsbrd. These guidelines assume that wsbrd has been configured and is running on the Raspberry Pi.

- Ping and UDP: Detailed guidelines to ping the nodes after they are connected to the Border Router and perform UDP communication between using Netcat.
- CoAP: Get the Wi-SUN Meter temperature and humidity values and control the boards LEDs using CoAP.
- Wi-SUN Multicast: Silicon Labs Wi-SUN stack implements Multicast Protocol for Low-Power and Lossy Networks (MPL) as described in RFC7731, to support multicast packets messaging specified in the Wi-SUN FAN Technical Profile Specification version 1.1v06.

Ping and UDP

Ping and UDP

To complete the following steps correctly, you need a Wi-SUN node running the **Wi-SUN – SoC Ping** application on the network. For more information, see *QSG181: Silicon Labs Wi-SUN SDK Quick-Start Guide* for details on how to bring-up the Wi-SUN Ping project.

Connect a Wi-SUN Node to the Border Router

After the project is flashed and running on the Wi-SUN device, connect to the CLI interface. From this interface, retrieve the Wi-SUN border router IP address and the Wi-SUN node IP address by executing **wisun get wisun**. They are available only if the device is successfully connected. If this is not the case, wait for the connection process to complete. If the connection is not successful, verify the border router and router configurations match (network name, Wi-SUN PHY, certificates, and so on). You should get the following trace on the console.

```
> wisun get wisun
wisun.network_name = Wi-SUN Network
wisun.phy_config_type = FAN 1.1 (1)
wisun.network_size = small (1)
wisun.tx_power = 20
wisun.regulatory_domain = NA (1)
wisun.operating_class = 0 (unused)
wisun.operating_mode = 0x0 (unused)
wisun.chan_plan_id = 1
wisun.phy_mode_id = 2
wisun.ch0_frequency = 0 (unused)
wisun.number_of_channels = 0 (unused)
wisun.channel_spacing = 100kHz (0) (unused)
wisun.join_state = Operational (5)
wisun.ip_address_global = fd12:3456::be33:acff:fef6:3161
wisun.ip_address_link_local = fe80::be33:acff:fef6:3161
wisun.ip_address_border_router = fd12:3456::86fd:27ff:fefe:55bd
wisun.ip_address_primary_parent = fd12:3456::86fd:27ff:fefe:55bd
wisun.regulation = none (0)
wisun.regulation_warning_threshold = 85
wisun.regulation_alert_threshold = 95
```

The Wi-SUN border router IP address is fd12:3456::86fd:27ff:fefe:55bd (wisun.ip_address_border_router value). The Wi-SUN node IP address is fd12:3456::be33:acff:fef6:3161 (wisun.ip_address_global value). These IP addresses are used in the following steps to address the devices.

Check the Communication Between the Raspberry Pi and the Node

From the same Linux terminal, ping the Wi-SUN node.

```
$ ping fd12:3456::be33:acff:fef6:3161

PING fd12:3456::be33:acff:fef6:3161(fd12:3456::be33:acff:fef6:3161) 56 data bytes
64 bytes from fd12:3456::be33:acff:fef6:3161: icmp_seq=1 ttl=64 time=675 ms
64 bytes from fd12:3456::be33:acff:fef6:3161: icmp_seq=2 ttl=64 time=680 ms
64 bytes from fd12:3456::be33:acff:fef6:3161: icmp_seq=3 ttl=64 time=705 ms
64 bytes from fd12:3456::be33:acff:fef6:3161: icmp_seq=4 ttl=64 time=925 ms
64 bytes from fd12:3456::be33:acff:fef6:3161: icmp_seq=5 ttl=64 time=281 ms
64 bytes from fd12:3456::be33:acff:fef6:3161: icmp_seq=6 ttl=64 time=149 ms
```

This communication goes through the Linux IP interface to the communication bus. It is then received by the EFR32 running the RCP image and sent over the air using the Wi-SUN protocol. Finally, the packet is received by the Wi-SUN node which replies with its own packet. The new packet takes the same route the other way round. The latency is the accumulation of all the step durations.

You have successfully opened an upstream communication between the Wi-SUN network and a device in the backhaul network. To go further, you can use this new communication link to open TCP/UDP sockets between a Wi-SUN node and the Linux host.

Send and Receive UDP Data

Assuming that you have retrieved the iPv6 addresses of the border router and the node, as explained in the beginning of this page. And you have a Wi-SUN node running **Wi-SUN – SoC CLI** project. You can send and receive UDP data using netcat server.

Send UDP data

To send data, make sure to open a UDP server port on your node first, using the following command:

```
wisun udp_server [port]
```

And on your Linux host, use netcat command to send UDP data:

```
$nc -u [Wi-SUN node IP address] [port]
```

Receive UDP Data

To receive data on your Linux host, open a UDP server port using netcat command:

```
$nc -6 -lu [port]
```

Then you can send data from your node with the following commands:

```
wisun udp_client [Wi-SUN border router IP address] [port]

wisun socket_write [Socket ID] [Wi-SUN border router IP address] [Port] [Message]
```

Note that the Socket ID value, will be returned with command `wisun udp_client`.

CoAP

CoAP Communication

Get the Metering Data

The Raspberry Pi gets CoAP metering data from the CoAP meter node using the libcoap client, if you have a node running **Wi-SUN SoC CoAP Meter** in the same network with the Linux Border Router. Follow the steps below to get CoAP metering data on your Raspberry Pi:

1. Install the libcoap library.

```
$sudo apt-get install libcoap-1-0

$sudo apt-get install libcoap-1-0-bin
```

2. From the node cli, make sure that you are connected to the network.

```
> wisun get wisun.join_state

wisun.connection_state = Operational (5)
```

3. Get your node global address.

```
> wisun get wisun.ip_address_global

wisun.ip_address_global = fd12:3456::be33:acff:fef6:3161
```

Discover the Available Resources

The CoAP protocol supports an interoperable discovery feature. A CoAP client can request the attributes hosted by a CoAP server. In the case of the CoAP Meter application, the available resources can be retrieved using the libcoap GET method with the standard discovery entry-point. The following command shows the sensor and LED resources.

```
$coap-client -m get -N coap://[fd12:3456::be33:acff:fef6:3161]:5683/.well-known/core

</gpio>;ct=40,</sensor>;ct=40,</gpio/led>;rt="led";if="gpio",</sensor/light>;rt="light";if="sensor",</sensor/humidity>;rt="humidity";if="sensor",
</sensor/temperature>;rt="temperature";if="sensor",</sensor/all>;rt="all";if="sensor"
```

Get the CoAP Meter Sensor Data

On your raspberry Pi, get the CoAP meter sensor data using libcoap with the following command:

```
$coap-client -m get -N coap://[fd12:3456::be33:acff:fef6:3161]:5683/sensor/all

{
  "id": 0,
  "temp": 28.18,
  "hum": 46.36,
  "lx": 512
}
```

Each sensor data value can be retrieved alone if you specify the corresponding resource in the libcoap command. To retrieve the humidity value, the user can use the following command:

```
$coap-client -m get -N coap://[fd12:3456::be33:acff:fef6:3161]:5683/sensor/humidity

46.59 %
```

Note that any machine that has a CoAP client and an IPv6 connectivity with the CoAP Meter node can get the metering data.

Toggle the LEDs

The libcoap PUT method allows you to toggle the CoAP Meter node LEDs remotely. Use the following command to toggle the LED0.

```
$coap-client -t text -m put -N -B 1 coap://[fd12:3456::be33:acff:fef6:3161]:5683/gpio/led -e "LED0%00"
```

LED1 can also be toggled with changing the payload to “LED1%00”.

Multicast

Wi-SUN Multicast

Silicon Labs Wi-SUN stack implements Multicast Protocol for Low-Power and Lossy Networks (MPL) as described in RFC7731, to support multicast packets messaging specified in the Wi-SUN FAN Technical Profile Specification version 1.1v06.

This page goes over the different aspects of Wi-SUN multicast aspects to allow users sending multicast messages over their Wi-SUN Network.

Wi-SUN Multicast Scopes

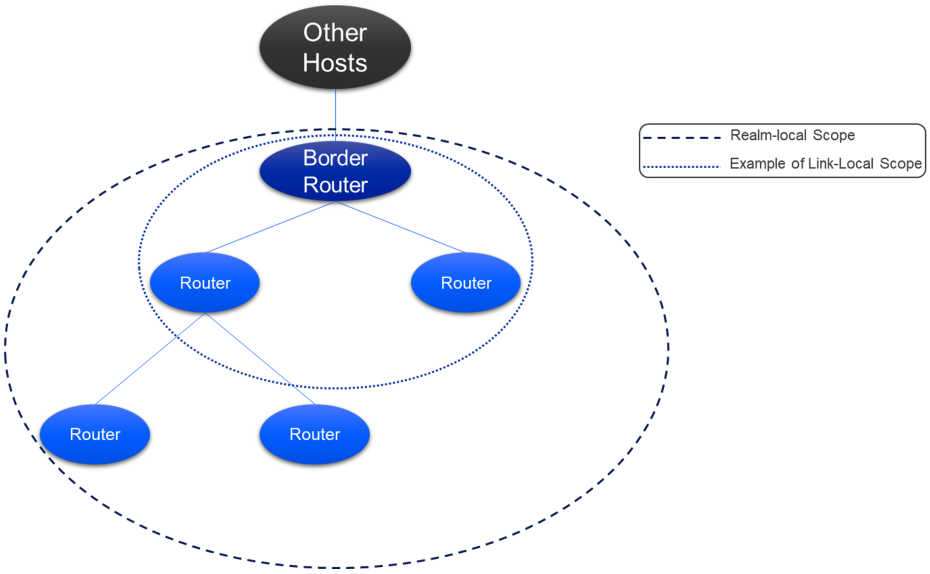
A **scope zone** as described by [RFC 4007] is a topological span within which the address may be used as a unique identifier for an interface or set of interfaces.

The Wi-SUN FAN Specification requires subscription to a couple of multicast addresses which covers two different scopes:

- **The Link Local scope (FF02::)**: which includes all neighbors in the limit of 1 hop.
- **The Realm Local scope (FF03::)**: which includes all the network nodes at any limit of hops.

With the Silicon Labs Wi-SUN FAN Stack, FFNs are able to originate multicast messages destined for the required scopes listed above or any other scope. They are also able to forward multicast messages, whether they originate from the same PAN or from another host.

As the figure below shows, other hosts can send multicast packets destined to multicast groups to which PAN nodes are subscribed. The border router must join those multicast groups to enable forwarding the multicast packets of other hosts into the PAN.



Wi-SUN Multicast Addresses

By default, all the Wi-SUN devices subscribe to the ALL_MPL_FORWARDERS address with a realm-local scope: `FF03::FC` as shown in the figure below.

In addition to the ALL_MPL_FORWARDERS address, the Border Router and the FFNs join their FAN interface to several predefined IPv6 multicast groups by default:

- **ff02::1 :** targets all the nodes in the link local scope.
- **ff02::2 :** targets all the routers in the link local scope.
- **ff03::1 :** targets all the nodes in a Realm local scope.
- **ff03::2 :** targets all the routers in a Realm local scope.

The Border Router and FFNs are also able to join other specific Multicast groups different than the ones joined by default.

Sending a Multicast Packet in Wi-SUN Network

This section explains how to send a multicast message from the Wi-SUN Linux Border Router (wsbrd) to the FFN nodes, using a socket created by the python script at the end of this section.

Setting Up The Nodes

1. Run the **Wi_SUN - SoC CLI example** sample application on your nodes.
2. Start a UDP server on the nodes.

```
wisun  udp_server 1234
```

> For multicast addresses other than the default ones listed in the section Wi-Sun Multicast Addresses, run the command below to join a multicast group.

```
wisun socket_set_option [socket id] join_multicast_group [multicast address]
```

Setting Up The Border Router

For multicast addresses other than the default ones listed in the section Wi-Sun Multicast Addresses, run the command below to join a multicast group *(eg: ff03::db8:0:0)*. Check D-Bus API documentation for more information.

```
sudo busctl --system call com.silabs.Wisun.BorderRouter /com/silabs/Wisun/BorderRouter com.silabs.Wisun.BorderRouter JoinMulticastGroup "ay" 16 0xff 0x03 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x0d 0xb8 0x00 0x00 0x00 0x00
```

> The multicast socket must be bound to the wsbrd interface in order to reach the Wi-SUN PAN. The script used in this document binds the socket to interface tun0 using the socket option SO_BINDTODEVICE.

Send the Multicast Message

Run the following command to send a message to all the routers in the Realm Local scope of the Border Router using the script multicast_packet_send.py.

```
python3 multicast_packet_send.py ff03::2 1234 "Hello World!!"
```

> The script sets the socket limit to 10 hops being that the default value on Linux is 1 hop. The number of hops can be changed by setting the socket option **IPV6_MULTICAST_HOPS** to a different value.

multicast_packet_send.py

```
import socket
import sys
import time

if len(sys.argv) != 3:
    print(
        'Usage: python3 multicast_packet_send.py [dest IPv6] [dest port] [message]' )
    exit(1)

sock = socket.socket(socket.AddressFamily.AF_INET6,
                     socket.SocketKind.SOCK_DGRAM)
sock.setsockopt(socket.IPPROTO_IPV6, socket.IPV6_MULTICAST_HOPS, 10)
sock.setsockopt(socket.SOL_SOCKET, socket.SO_BINDTODEVICE, bytes("tun0", 'ascii'))
sock.sendto(sys.argv[3], (sys.argv[1], int(sys.argv[2])))

print('packet sent')
```

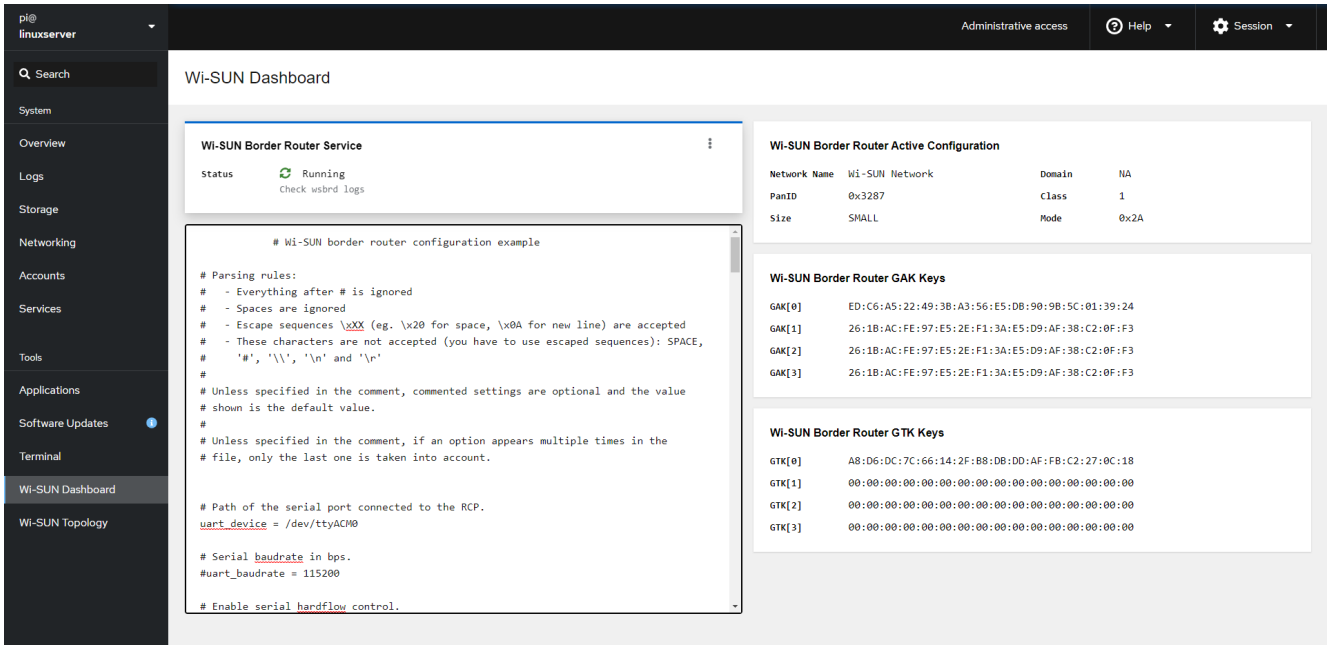
Border Router GUI

Wi-SUN Border Router GUI

The Wi-SUN border router GUI is a companion tool for Silicon Labs' Wi-SUN border router reference design, wsbrd. The tool helps manage the network by enabling a remote border router configuration and provides a visual representation of the connected Wi-SUN devices. For more information about the installation of the tool and the quick start guide, refer to the wisun-br-gui repository.

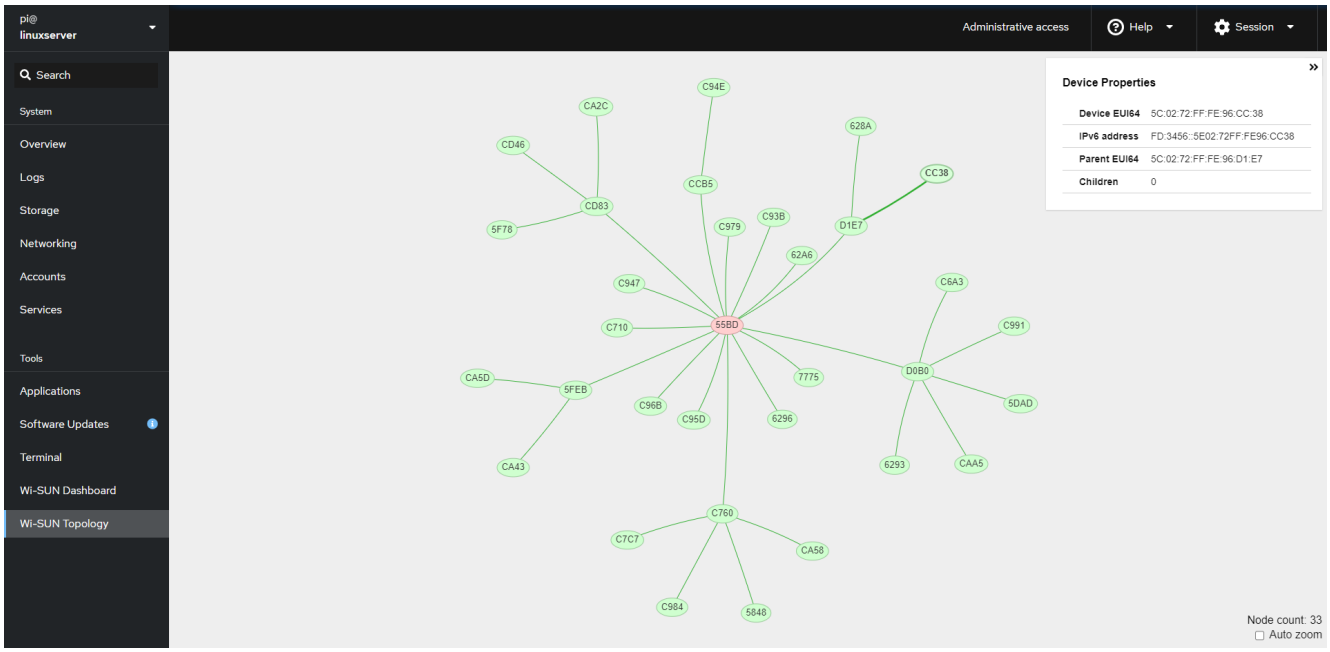
Wi-SUN Dashboard

The Wi-SUN border router GUI provides a user interface to configure the border router by modifying the wsbrd.conf file and allows the user to start, restart, and stop the Wi-SUN border router service. In the same interface, the tool shows the running configuration on the border router (Network name, PanID, Size, Domain, Class, Mode, and the GAK/GTK Keys).



Wi-SUN Topology

The Wi-SUN Topology tool draws your implemented Wi-SUN network in the form of a graph constructed of nodes and edges linking each child to its parent. The tool also lists properties of each device when it is selected in a separate box.


pi@
linuxserver
Administrative access
Help
Session
Search
System
Overview
Logs
Storage
Networking
Accounts
Services
Tools
Applications
Software Updates
Terminal
Wi-SUN Dashboard
Wi-SUN Topology
Device Properties
Device EUI64 5C:02:72:FF:FE:96:CC:38
IPv6 address FD:3456::5E02:72FF:FE96:CC38
Parent EUI64 5C:02:72:FF:FE:96:D1:E7
Children 0
C94E
CA2C
628A
CD46
CC38
CCB5
CD83
D1E7
5F78
C979
C93B
62A6
C947
C6A3
C710
55BD
C991
CA5D
7775
D0B0
5FEB
C96B
5DAD
C95D
6296
CA43
6293
CAA5
C760
C7C7
CA58
C984
5848
Node count: 33
Auto zoom

Overview

Network Performance

This section introduces ways to optimize a Wi-SUN network's performance.

- Using the Wi-SUN Network Measurement Application (PDF): Describes how to use the Wi-SUN Network Measurement Application to test Silicon Labs Wi-SUN FAN stack performance.
- Silicon Labs Wi-SUN Mesh Network Performance (PDF): Details methods for testing the Wi-SUN FAN stack network performance compared to other mesh networks available. When selecting a network or device, designers need to know the network's performance and behavior characteristics such as battery life, network connection time and latency, and the impact of network size on scalability and reliability.

Wi-SUN Overview

Wi-SUN Overview

Wi-SUN Stack

The Wi-SUN stack API is the primary Application Programming Interface (API) for applications running on Silicon Labs EFR32 Wireless Gecko SoCs to interact with the Silicon Labs Wi-SUN FAN wireless stack. It allows the application to manage the connection to a Wi-SUN FAN network as well as to communicate with other devices in the network using a socket-based communication interface.

See the Wi-SUN Stack API for more details.

Stack Plugins

Wi-SUN stack plugin components are software modules tightly linked to the stack that provide means to customize it: debug, manufacturing or Wi-SUN specific optional features. They can have significant impact on key capabilities and footprint.

- RF Test provides low-level APIs to produce an RF tone or a modulated packet and calibrate the radio.
- Stack Trace and Debug provides extended trace capabilities to the stack and stack plugin components.

Service Components

The following software components are provided to help and accelerate Wi-SUN application developments by offering common functionalities. They can easily be added to an existing Wi-SUN application through Simplicity Studio graphical interface. The components are shared in source code in the Gecko SDK.

- Application Core provides a set of high-level helper APIs designed to ease the application development.
- Util Functions provides utility functions.
- CoAP provides a CoAP (Constrained Application Protocol) implementation running on top of the Wi-SUN stack.
- Ping provides a ping implementation based on the ICMPv6 protocol.
- iPerf provides an iPerf2 implementation to test the throughput over UDP.
- Over-The-Air Device Firmware Upgrade (Alpha) provides an Over-The-Air Device Firmware Upgrade solution for Wi-SUN devices.
- Silicon Labs socket API (deprecated) provides a compatibility with Silicon Labs former socket API.

Versioning

Silicon Labs Wi-SUN solution follows the Semantic Versioning guidelines for release cycle transparency and to maintain backward compatibility.

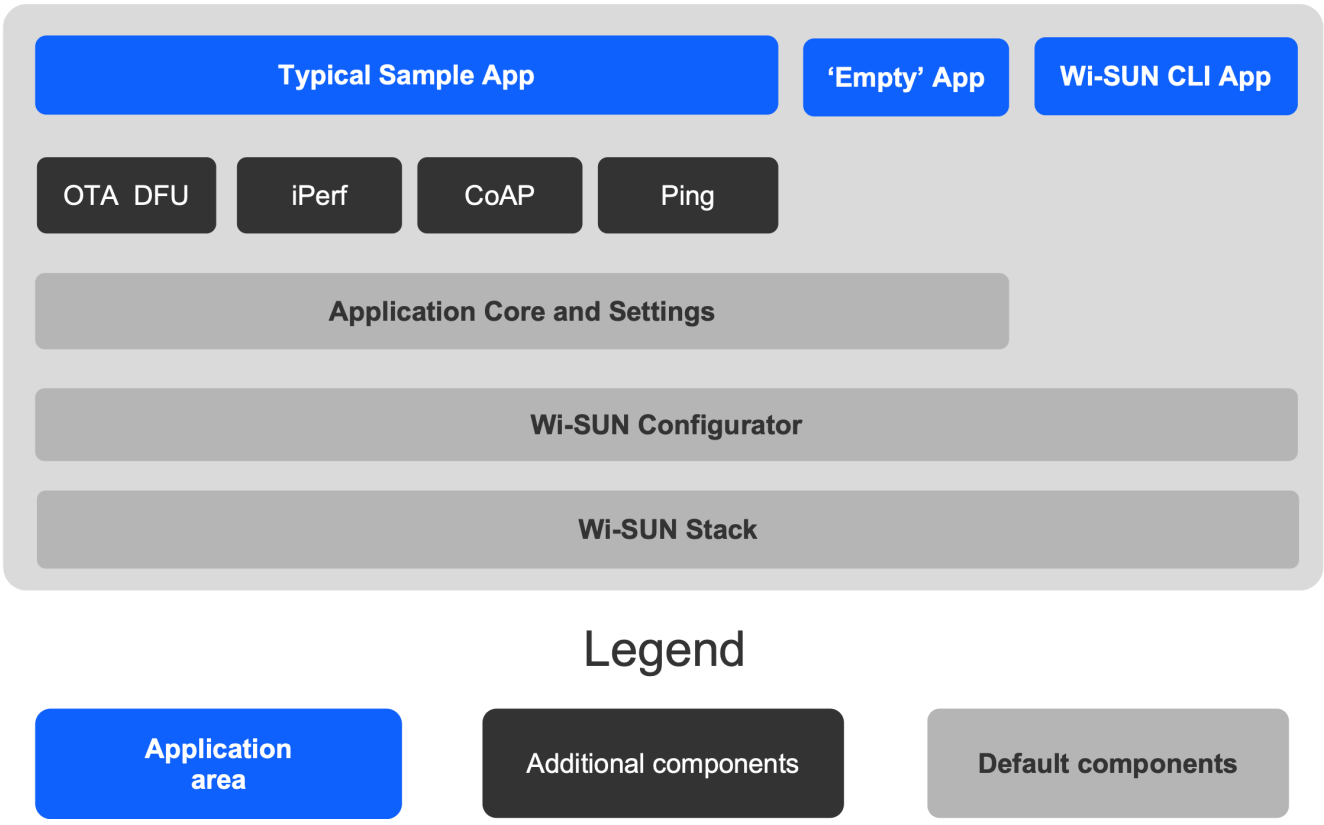
Wi-SUN Services

Wi-SUN Services

The Wi-SUN service software components are provided to help and accelerate Wi-SUN application developments by offering common functionalities. They can easily be added to an existing Wi-SUN application through Simplicity Studio graphical interface. The components are shared in source code in the Gecko SDK.

The service components are especially useful to build upon the Wi-SUN - SoC Empty project and start as close as possible to a final application targeted by a developer.

The following diagram shows the software components available for users to leverage in their Wi-SUN application. Moreover, it highlights each component dependencies to other software components.



Modules

Util Functions

Util Functions

The Util Functions component provides helper functions to inform the application about the Wi-SUN PHY configured in the RAIL configuration file.

Functions

sl_status_t	sl_wisun_util_get_rf_settings(uint8_t *reg_domain, uint8_t *op_class, uint16_t *op_mode) SL_DEPRECATED_API_SDK_4_2 Get frequency band settings of the first RAIL configuration listed in RAIL's channelConfigs array.
sl_status_t	sl_wisun_util_get_phy_config(sl_wisun_phy_config_t *phy_config) Get PHY settings of the first RAIL configuration listed in RAIL's channelConfigs array.
sl_status_t	sl_wisun_util_connect(const uint8_t *network_name) Connect to a Wi-SUN network.

Function Documentation

sl_wisun_util_get_rf_settings

> sl_status_t sl_wisun_util_get_rf_settings (uint8_t *reg_domain, uint8_t *op_class, uint16_t *op_mode) SL_DEPRECATED_API_SDK_4_2

Get frequency band settings of the first RAIL configuration listed in RAIL's channelConfigs array.

Parameters

[out]	reg_domain	Regulatory domain of the Wi-SUN network
[out]	op_class	Operational class of the Wi-SUN network
[out]	op_mode	Operational mode of the Wi-SUN network

Returns

- SL_STATUS_OK if successful, an error code otherwise
- One of the following:
  - SL_STATUS_OK if successful
  - SL_STATUS_INVALID_CONFIGURATION if a configuration that cannot be managed by the plugin is used
  - SL_STATUS_FAIL if an other error occured

Warnings

- Do not call this function while the Wi-SUN stack is started.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See sl_wisun_util_get_phy_config() for a replacement.

Definition at line `62` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_util.h`

sl_wisun_util_get_phy_config

```
sl_status_t sl_wisun_util_get_phy_config (sl_wisun_phy_config_t *phy_config)
```

Get PHY settings of the first RAIL configuration listed in RAIL's channelConfigs array.

Parameters

[out]	phy_config	Pointer to PHY configuration

Returns

- SL_STATUS_OK if successful, an error code otherwise
- One of the following:
  - SL_STATUS_OK if successful
  - SL_STATUS_INVALID_CONFIGURATION if a configuration that cannot be managed by the plugin is used
  - SL_STATUS_FAIL if an other error occured

Warnings

- Do not call this function while the Wi-SUN stack is started.

Definition at line `79` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_util.h`

sl_wisun_util_connect

```
sl_status_t sl_wisun_util_connect (const uint8_t *network_name)
```

Connect to a Wi-SUN network.

Parameters

[in]	network_name	Name of the Wi-SUN network as a zero-terminated string

Returns

- SL_STATUS_OK if successful, an error code otherwise
- One of the following:
  - SL_STATUS_OK if successful
  - SL_STATUS_INVALID_CONFIGURATION if a configuration that cannot be managed by the plugin is used
  - SL_STATUS_FAIL if an other error occured

Since Wi-SUN frequency band settings are deduced from first RAIL configuration listed in RAIL's channelConfigs array, using this function is not recommended if more than one RAIL configuration is described.

Definition at line `97` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_util.h`

Application Core

Application Core

The Application Core component provides utilities common to most Wi-SUN applications with the following features:

- **A Wi-SUN connection helper function app_wisun_network_connect()** configures the network size setting, the TX output power, the certificates, and keys. The last function action is to start the Wi-SUN connection process. The app_wisun_network_is_connected() function provides a simple connection status getter too.
- **Event handlers for basic Wi-SUN network events** component includes the basic event handler implementations.
- **Available addresses**. Retrieve known IPv6 addresses with the app_wisun_get_current_addresses() function.
- **Error Getter**. Use app_wisun_core_get_error() function to get the status of the Stack API calls.

To use the component in your application, add it to your project and initialize it with app_wisun_core_init().

Modules

Application Core API type definitions

void	app_wisun_project_info_init(const char *app_name) Initialize Wi-SUN project information.
void	app_wisun_project_info_print(const bool json_format) Print Wi-SUN project information.
const app_project_info_t *	app_wisun_project_info_get(void) Get Wi-SUN Project info.
void	app_wisun_wait_for_connection(void) Wait for the connection.
void	app_wisun_connect_and_wait(void) Connect and wait for connection.
bool	app_wisun_network_is_connected(void) The network is connected.
void	app_wisun_dispatch_thread(void) Thread dispatch function.

Functions

void	app_wisun_core_init(void) Initialize Wi-SUN application core.
bool	app_wisun_core_get_error(app_core_error_state_flag_t flag) Get application core error.
void	app_wisun_network_connect(void) Connect to the Wi-SUN network.

void	app_wisun_get_current_addresses(current_addr_t *const dest_addresses) Get the current addresses.
void	app_wisun_set_regulation_active(bool enabled) Set the regional regulation to active or passive.
bool	app_wisun_get_regulation_active(void) Return the state of the regional regulation.
bool	app_wisun_get_remaining_tx_budget(uint32_t *const budget_out) Get the remaining budget from the transmission quota.
void	app_wisun_set_regulation_thresholds(const int8_t warning_level, const int8_t alert_level) Set up warning and alert thresholds for the regional regulation.
bool	app_wisun_get_regulation_thresholds(regulation_thresholds_t *thresholds_out) Get the warning and alert levels for approaching/exceeded the TX budget.
sl_wisun_join_state_t	app_wisun_get_join_state(void) Get Wi-SUN join state.
void	app_wisun_get_time_stat(app_core_time_stat_t *const tstat) Get time statistic.

undefined Documentation

app_wisun_project_info_init

```
void app_wisun_project_info_init (const char *app_name)
```

Initialize Wi-SUN project information.

Parameters

[in]	app_name	Application name

Init internal instance

Definition at line 94 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core_util.h

app_wisun_project_info_print

```
void app_wisun_project_info_print (const bool json_format)
```

Print Wi-SUN project information.

Parameters

[in]	json_format	Json format required indicator

Print project info in pretty or json format.

Definition at line 101 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core_util.h

app_wisun_project_info_get

```
const app_project_info_t * app_wisun_project_info_get (void)
```

Get Wi-SUN Project info.

Parameters

N/A		

Get a constant instance of internal Wi-SUN project info **Returns**

- app_project_info_t * Project info

Definition at line 108 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core_util.h

app_wisun_wait_for_connection

```
void app_wisun_wait_for_connection (void)
```

Wait for the connection.

Parameters

N/A		

This function doesn't call the app_wisun_network_connect() function. The function provides a delay loop with optional heart beat printing till the connection state has not been changed.

Definition at line 116 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core_util.h

app_wisun_connect_and_wait

```
void app_wisun_connect_and_wait (void)
```

Connect and wait for connection.

Parameters

N/A		

The function calls app_wisun_network_connect() function and app_wisun_wait_for_connection() function. It can be useful at the beginning of application thread.

Definition at line 124 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core_util.h

app_wisun_network_is_connected

```
bool app_wisun_network_is_connected (void)
```

The network is connected.

Parameters

N/A		

Wrapper function of join state getter **Returns**

- true Connected
- false Not connected

Definition at line 132 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core_util.h

app_wisun_dispatch_thread

```
void app_wisun_dispatch_thread (void)
```

Thread dispatch function.

Parameters

N/A		

For low power LFN mode, the delay value is 'APP_THREAD_LP_DISPATCH_MS', for FFN mode, the delay is 1ms

Definition at line 139 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core_util.h

Function Documentation

app_wisun_core_init

```
void app_wisun_core_init (void)
```

Initialize Wi-SUN application core.

Parameters

N/A		

Initializing mutex, socket handler and set Wi-SUN settings.

Definition at line 145 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_core_get_error

```
bool app_wisun_core_get_error (app_core_error_state_flag_t flag)
```

Get application core error.

Parameters

[in]	flag	is the indicator of the error

The function retrieves the application core error status based on the flag. **Returns**

- bool True if error flag is set, otherwise false

Definition at line 154 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_network_connect

```
void app_wisun_network_connect (void)
```

Connect to the Wi-SUN network.

Parameters

N/A		

Network initialization and connection. The function initializes the network with parameters (Network name, TX Power, Network size, etc.) by the stored settings in NVM if the settings component is added to the project, otherwise with the default settings.

Definition at line 165 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_get_current_addresses

```
void app_wisun_get_current_addresses (current_addr_t *const dest_addresses)
```

Get the current addresses.

Parameters

[out]	dest_addresses	Destination

Copy cached addresses into destination.

Definition at line 172 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_set_regulation_active

```
void app_wisun_set_regulation_active (bool enabled)
```

Set the regional regulation to active or passive.

Parameters

[in]	enabled	(true = active, false = not active)

After a stack API call for regional regulation, this function can be used to store the status of the regulation (active or not).

Definition at line 180 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_get_regulation_active

```
bool app_wisun_get_regulation_active (void)
```

Return the state of the regional regulation.

Parameters

N/A		

This function tells the caller if a regulation is currently active. **Returns**

- Boolean indicating if a regional regulation is currently active.

Definition at line 187 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_get_remaining_tx_budget

```
bool app_wisun_get_remaining_tx_budget (uint32_t *const budget_out)
```

Get the remaining budget from the transmission quota.

Parameters

[out]	budget_out	pointer to return the remaining budget to.

Returns the state of the regional regulation and the remaining budget in ms if applicable, or zero budget if exceeded or not regulated. **Returns**

- Boolean to indicate if the returned value reflects a usable value.

Definition at line 196 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_set_regulation_thresholds

> void app_wisun_set_regulation_thresholds (const int8_t warning_level, const int8_t alert_level)

Set up warning and alert thresholds for the regional regulation.

Parameters

[in]	warning_level	new percentage for the warning threshold
[in]	alert_level	new percentage for the alert threshold

Sets up the percentages of warnings and alerts where the regulation indicate that the transmission quota is approached/exceeded.

Definition at line 205 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_get_regulation_thresholds

> bool app_wisun_get_regulation_thresholds (regulation_thresholds_t *thresholds_out)

Get the warning and alert levels for approaching/exceeded the TX budget.

Parameters

[out]	thresholds_out	pointer to the struct to hold the thresholds

Values representing percentages of the allowed transmission quota in ms are returned for the warning and alert levels, respectively. **Returns**

- Boolean to indicate if the operation was successful

Definition at line 214 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_get_join_state

> sl_wisun_join_state_t app_wisun_get_join_state (void)

Get Wi-SUN join state.

Parameters

N/A		

Thread-safe getter to get connection state. Join state is stored in appropriate event callback. **Returns**

- sl_wisun_join_state_t Join state value.

Definition at line 222 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_wisun_get_time_stat

```
void app_wisun_get_time_stat (app_core_time_stat_t *const tstat)
```

Get time statistic.

Parameters

[out]	tstat	Time statistic structure

Create a copy of time statistic storage with up-to-date values

Definition at line 229 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

Application Core API type definitions

Application Core API type definitions

Modules

Enumerations

enum app_core_error_state_flag {

SETTING_ERROR_FLAG_BIT = 0
CONNECTION_FAILED_ERROR_FLAG_BIT
SET_NETWORK_SIZE_ERROR_FLAG_BIT
SET_TX_POWER_ERROR_FLAG_BIT
SET_DWELL_INTERVAL_ERROR_FLAG_BIT
SET_MAC_ADDR_ERROR_FLAG_BIT
SET_ALLOW_MAC_ADDR_ERROR_FLAG_BIT
SET_DENY_MAC_ADDR_ERROR_FLAG_BIT
SET_TRUSTED_CERTIFICATE_ERROR_FLAG_BIT
SET_DEVICE_CERTIFICATE_ERROR_FLAG_BIT
SET_DEVICE_PRIVATE_KEY_ERROR_FLAG_BIT
GET_RF_SETTINGS_ERROR_FLAG_BIT

}

Error flag bits enum type definition.

Typedefs

typedef enum app_core_error_state_flag — app_core_error_state_flag_t
Error flag bits enum type definition.

typedef struct current_addr — current_addr_t
Current address storage structure definition.

typedef struct regulation_thresholds — regulation_thresholds_t
Regulation thresholds.

typedef struct app_core_time_stat — app_core_time_stat_t
Application time statistic.

Enumeration Documentation

app_core_error_state_flag

app_core_error_state_flag

Error flag bits enum type definition.

Enumerator	
SETTING_ERROR_FLAG_BIT	Setting Error Flag bit.
CONNECTION_FAILED_ERROR_FLAG_BIT	Connection Failed Error Flag bit.
SET_NETWORK_SIZE_ERROR_FLAG_BIT	Network Size Error Flag bit.
SET_TX_POWER_ERROR_FLAG_BIT	TX Power Error Flag bit.
SET_DWELL_INTERVAL_ERROR_FLAG_BIT	Dwel interval Error flag bit.
SET_MAC_ADDR_ERROR_FLAG_BIT	Setting MAC address Error Flag bit.
SET_ALLOW_MAC_ADDR_ERROR_FLAG_BIT	setting Allow mac address Error Flag bit
SET_DENY_MAC_ADDR_ERROR_FLAG_BIT	setting Deny mac address Error Flag bit
SET_TRUSTED_CERTIFICATE_ERROR_FLAG_BIT	Trusted Certificate Error Flag bit.
SET_DEVICE_CERTIFICATE_ERROR_FLAG_BIT	Device Certificate Error Flag bit.
SET_DEVICE_PRIVATE_KEY_ERROR_FLAG_BIT	Device Private Key Error Flag bit.
GET_RF_SETTINGS_ERROR_FLAG_BIT	Get RF Setting Error Flag bit.

Definition at line 67 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

Typedef Documentation

app_core_error_state_flag_t

typedef enum app_core_error_state_flag app_core_error_state_flag_t

Error flag bits enum type definition.

Definition at line 92 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

current_addr_t

typedef struct current_addr current_addr_t

Current address storage structure definition.

Definition at line 106 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

regulation_thresholds_t

typedef struct regulation_thresholds regulation_thresholds_t

Regulation thresholds.

Definition at line 114 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_core_time_stat_t

```
typedef struct app_core_time_stat app_core_time_stat_t
```

Application time statistic.

Definition at line 130 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

current_addr

Current address storage structure definition.

Public Attributes

in6_addr_t	link_local Link local address.
in6_addr_t	global Global address.
in6_addr_t	border_router Border Router address.
in6_addr_t	primary_parent Primary Parent address.
in6_addr_t	secondary_parent Secondary Parent address.

Public Attribute Documentation

link_local

```
in6_addr_t current_addr::link_local
```

Link local address.

Definition at line 97 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

global

```
in6_addr_t current_addr::global
```

Global address.

Definition at line 99 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

border_router

```
in6_addr_t current_addr::border_router
```

Border Router address.

Definition at line 101 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

primary_parent

> in6_addr_t current_addr::primary_parent

Primary Parent address.

Definition at line 103 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

secondary_parent

> in6_addr_t current_addr::secondary_parent

Secondary Parent address.

Definition at line 105 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

regulation_thresholds

Regulation thresholds.

Public Attributes

int8_t	warning_threshold Warning thresholds.
int8_t	alert_threshold Alert thresholds.

Public Attribute Documentation

warning_threshold

> int8_t regulation_thresholds::warning_threshold

Warning thresholds.

Definition at line 111 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

alert_threshold

> int8_t regulation_thresholds::alert_threshold

Alert thresholds.

Definition at line 113 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

app_core_time_stat

Application time statistic.

Public Attributes

uint64_t	curr_ms Current ms.
uint64_t	connected_ms Last connected ms.
uint64_t	tot_connected_ms Total connected ms.
uint64_t	disconnected_ms Last disconnected ms.
uint64_t	tot_disconnected_ms Total disconnected ms.
uint32_t	conn_cnt Connection counter.

Public Attribute Documentation

curr_ms

```
uint64_t app_core_time_stat::curr_ms
```

Current ms.

Definition at line 119 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

connected_ms

```
uint64_t app_core_time_stat::connected_ms
```

Last connected ms.

Definition at line 121 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

tot_connected_ms

```
uint64_t app_core_time_stat::tot_connected_ms
```

Total connected ms.

Definition at line 123 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

disconnected_ms

```
uint64_t app_core_time_stat::disconnected_ms
```

Last disconnected ms.

Definition at line 125 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

tot_disconnected_ms

```
uint64_t app_core_time_stat::tot_disconnected_ms
```

Total disconnected ms.

Definition at line 127 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

conn_cnt

```
uint32_t app_core_time_stat::conn_cnt
```

Connection counter.

Definition at line 129 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/app_core/sl_wisun_app_core.h

CoAP

CoAP

The Wi-SUN CoAP component provides an application layer implementation on top of the Wi-SUN stack and its socket API. The CoAP protocol is designed as a specialized Internet Application Protocol for constrained devices in lossy networks (see RFC 7252 for additional information). Wi-SUN FAN networks fit this definition. The CoAP implementation relies on the UDP transport layer to send and receive packets.

A number of helper functions are part of the component to help build CoAP payloads, parse CoAP packets, free CoAP packets, and more.

The CoAP Resource Handler service handles registered resources. Resource Discovery request provides an interface to get information about available resources. Resource has more attributes over URI (Uniform Resource Identifier) path, which can be filtered by particular Resource Discovery request.

The CoAP Notification service provides an interface to create and schedule notifications to the remote host. Schedule time and send condition can be customized depending on the application.

To use the CoAP component in your application, add it to your project and initialize it with sl_wisun_coap_init().

Modules

CoAP type definitions

Functions

void	sl_wisun_coap_init(const sl_wisun_coap_tx_callback tx_callback, const sl_wisun_coap_rx_callback rx_callback, const sl_wisun_coap_version_t version) Initialize Wi-SUN CoAP.
__STATIC_INLINE void	sl_wisun_coap_init_default(void) Initialize Wi-SUN CoAP default.
void *	sl_wisun_coap_malloc(uint16_t size) Implement malloc.
void	sl_wisun_coap_free(void *addr) Free Wi-SUN CoAP.
sl_wisun_coap_packet_t *	sl_wisun_coap_parser(uint16_t packet_data_len, uint8_t *packet_data) CoAP parser wrapper function.
uint16_t	sl_wisun_coap_builder_calc_size(const sl_wisun_coap_packet_t *message) CoAP packet calc size wrapper.
int16_t	sl_wisun_coap_builder(uint8_t *dest_buff, const sl_wisun_coap_packet_t *message) CoAP message builder Wi-SUN.
sl_wisun_coap_packet_t *	sl_wisun_coap_build_response(const sl_wisun_coap_packet_t *req, sl_wisun_coap_message_code_t msg_code) Build generic response for request wrapper function.
void	sl_wisun_coap_print_packet(const sl_wisun_coap_packet_t *packet, const bool hex_format) Print CoAP packets and all of attached buffer, payload, token, uri_path.

char *	sl_wisun_coap_get_uri_path_str(const sl_wisun_coap_packet_t *const packet) Prepare URI path string.
__STATIC_INLINE void	sl_wisun_coap_destroy_uri_path_str(char *uri_str) Destroy URI path string.
const sl_wisun_coap_handle_t *	sl_wisun_coap_get_lib_handler(void) Get the library handler pointer from the internal handler structure.
void	sl_wisun_coap_destroy_packet(sl_wisun_coap_packet_t *packet) Destroy packet.
char *	sl_wisun_coap_get_payload_str(const sl_wisun_coap_packet_t *const packet) Prepare payload string.
__STATIC_INLINE void	sl_wisun_coap_destroy_payload_str(char *str) Destroy payload string.

Macros

#define	SL_COAP_SERVICE_LOOP () CoAP Service loop.

Function Documentation

sl_wisun_coap_init

> void sl_wisun_coap_init (const sl_wisun_coap_tx_callback tx_callback, const sl_wisun_coap_rx_callback rx_callback, const sl_wisun_coap_version_t version)

Initialize Wi-SUN CoAP.

Parameters

[in]	tx_callback	TX callback, if it's NULL, the default callback is applied
[in]	rx_callback	RX callback, if it's NULL, the default callback is applied
[in]	version	CoAP version

Set the Wi-SUN CoAP internal descriptor.

Definition at line 172 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_init_default

> __STATIC_INLINE void sl_wisun_coap_init_default (void)

Initialize Wi-SUN CoAP default.

Parameters

N/A		

Initializes the internal descriptor with default values.

Definition at line 180 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_malloc

> void * sl_wisun_coap_malloc (uint16_t size)

Implement malloc.

Parameters

N/A	size	size for malloc

OS-dependent thread-safe implementation. **Returns**

- void* the memory pointer

Definition at line 191 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_free

> void sl_wisun_coap_free (void *addr)

Free Wi-SUN CoAP.

Parameters

N/A	addr	address ptr

OS-dependent thread-safe implementation.

Definition at line 198 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

Referenced by sl_wisun_coap_destroy_payload_str , and sl_wisun_coap_destroy_uri_path_str

sl_wisun_coap_parser

> sl_wisun_coap_packet_t * sl_wisun_coap_parser (uint16_t packet_data_len, uint8_t *packet_data)

CoAP parser wrapper function.

Parameters

[in]	packet_data_len	packet data size
[in]	packet_data	packet data ptr

Used sn_coap_parser **Returns**

- sl_wisun_coap_packet_t* Parsed packet pointer

Definition at line 207 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_builder_calc_size

> uint16_t sl_wisun_coap_builder_calc_size (const sl_wisun_coap_packet_t *message)

CoAP packet calc size wrapper.

Parameters

[in]	message	message ptr

Used sn_coap_builder_calc_needed_packet_data_size. **Returns**

- uint16_t size

Definition at line 216 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_builder

```
int16_t sl_wisun_coap_builder (uint8_t *dest_buff, const sl_wisun_coap_packet_t *message)
```

CoAP message builder Wi-SUN.

Parameters

[out]	dest_buff	destination buffer for raw message
[in]	message	message structure

Used sl_wisun_coap_builder. **Returns**

- int16_t On success bytes of built message, on failure -1 if CoAP header structure is wrong -2 if NULL ptr set as argument

Definition at line 227 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_build_response

```
sl_wisun_coap_packet_t * sl_wisun_coap_build_response (const sl_wisun_coap_packet_t *req,
sl_wisun_coap_message_code_t msg_code)
```

Build generic response for request wrapper function.

Parameters

[in]	req	request
[in]	msg_code	message code to build

Used sn_coap_build_response. **Returns**

- sl_wisun_coap_header_t* built packet ptr on the heap

Definition at line 237 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_print_packet

```
void sl_wisun_coap_print_packet (const sl_wisun_coap_packet_t *packet, const bool hex_format)
```

Print CoAP packets and all of attached buffer, payload, token, uri_path.

Parameters

[in]	packet	packet to print
[in]	hex_format	hex format bool to decide buffer print format

Pretty printer function, with hex format option for buffers

Definition at line 246 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_get_uri_path_str

```
char * sl_wisun_coap_get_uri_path_str (const sl_wisun_coap_packet_t *const packet)
```

Prepare URI path string.

Parameters

[in]	packet	Packet

'\0' terminated string in the heap, it must be freed **Returns**

- char* URI path string, NULL on error

Definition at line `255` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h`

sl_wisun_coap_destroy_uri_path_str

```
__STATIC_INLINE void sl_wisun_coap_destroy_uri_path_str (char *uri_str)
```

Destroy URI path string.

Parameters

[in]	uri_str	URI string ptr

Call free on allocated pointer

Definition at line `262` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h`

sl_wisun_coap_get_lib_handler

```
const sl_wisun_coap_handle_t * sl_wisun_coap_get_lib_handler (void)
```

Get the library handler pointer from the internal handler structure.

Parameters

N/A		

Not thread safe! It is needed only to use Pelion mbed-coap library functions **Returns**

- const sl_wisun_coap_handle_t* pointer to the lib handler

Definition at line `272` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h`

sl_wisun_coap_destroy_packet

```
void sl_wisun_coap_destroy_packet (sl_wisun_coap_packet_t *packet)
```

Destroy packet.

Parameters

N/A	packet	packet

It must be used to avoid memory leaks! Free the all of allocated buffer for packet

Definition at line 280 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_get_payload_str

```
char * sl_wisun_coap_get_payload_str (const sl_wisun_coap_packet_t *const packet)
```

Prepare payload string.

Parameters

[in]	packet	Packet

'\0' terminated string in the heap, it must be freed **Returns**

- char* payload string, NULL on error

Definition at line 288 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_destroy_payload_str

```
__STATIC_INLINE void sl_wisun_coap_destroy_payload_str (char *str)
```

Destroy payload string.

Parameters

[in]	str	String

'\0' terminated string in the heap, it must be freed

Definition at line 295 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

Macro Definition Documentation

SL_COAP_SERVICE_LOOP

```
#define SL_COAP_SERVICE_LOOP
```

Value:

```
()
```

CoAP Service loop.

Definition at line 64 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

CoAP type definitions

CoAP type definitions

Modules

sl_wisun_coap

Typedefs

typedef struct coap_s	sl_wisun_coap_handle_t Wi-SUN CoAP handler for Wi-SUN component.
typedef void *(*	sl_wisun_coap_malloc_t)(uint16_t size) Wi-SUN CoAP malloc function pointer typedef.
typedef void(*	sl_wisun_coap_free_t)(void *mem) Wi-SUN CoAP free function pointer typedef.
typedef coap_version_e	sl_wisun_coap_version_t Wi-SUN CoAP version typedef.
typedef uint8_t(*	sl_wisun_coap_tx_callback)(uint8_t *packet_data, uint16_t packet_data_size, sn_nsdl_addr_s *addr, void *param) Wi-SUN CoAP TX callback function pointer typedef.
typedef int8_t(*	sl_wisun_coap_rx_callback)(sn_coap_hdr_s *header, sn_nsdl_addr_s *addr, void *param) Wi-SUN CoAP RX callback function pointer typedef.
typedef sn_coap_hdr_s	sl_wisun_coap_packet_t Wi-SUN CoAP message typedef.
typedef sn_coap_msg_code_e	sl_wisun_coap_message_code_t Wi-SUN CoAP message code typedef.
typedef sn_coap_msg_type_e	sl_wisun_coap_message_type_t Wi-SUN CoAP message type typedef.
typedef sn_coap_option_numbers_e	sl_wisun_coap_option_num_t Wi-SUN CoAP option number typedef.
typedef sn_coap_options_list_s	sl_wisun_coap_option_list_t Wi-SUN CoAP option list typedef.
typedef struct sl_wisun_coap	sl_wisun_coap_t Wi-SUN CoAP descriptor structure.

Macros

#define SL_WISUN_COAP_URI_PATH_MAX_SIZE (128U)
Maximum size of the URI path string.

Typedef Documentation

sl_wisun_coap_handle_t

```
typedef struct coap_s sl_wisun_coap_handle_t
```

Wi-SUN CoAP handler for Wi-SUN component.

Definition at line 77 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_malloc_t

```
typedef void *(* sl_wisun_coap_malloc_t) (uint16_t size) )(uint16_t size)
```

Wi-SUN CoAP malloc function pointer typedef.

Definition at line 80 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_free_t

```
typedef void(* sl_wisun_coap_free_t) (void *mem) )(void *mem)
```

Wi-SUN CoAP free function pointer typedef.

Definition at line 83 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_version_t

```
typedef coap_version_e sl_wisun_coap_version_t
```

Wi-SUN CoAP version typedef.

Definition at line 86 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_tx_callback

```
typedef uint8_t(* sl_wisun_coap_tx_callback) (uint8_t *packet_data, uint16_t packet_data_size, sn_nsdl_addr_s *addr, void *param) )(uint8_t *packet_data, uint16_t packet_data_size, sn_nsdl_addr_s *addr, void *param)
```

Wi-SUN CoAP TX callback function pointer typedef.

Definition at line 89 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_rx_callback

```
typedef int8_t(* sl_wisun_coap_rx_callback) (sn_coap_hdr_s *header, sn_nsdl_addr_s *addr, void *param) )(sn_coap_hdr_s *header, sn_nsdl_addr_s *addr, void *param)
```

Wi-SUN CoAP RX callback function pointer typedef.

Definition at line 92 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_packet_t

```
typedef sn_coap_hdr_s sl_wisun_coap_packet_t
```

Wi-SUN CoAP message typedef.

Definition at line 95 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_message_code_t

```
typedef sn_coap_msg_code_e sl_wisun_coap_message_code_t
```

Wi-SUN CoAP message code typedef.

Definition at line 98 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_message_type_t

```
typedef sn_coap_msg_type_e sl_wisun_coap_message_type_t
```

Wi-SUN CoAP message type typedef.

Definition at line 101 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_option_num_t

```
typedef sn_coap_option_numbers_e sl_wisun_coap_option_num_t
```

Wi-SUN CoAP option number typedef.

Definition at line 104 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_option_list_t

```
typedef sn_coap_options_list_s sl_wisun_coap_option_list_t
```

Wi-SUN CoAP option list typedef.

Definition at line 107 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap_t

```
typedef struct sl_wisun_coap sl_wisun_coap_t
```

Wi-SUN CoAP descriptor structure.

Definition at line 123 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

Macro Definition Documentation

SL_WISUN_COAP_URI_PATH_MAX_SIZE

```
#define SL_WISUN_COAP_URI_PATH_MAX_SIZE
```

Value:

```
(128U)
```

Maximum size of the URI path string.

Definition at line 74 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

sl_wisun_coap

Wi-SUN CoAP descriptor structure.

Public Attributes

sl_wisun_coap_handle_t *	handler lib handler
sl_wisun_coap_malloc_t	malloc malloc function
sl_wisun_coap_free_t	free free function
sl_wisun_coap_tx_callback	tx_callback TX callback.
sl_wisun_coap_rx_callback	rx_callback RX callback.
sl_wisun_coap_version_t	version CoAP version.

Public Attribute Documentation

handler

```
sl_wisun_coap_handle_t* sl_wisun_coap::handler
```

lib handler

Definition at line 112 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

malloc

```
sl_wisun_coap_malloc_t sl_wisun_coap::malloc
```

malloc function

Definition at line 114 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

free

```
sl_wisun_coap_free_t sl_wisun_coap::free
```

free function

Definition at line 116 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

tx_callback

> sl_wisun_coap_tx_callback sl_wisun_coap::tx_callback

TX callback.

Definition at line 118 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

rx_callback

> sl_wisun_coap_rx_callback sl_wisun_coap::rx_callback

RX callback.

Definition at line 120 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

version

> sl_wisun_coap_version_t sl_wisun_coap::version

CoAP version.

Definition at line 122 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/coap/sl_wisun_coap.h

Ping

Ping

The Ping component implements a ping service based on the ICMPv6 protocol. The component sends ICMPv6 packets, receives the associated responses, and computes the round-trip latency. Using the sl_wisun_ping_request() API, an application emits a single ping packet. To receive the pong response, the application can then call the sl_wisun_ping_response() API.

The sl_wisun_ping() function provides a simple solution to periodically send and receive ping packets. To stop the process, call the sl_wisun_ping_stop() function. A ping test is configurable in size, number of pings sent, and timeout if a response is not received. Ping service supports multicast group addresses, where multiple response can be received from different addresses.

To initialize the component, call sl_wisun_ping_init() function.

Modules

Ping API type definitions

Functions

void	sl_wisun_ping_init(void) Initialize the ping service module.
void	sl_wisun_ping_request(const sl_wisun_ping_info_t *const ping_request) Send a ping request.
void	sl_wisun_ping_response(sl_wisun_ping_info_t *const ping_response) Retrieve a ping response.
sl_status_t	sl_wisun_ping(const sockaddr_in6_t *const remote_addr, const uint16_t packet_count, const uint16_t packet_length, sl_wisun_ping_stat_hnd_t stat_hnd, sl_wisun_ping_req_resp_done_hnd_t req_resp_sent_hnd) Provide a high level ping API.
void	sl_wisun_ping_stop(void) Stop the current ping process.

Function Documentation

sl_wisun_ping_init

```
void sl_wisun_ping_init (void)
```

Initialize the ping service module.

Parameters

N/A		

This function initializes the service thread, mutex, and message queues.

Definition at line 165 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_request

```
void sl_wisun_ping_request (const sl_wisun_ping_info_t *const ping_request)
```

Send a ping request.

Parameters

[in]	ping_request	Ping Request Information

The function sends a single ICMPv6 request.

Definition at line 172 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_response

```
void sl_wisun_ping_response (sl_wisun_ping_info_t *const ping_response)
```

Retrieve a ping response.

Parameters

[out]	ping_response	Ping Response Information

The function retrieves a ping response information.

Definition at line 179 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping

```
sl_status_t sl_wisun_ping (const sockaddr_in6_t *const remote_addr, const uint16_t packet_count, const uint16_t
packet_length, sl_wisun_ping_stat_hnd_t stat_hnd, sl_wisun_ping_req_resp_done_hnd_t req_resp_sent_hnd)
```

Provide a high level ping API.

Parameters

[in]	remote_addr	Remote destination address
[in]	packet_count	Count of packets
[in]	packet_length	ICMPv6 packet length including header
[in]	stat_hnd	Custom statistic handler function
[in]	req_resp_sent_hnd	Request/Response sent handler function

The function provides an interface for periodically sending and receiving ping ICMPv6 packets, and collecting statistic data.

Returns

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL or SL_STATUS_ABORT

Definition at line 192 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_stop

```
void sl_wisun_ping_stop (void)
```

Stop the current ping process.

Parameters

N/A		

Reset request and response queues and send a special ping request with interrupt ping status.

Definition at line `203` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h`

Ping API type definitions

Modules

Typedefs

typedef struct sl_wisun_ping_echo_request — sl_wisun_ping_echo_request_t
Ping echo request packed structure type definitions.

typedef sl_wisun_ping_echo_request_t — sl_wisun_ping_echo_response_t
Ping response type definition.

typedef struct sl_wisun_ping_info — sl_wisun_ping_info_t
Ping info structure type definition.

typedef struct sl_wisun_ping_stat — sl_wisun_ping_stat_t
Statistic ping type definition.

typedef void(* — sl_wisun_ping_stat_hnd_t)(sl_wisun_ping_stat_t *stat)
Ping statistic typedef.

typedef void(* — sl_wisun_ping_req_resp_done_hnd_t)(sl_wisun_ping_info_t *req, sl_wisun_ping_info_t *resp)
Ping request/response sent handler.

Macros

#define — SL_WISUN_PING_MAX_REQUEST_RESPONSE (128U)
Maximum count of ping request/response for message queues.

#define — SL_WISUN_PING_MIN_PACKET_LENGTH (9U)
Minimum packet length with 1 byte payload.

#define — SL_WISUN_PING_MAX_PACKET_LENGTH (SL_WISUN_PING_MIN_PACKET_LENGTH - 1 + SL_WISUN_PING_MAX_PAYLOAD_LENGTH)
Max packet length.

#define — SL_WISUN_PING_TYPE_ECHO_REQUEST (128U)
Ping echo request type field value.

#define — SL_WISUN_PING_TYPE_ECHO_RESPONSE (129U)
Ping echo response type field value.

#define — SL_WISUN_PING_CODE_ECHO_REQUEST (0U)
Ping echo request code field value.

#define SL_WISUN_PING_CODE_ECHO_RESPONSE (0U)
Ping echo response code field value.

#define SL_WISUN_PING_ICMP_PORT (0U)
Dedicated port for ICMPv6 echo messages.

Typedef Documentation

sl_wisun_ping_echo_request_t

```
typedef struct sl_wisun_ping_echo_request sl_wisun_ping_echo_request_t
```

Ping echo request packed structure type definitions.

Definition at line 100 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_echo_response_t

```
typedef sl_wisun_ping_echo_request_t sl_wisun_ping_echo_response_t
```

Ping response type definition.

Definition at line 104 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_info_t

```
typedef struct sl_wisun_ping_info sl_wisun_ping_info_t
```

Ping info structure type definition.

Definition at line 124 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_stat_t

```
typedef struct sl_wisun_ping_stat sl_wisun_ping_stat_t
```

Statistic ping type definition.

Definition at line 142 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_stat_hnd_t

```
typedef void(* sl_wisun_ping_stat_hnd_t) (sl_wisun_ping_stat_t *stat) )(sl_wisun_ping_stat_t *stat)
```

Ping statistic typedef.

Definition at line 145 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_req_resp_done_hnd_t

```
typedef void(* sl_wisun_ping_req_resp_done_hnd_t) (sl_wisun_ping_info_t *req, sl_wisun_ping_info_t *resp) )
(sl_wisun_ping_info_t *req, sl_wisun_ping_info_t *resp)
```

Ping request/response sent handler.

Definition at line 148 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

Macro Definition Documentation

SL_WISUN_PING_MAX_REQUEST_RESPONSE

```
#define SL_WISUN_PING_MAX_REQUEST_RESPONSE
```

Value:

```
(128U)
```

Maximum count of ping request/response for message queues.

Definition at line 62 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

SL_WISUN_PING_MIN_PACKET_LENGTH

```
#define SL_WISUN_PING_MIN_PACKET_LENGTH
```

Value:

```
(9U)
```

Minimum packet length with 1 byte payload.

Definition at line 65 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

SL_WISUN_PING_MAX_PACKET_LENGTH

```
#define SL_WISUN_PING_MAX_PACKET_LENGTH
```

Value:

```
(SL_WISUN_PING_MIN_PACKET_LENGTH - 1 + SL_WISUN_PING_MAX_PAYLOAD_LENGTH)
```

Max packet length.

Definition at line 68 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

SL_WISUN_PING_TYPE_ECHO_REQUEST

```
#define SL_WISUN_PING_TYPE_ECHO_REQUEST
```

Value:

```
(128U)
```

Ping echo request type field value.

Definition at line 71 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

SL_WISUN_PING_TYPE_ECHO_RESPONSE

```
#define SL_WISUN_PING_TYPE_ECHO_RESPONSE
```

Value:

```
(129U)
```

Ping echo response type field value.

Definition at line 74 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

SL_WISUN_PING_CODE_ECHO_REQUEST

```
#define SL_WISUN_PING_CODE_ECHO_REQUEST
```

Value:

```
(0U)
```

Ping echo request code field value.

Definition at line 77 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

SL_WISUN_PING_CODE_ECHO_RESPONSE

```
#define SL_WISUN_PING_CODE_ECHO_RESPONSE
```

Value:

```
(0U)
```

Ping echo response code field value.

Definition at line 80 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

SL_WISUN_PING_ICMP_PORT

```
#define SL_WISUN_PING_ICMP_PORT
```

Value:

```
(0U)
```

Dedicated port for ICMPv6 echo messages.

Definition at line 83 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_echo_request

Ping echo request packed structure type definitions.

Public Attributes

uint8_t	type type
uint8_t	code Code.
uint16_t	checksum Checksum.
uint16_t	identifier Identifier.
uint16_t	sequence_number Sequence number.
uint8_t	payload Payload array.

Public Attribute Documentation

type

```
uint8_t sl_wisun_ping_echo_request::type
```

type

Definition at line 89 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

code

```
uint8_t sl_wisun_ping_echo_request::code
```

Code.

Definition at line 91 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

checksum

```
uint16_t sl_wisun_ping_echo_request::checksum
```

Checksum.

Definition at line 93 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

identifier

> uint16_t sl_wisun_ping_echo_request::identifier

Identifier.

Definition at line `95` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h`

sequence_number

> uint16_t sl_wisun_ping_echo_request::sequence_number

Sequence number.

Definition at line `97` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h`

payload

> uint8_t sl_wisun_ping_echo_request::payload[SL_WISUN_PING_MAX_PAYLOAD_LENGTH]

Payload array.

Definition at line `99` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h`

sl_wisun_ping_info

Ping info structure type definition.

Public Attributes

uint16_t	identifier Identifier.
uint16_t	sequence_number Sequence number.
uint16_t	packet_length ICMPv6 packet length including header.
uint32_t	response_time_ms Response time millisecond.
sockaddr_in6_t	remote_addr Wi-SUN remote address.
uint32_t	start_time_stamp Start time stamp.
uint32_t	stop_time_stamp Stop time stamp.
bool	lost Lost packet flag.

Public Attribute Documentation

identifier

```
uint16_t sl_wisun_ping_info::identifier
```

Identifier.

Definition at line 109 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sequence_number

```
uint16_t sl_wisun_ping_info::sequence_number
```

Sequence number.

Definition at line 111 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

packet_length

uint16_t sl_wisun_ping_info::packet_length

ICMPv6 packet length including header.

Definition at line 113 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

response_time_ms

uint32_t sl_wisun_ping_info::response_time_ms

Response time millisecond.

Definition at line 115 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

remote_addr

sockaddr_in6_t sl_wisun_ping_info::remote_addr

Wi-SUN remote address.

Definition at line 117 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

start_time_stamp

uint32_t sl_wisun_ping_info::start_time_stamp

Start time stamp.

Definition at line 119 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

stop_time_stamp

uint32_t sl_wisun_ping_info::stop_time_stamp

Stop time stamp.

Definition at line 121 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

lost

bool sl_wisun_ping_info::lost

Lost packet flag.

Definition at line 123 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

sl_wisun_ping_stat

Statistic ping type definition.

Public Attributes

sockaddr_in6_t	remote_addr Wi-SUN remote address.
uint16_t	packet_count Packet count.
uint16_t	packet_length Packet length.
uint16_t	lost Lost packet count.
uint32_t	min_time_ms Minimum echo time millisecond.
uint32_t	max_time_ms Maximum echo time millisecond.
uint32_t	avg_time_ms Average echo time millisecond.

Public Attribute Documentation

remote_addr

```
sockaddr_in6_t sl_wisun_ping_stat::remote_addr
```

Wi-SUN remote address.

Definition at line 129 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

packet_count

```
uint16_t sl_wisun_ping_stat::packet_count
```

Packet count.

Definition at line 131 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

packet_length

```
uint16_t sl_wisun_ping_stat::packet_length
```

Packet length.

Definition at line 133 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

lost

```
uint16_t sl_wisun_ping_stat::lost
```

Lost packet count.

Definition at line 135 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

min_time_ms

```
uint32_t sl_wisun_ping_stat::min_time_ms
```

Minimum echo time millisecond.

Definition at line 137 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

max_time_ms

```
uint32_t sl_wisun_ping_stat::max_time_ms
```

Maximum echo time millisecond.

Definition at line 139 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

avg_time_ms

```
uint32_t sl_wisun_ping_stat::avg_time_ms
```

Average echo time millisecond.

Definition at line 141 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ping/sl_wisun_ping.h

iPerf

iPerf

The iPerf component provides an iPerf2-compatible solution to measure UDP throughput. It has a full UDP support and implements a server and client modes, which are capable of sending and receiving packets to measure the bandwidth performance, the inter-arrival jitter, and packet loss.

The component relies on different functions to help configure and run your iPerf test. It allows you to set up your server and client port, your preferred bandwidth, number of packets, remote peer address, and so on. At the end of every iPerf test, an iPerf report is output. Multicast target measurement is supported also. The component can be used with any network stack because it has a portable custom network interface.

To use the iPerf component in your application, add it to your project and initialize it with sl_iperf_service_init().

Modules

iPerf type definitions

Functions

void sl_iperf_service_init(void)
Initialize the iPerf service.

void sl_iperf_test_init(sl_iperf_test_t *const test, sl_iperf_mode_t mode, sl_iperf_protocol_t protocol)
Initialize the iPerf test.

void sl_iperf_test_set_default_logger(sl_iperf_test_t *const test)
Set the default internal logger for the test descriptor.

void sl_iperf_test_set_default_buff(sl_iperf_test_t *const test)
Set the default internal test buffer.

bool sl_iperf_test_add(sl_iperf_test_t *const test)
Add the iPerf test to the execution queue.

bool sl_iperf_test_get(sl_iperf_test_t *const test, const uint32_t timeout_ms)
Get the iPerf test from the result queue.

void sl_iperf_test_udp_client(sl_iperf_test_t *test)
Execute the iPerf UDP client test.

void sl_iperf_test_udp_server(sl_iperf_test_t *test)
Execute the iPerf UDP server test.

Function Documentation

sl_iperf_service_init

```
void sl_iperf_service_init (void)
```

Initialize the iPerf service.

Parameters

N/A		

Init OS objects and default contents

Definition at line 108 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

sl_iperf_test_init

```
void sl_iperf_test_init (sl_iperf_test_t *const test, sl_iperf_mode_t mode, sl_iperf_protocol_t protocol)
```

Initialize the iPerf test.

Parameters

[inout]	test	Test descriptor
[in]	mode	Mode
[in]	protocol	Protocol

Initialize a test descriptor with default content

Definition at line 117 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

sl_iperf_test_set_default_logger

```
void sl_iperf_test_set_default_logger (sl_iperf_test_t *const test)
```

Set the default internal logger for the test descriptor.

Parameters

[out]	test	Test descriptor

Helper function

Definition at line 126 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

sl_iperf_test_set_default_buff

```
void sl_iperf_test_set_default_buff (sl_iperf_test_t *const test)
```

Set the default internal test buffer.

Parameters

[out]	test	Test descriptor

Helper function

Definition at line 133 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

sl_iperf_test_add

```
bool sl_iperf_test_add (sl_iperf_test_t *const test)
```

Add the iPerf test to the execution queue.

Parameters

[in]	test	Test descriptor

Add test to the input messagequeue. **Returns**

- true On Success
- false On Failure

Definition at line 143 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

sl_iperf_test_get

```
bool sl_iperf_test_get (sl_iperf_test_t *const test, const uint32_t timeout_ms)
```

Get the iPerf test from the result queue.

Parameters

[out]	test	Destination test descriptor
[in]	timeout_ms	Timeout for getting test from messagequeue

Get the test from the output messagequeue. **Returns**

- true On Success
- false On Failure

Definition at line 153 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

sl_iperf_test_udp_client

```
void sl_iperf_test_udp_client (sl_iperf_test_t *test)
```

Execute the iPerf UDP client test.

Parameters

[inout]	test	Test descriptor

iPerf UDP client test.

Definition at line 66 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_udp_clnt.h

sl_iperf_test_udp_server

```
void sl_iperf_test_udp_server (sl_iperf_test_t *test)
```

Execute the iPerf UDP server test.

Parameters

[inout]	test	Test descriptor

iPerf UDP server test.

Definition at line 66 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_udp_srv.h

iPerf type definitions

iPerf type definitions

Modules

- sl_iperf_opt
- sl_iperf_stats
- sl_iperf_conn
- sl_iperf_log_str_buff
- sl_iperf_log
- sl_iperf_test
- sl_iperf_udp_datagram
- sl_iperf_udp_srv_hdr
- sl_iperf_udp_clnt_hdr_v1
- sl_iperf_clnt_hdr_ext
- sl_iperf_clnt_hdr_isoch_payload
- sl_iperf_clnt_hdr_ext_starttime_fq
- sl_iperf_clnt_hdr_ext_isoch_settings
- sl_iperf_udp_clnt_hdr

Enumerations

enum sl_iperf_mode {

SL_IPERF_MODE_SERVER
SL_IPERF_MODE_CLIENT

}

Iperf mode.

enum sl_iperf_status {

SL_IPERF_TEST_STATUS_FREE
SL_IPERF_TEST_STATUS_QUEUED
SL_IPERF_TEST_STATUS_RUNNING
SL_IPERF_TEST_STATUS_DONE
SL_IPERF_TEST_STATUS_ERR

}

Test status type definition.

enum sl_iperf_opt_bw_format {

SL_IPERF_OPT_BW_FORMAT_BITS_PER_SEC
SL_IPERF_OPT_BW_FORMAT_KBITS_PER_SEC
SL_IPERF_OPT_BW_FORMAT_MBITS_PER_SEC
SL_IPERF_OPT_BW_FORMAT_GBITS_PER_SEC
SL_IPERF_OPT_BW_FORMAT_BYTES_PER_SEC
SL_IPERF_OPT_BW_FORMAT_KBYTES_PER_SEC
SL_IPERF_OPT_BW_FORMAT_MBYTES_PER_SEC
SL_IPERF_OPT_BW_FORMAT_GBYTES_PER_SEC

}

Bandwidth format enumeration type definition.

enum sl_iperf_err {

SL_IPERF_ERR_NONE
SL_IPERF_ERR_NETWORK_CONNECTION
SL_IPERF_ERR_SERVER_SOCK_BIND
SL_IPERF_ERR_SERVER_SOCK_OPEN
SL_IPERF_ERR_SERVER_SOCK_CLOSE
SL_IPERF_ERR_SERVER_SOCK_LISTEN
SL_IPERF_ERR_SERVER_SOCK_ACCEPT
SL_IPERF_ERR_SERVER_SOCK_RX
SL_IPERF_ERR_SERVER_SOCK_WIN_SIZE
SL_IPERF_ERR_CLIENT_SOCK_OPEN
SL_IPERF_ERR_CLIENT_SOCK_BIND
SL_IPERF_ERR_CLIENT_SOCK_CONN
SL_IPERF_ERR_CLIENT_SOCK_TX
SL_IPERF_ERR_CLIENT_SOCK_TX_INV_ARG
SL_IPERF_ERR_CLIENT_SOCK_CLOSE

}

iPerf error enumeration type definition

Typedefs

typedef enum sl_iperf_mode — sl_iperf_mode_t
Iperf mode.

typedef uint16_t — sl_iperf_test_id_t
Test ID type definition.

typedef enum sl_iperf_status — sl_iperf_status_t
Test status type definition.

typedef enum sl_iperf_opt_bw_format — sl_iperf_opt_bw_format
Bandwidth format enumeration type definition.

typedef struct sl_iperf_opt — sl_iperf_opt_t
iPerf test option type definition

typedef struct sl_iperf_stats — sl_iperf_stats_t
iPerf statistic data type definition

typedef struct sl_iperf_conn — sl_iperf_conn_t
iPerf connection descriptor type definition

typedef enum sl_iperf_err — sl_iperf_error_t
iPerf error enumeration type definition

typedef struct sl_iperf_log_str_buff — sl_iperf_log_str_buff_t
Log string buffer type definition.

typedef struct sl_iperf_log — sl_iperf_log_t
iPerf log type definition

typedef int32_t(* — sl_iperf_log_print_t)(sl_iperf_log_t *const log, const char *format,...)
Printer function type definition.

typedef struct sl_iperf_test — sl_iperf_test_t
iPerf test descriptor

typedef void(* — sl_iperf_test_callback_t)(sl_iperf_test_t *)
iPerf Test callback type definition

typedef struct sl_iperf_udp_datagram — sl_iperf_udp_datagram_t
iPerf UDP datagram structure type definition

typedef struct sl_iperf_udp_srv_hdr — sl_iperf_udp_srv_hdr_t
iPerf server header

typedef struct sl_iperf_udp_clnt_hdr_v1 — sl_iperf_clnt_hdr_v1_t
iPerf CLient Header v1

typedef struct sl_iperf_clnt_hdr_ext — sl_iperf_clnt_hdr_ext_t
iPerf Client Header extended

typedef struct sl_iperf_clnt_hdr_isoch_payload — sl_iperf_clnt_hdr_isoch_payload_t
iPerf Client Isochronus payload

typedef struct sl_iperf_clnt_hdr_ext_starttime_fq — sl_iperf_clnt_hdr_ext_starttime_fq_t
iPerf Client header extended FQ start time

typedef struct sl_iperf_clnt_hdr_ext_isoch_settings — sl_iperf_clnt_hdr_ext_isoch_settings_t
iPerf Client Isochronus settings

typedef struct sl_iperf_udp_clnt_hdr — sl_iperf_udp_clnt_hdr_t
iPerf Client UDP header

Macros

#define — SL_IPERF_UDP_SERVER_FIN_ACK_SIZE (128U)
iPerf UDP server final ack size

#define — SL_IPERF_HEADER_VERSION1 (0x80000000UL)
iPerf header version1 mask

#define — SL_IPERF_HEADER_VERSION2 (0x04000000UL)
iPerf header version2 mask

#define — SL_IPERF_HEADER_EXTEND (0x40000000UL)
iPerf header extended mask

#define — SL_IPERF_HEADER_SEQNO64B (0x08000000UL)
iPerf header seqno 64bit mask

#define SL_IPERF_HEADER_UDPTEST (0x20000000UL)
iPerf header UDP test mask

#define SL_IPERF_HEADER_EPOCH_START (0x00001000UL)
iPerf header epoch start mask

#define SL_IPERF_HEADER_TRIPTIME (0x00000010UL)
iPerf header triptime mask

#define SL_IPERF_HEADER_TIME_MODE (0x80000000UL)
iPerf header time mode mask

#define SL_IPERF_SERVER_UDP_TX_FINACK_COUNT (10U)
iPerf Server TX FINACK max count to retry

#define SL_IPERF_IP_STR_BUFF_LEN (40U)
iPerf ip address string max length definition

Enumeration Documentation

sl_iperf_mode

> sl_iperf_mode

Iperf mode.

Enumerator	
SL_IPERF_MODE_SERVER	Server mode.
SL_IPERF_MODE_CLIENT	Client mode.

Definition at line 65 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_status

> sl_iperf_status

Test status type definition.

Enumerator	
SL_IPERF_TEST_STATUS_FREE	Test unused.
SL_IPERF_TEST_STATUS_QUEUED	Test queued.
SL_IPERF_TEST_STATUS_RUNNING	Test running.
SL_IPERF_TEST_STATUS_DONE	Test done with no error.
SL_IPERF_TEST_STATUS_ERR	Test done with error.

Definition at line 76 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_opt_bw_format

> sl_iperf_opt_bw_format

Bandwidth format enumeration type definition.

Enumerator

SL_IPERF_OPT_BW_FORMAT_BITS_PER_SEC	Bits/sec format.
SL_IPERF_OPT_BW_FORMAT_KBITS_PER_SEC	KBits/sec format.
SL_IPERF_OPT_BW_FORMAT_MBITS_PER_SEC	MBits/sec format.
SL_IPERF_OPT_BW_FORMAT_GBITS_PER_SEC	GBits/sec format.
SL_IPERF_OPT_BW_FORMAT_BYTES_PER_SEC	Bytes/sec format.
SL_IPERF_OPT_BW_FORMAT_KBYTES_PER_SEC	KBytes/sec format.
SL_IPERF_OPT_BW_FORMAT_MBYTES_PER_SEC	MBytes/sec format.
SL_IPERF_OPT_BW_FORMAT_GBYTES_PER_SEC	GBytes/sec format.

Definition at line 93 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_err

```
sl_iperf_err
```

iPerf error enumeration type definition

Enumerator	
SL_IPERF_ERR_NONE	No error.
SL_IPERF_ERR_NETWORK_CONNECTION	Network connection error.
SL_IPERF_ERR_SERVER_SOCK_BIND	Server socket bind error.
SL_IPERF_ERR_SERVER_SOCK_OPEN	Server socket open error.
SL_IPERF_ERR_SERVER_SOCK_CLOSE	Server socket close error.
SL_IPERF_ERR_SERVER_SOCK_LISTEN	Server socket listen error.
SL_IPERF_ERR_SERVER_SOCK_ACCEPT	Server socket accept error.
SL_IPERF_ERR_SERVER_SOCK_RX	Server socket RX error.
SL_IPERF_ERR_SERVER_SOCK_WIN_SIZE	Server socket windows size error.
SL_IPERF_ERR_CLIENT_SOCK_OPEN	Client socket open error.
SL_IPERF_ERR_CLIENT_SOCK_BIND	Client socket bind error.
SL_IPERF_ERR_CLIENT_SOCK_CONN	Client socket connect error.
SL_IPERF_ERR_CLIENT_SOCK_TX	Client socket TX error.
SL_IPERF_ERR_CLIENT_SOCK_TX_INV_ARG	Client socket TX invalid argument error.
SL_IPERF_ERR_CLIENT_SOCK_CLOSE	Client socket socket close error.

Definition at line 213 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

Typedef Documentation

sl_iperf_mode_t

```
typedef enum sl_iperf_mode sl_iperf_mode_t
```

Iperf mode.

Definition at line 70 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_test_id_t

```
typedef uint16_t sl_iperf_test_id_t
```

Test ID type definition.

Definition at line 73 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_status_t

```
typedef enum sl_iperf_status sl_iperf_status_t
```

Test status type definition.

Definition at line 87 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_opt_bw_format

```
typedef enum sl_iperf_opt_bw_format sl_iperf_opt_bw_format
```

Bandwidth format enumeration type definition.

Definition at line 110 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_opt_t

```
typedef struct sl_iperf_opt sl_iperf_opt_t
```

iPerf test option type definition

Definition at line 140 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_stats_t

```
typedef struct sl_iperf_stats sl_iperf_stats_t
```

iPerf statistic data type definition

Definition at line 192 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_conn_t

```
typedef struct sl_iperf_conn sl_iperf_conn_t
```

iPerf connection descriptor type definition

Definition at line 210 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_error_t

```
typedef enum sl_iperf_err sl_iperf_error_t
```

iPerf error enumeration type definition

Definition at line 244 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_log_str_buff_t

```
typedef struct sl_iperf_log_str_buff sl_iperf_log_str_buff_t
```

Log string buffer type definition.

Definition at line 254 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_log_t

```
typedef struct sl_iperf_log sl_iperf_log_t
```

iPerf log type definition

Definition at line 268 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_log_print_t

```
typedef int32_t(* sl_iperf_log_print_t) (sl_iperf_log_t *const log, const char *format,...) )(sl_iperf_log_t *const log, const char *format,...)
```

Printer function type definition.

Definition at line 271 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_test_t

```
typedef struct sl_iperf_test sl_iperf_test_t
```

iPerf test descriptor

Definition at line 291 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_test_callback_t

```
typedef void(* sl_iperf_test_callback_t) (sl_iperf_test_t *) )(sl_iperf_test_t *)
```

iPerf Test callback type definition

Definition at line 294 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_udp_datagram_t

```
typedef struct sl_iperf_udp_datagram sl_iperf_udp_datagram_t
```

iPerf UDP datagram structure type definition

Definition at line 309 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_udp_srv_hdr_t

```
typedef struct sl_iperf_udp_srv_hdr sl_iperf_udp_srv_hdr_t
```

iPerf server header

Definition at line 335 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_v1_t

```
typedef struct sl_iperf_udp_clnt_hdr_v1 sl_iperf_clnt_hdr_v1_t
```

iPerf CLient Header v1

Definition at line 351 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_ext_t

```
typedef struct sl_iperf_clnt_hdr_ext sl_iperf_clnt_hdr_ext_t
```

iPerf Client Header extended

Definition at line 377 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_isoch_payload_t

```
typedef struct sl_iperf_clnt_hdr_isoch_payload sl_iperf_clnt_hdr_isoch_payload_t
```

iPerf Client Isochronus payload

Definition at line 397 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_ext_starttime_fq_t

```
typedef struct sl_iperf_clnt_hdr_ext_starttime_fq sl_iperf_clnt_hdr_ext_starttime_fq_t
```

iPerf Client header extended FQ start time

Definition at line 411 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_ext_isoch_settings_t

```
typedef struct sl_iperf_clnt_hdr_ext_isoch_settings sl_iperf_clnt_hdr_ext_isoch_settings_t
```

iPerf Client Isochronus settings

Definition at line 431 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_udp_clnt_hdr_t

```
typedef struct sl_iperf_udp_clnt_hdr sl_iperf_udp_clnt_hdr_t
```

iPerf Client UDP header

Definition at line 447 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

Macro Definition Documentation

SL_IPERF_UDP_SERVER_FIN_ACK_SIZE

```
#define SL_IPERF_UDP_SERVER_FIN_ACK_SIZE
```

Value:

```
(128U)
```

iPerf UDP server final ack size

Definition at line 65 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_VERSION1

```
#define SL_IPERF_HEADER_VERSION1
```

Value:

```
(0x80000000UL)
```

iPerf header version1 mask

Definition at line 68 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_VERSION2

```
#define SL_IPERF_HEADER_VERSION2
```

Value:

```
(0x04000000UL)
```

iPerf header version2 mask

Definition at line 71 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_EXTEND

```
#define SL_IPERF_HEADER_EXTEND
```

Value:

```
(0x40000000UL)
```

iPerf header extended mask

Definition at line 74 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_SEQNO64B

```
#define SL_IPERF_HEADER_SEQNO64B
```

Value:

```
(0x08000000UL)
```

iPerf header seqno 64bit mask

Definition at line 77 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_UDPTEST

```
#define SL_IPERF_HEADER_UDPTEST
```

Value:

```
(0x20000000UL)
```

iPerf header UDP test mask

Definition at line 80 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_EPOCH_START

```
#define SL_IPERF_HEADER_EPOCH_START
```

Value:

```
(0x00001000UL)
```

iPerf header epoch start mask

Definition at line 83 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_TRIPTIME

```
#define SL_IPERF_HEADER_TRIPTIME
```

Value:

```
(0x00000010UL)
```

iPerf header triptime mask

Definition at line 86 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_HEADER_TIME_MODE

```
#define SL_IPERF_HEADER_TIME_MODE
```

Value:

```
(0x80000000UL)
```

iPerf header time mode mask

Definition at line 89 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_SERVER_UDP_TX_FINACK_COUNT

```
#define SL_IPERF_SERVER_UDP_TX_FINACK_COUNT
```

Value:

```
(10U)
```

iPerf Server TX FINACK max count to retry

Definition at line 92 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf.h

SL_IPERF_IP_STR_BUFF_LEN

```
#define SL_IPERF_IP_STR_BUFF_LEN
```

Value:

```
(40U)
```

iPerf ip address string max length definition

Definition at line 90 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_opt

iPerf test option type definition

Public Attributes

sl_iperf_mode_t	mode Server or client mode.
sl_iperf_protocol_t	protocol UDP or TCP protocol.
uint16_t	port Server or client port.
char	remote_addr Remote address for TX.
uint32_t	bandwidth expected bandwidth in bits/sec
uint16_t	packet_nbr Nbr of packets to tx.
uint16_t	buf_len Buf len to tx or rx.
uint16_t	duration_ms Time in sec to tx.
uint16_t	win_size Win size to tx or rx.
bool	persistent Server in persistent mode.
uint16_t	interval_ms Interval (ms) between bandwidth update.
sl_iperf_opt_bw_format	bw_format Bandwidth format.
bool	multicast Join multicast group (address is stored in 'remote_addr')

Public Attribute Documentation

mode

```
sl_iperf_mode_t sl_iperf_opt::mode
```

Server or client mode.

Definition at line 115 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

protocol

```
sl_iperf_protocol_t sl_iperf_opt::protocol
```

UDP or TCP protocol.

Definition at line 117 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

port

```
uint16_t sl_iperf_opt::port
```

Server or client port.

Definition at line 119 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

remote_addr

```
char sl_iperf_opt::remote_addr[SL_IPERF_IP_STR_BUFF_LEN]
```

Remote address for TX.

Definition at line 121 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

bandwidth

```
uint32_t sl_iperf_opt::bandwidth
```

expected bandwidth in bits/sec

Definition at line 123 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

packet_nbr

```
uint16_t sl_iperf_opt::packet_nbr
```

Nbr of packets to tx.

Definition at line 125 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

buf_len

```
uint16_t sl_iperf_opt::buf_len
```

Buf len to tx or rx.

Definition at line 127 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

duration_ms

```
uint16_t sl_iperf_opt::duration_ms
```

Time in sec to tx.

Definition at line 129 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

win_size

```
uint16_t sl_iperf_opt::win_size
```

Win size to tx or rx.

Definition at line 131 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

persistent

```
bool sl_iperf_opt::persistent
```

Server in persistent mode.

Definition at line 133 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

interval_ms

```
uint16_t sl_iperf_opt::interval_ms
```

Interval (ms) between bandwidth update.

Definition at line 135 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

bw_format

```
sl_iperf_opt_bw_format sl_iperf_opt::bw_format
```

Bandwidth format.

Definition at line 137 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

multicast

```
bool sl_iperf_opt::multicast
```

Join multicast group (address is stored in 'remote_addr')

Definition at line 139 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_stats

iPerf statistic data type definition

Public Attributes

uint32_t	nbr_calls Nbr of I/O sys calls.
uint32_t	bytes Nbr of bytes rx'd or tx'd on net.
uint32_t	tot_packets Nbr of rx'd or tx'd and lost packets.
uint32_t	nbr_rcv_snt_packets Nbr of rx'd or tx'd packets.
uint32_t	errs Nbr of rx or tx errs.
uint32_t	transitory_error_cnts Nbr of transitory err.
uint32_t	last_recv_pkt_cnt Last received packets (for update)
sl_iperf_ts_ms_t	ts_curr_recv_ms Current received packet timestamp.
sl_iperf_ts_ms_t	ts_prev_recv_ms Previous received packt timestamp.
sl_iperf_ts_ms_t	ts_curr_sent_ms Current sent packet timestamp.
sl_iperf_ts_ms_t	ts_prev_sent_ms Previous sent packet timestamp.
int64_t	udp_jitter UDP jitter.
int32_t	udp_rx_last_pkt Prev pkt ID rx'd.
uint32_t	udp_lost_pkt Nbr of UDP pkt lost.
uint32_t	udp_out_of_order Nbr of pkt rx'd out of order.
uint32_t	udp_dup_pkt Nbr of pkt ID rx'd more than once.
bool	udp_async_error First UDP pkt rx'd.

	bool	end_err Err with UDP FIN or FINACK.
	sl_iperf_ts_ms_t	ts_start_ms Start timestamp (ms).
	sl_iperf_ts_ms_t	ts_end_ms End timestamp (ms).
	uint32_t	bandwidth Rx or Tx cur bandwidth in bits/s.
	uint32_t	finack_tot_len Total length of received bytes in Final ACK.
	sl_iperf_ts_ms_t	finack_duration_ms Time duration in Final ACK.
	uint32_t	finack_pkt Packet count in Final ACK.

Public Attribute Documentation

nbr_calls

```
uint32_t sl_iperf_stats::nbr_calls
```

Nbr of I/O sys calls.

Definition at line 145 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

bytes

```
uint32_t sl_iperf_stats::bytes
```

Nbr of bytes rx'd or tx'd on net.

Definition at line 147 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

tot_packets

```
uint32_t sl_iperf_stats::tot_packets
```

Nbr of rx'd or tx'd and lost packets.

Definition at line 149 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

nbr_rcv_snt_packets

```
uint32_t sl_iperf_stats::nbr_rcv_snt_packets
```

Nbr of rx'd or tx'd packets.

Definition at line 151 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

errs

```
uint32_t sl_iperf_stats::errs
```

Nbr of rx or tx errs.

Definition at line 153 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

transitory_error_cnts

```
uint32_t sl_iperf_stats::transitory_error_cnts
```

Nbr of transitory err.

Definition at line 155 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

last_recv_pkt_cnt

```
uint32_t sl_iperf_stats::last_recv_pkt_cnt
```

Last received packets (for update)

Definition at line 157 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

ts_curr_recv_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_curr_recv_ms
```

Current received packet timestamp.

Definition at line 159 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

ts_prev_recv_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_prev_recv_ms
```

Previous received packt timestamp.

Definition at line 161 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

ts_curr_sent_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_curr_sent_ms
```

Current sent packet timestamp.

Definition at line 163 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

ts_prev_sent_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_prev_sent_ms
```

Previous sent packet timestamp.

Definition at line 165 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

udp_jitter

```
int64_t sl_iperf_stats::udp_jitter
```

UDP jitter.

Definition at line 167 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

udp_rx_last_pkt

```
int32_t sl_iperf_stats::udp_rx_last_pkt
```

Prev pkt ID rx'd.

Definition at line 169 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

udp_lost_pkt

```
uint32_t sl_iperf_stats::udp_lost_pkt
```

Nbr of UDP pkt lost.

Definition at line 171 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

udp_out_of_order

```
uint32_t sl_iperf_stats::udp_out_of_order
```

Nbr of pkt rx'd out of order.

Definition at line 173 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

udp_dup_pkt

```
uint32_t sl_iperf_stats::udp_dup_pkt
```

Nbr of pkt ID rx'd more than once.

Definition at line 175 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

udp_async_error

```
bool sl_iperf_stats::udp_async_error
```

First UDP pkt rx'd.

Definition at line 177 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

end_err

```
bool sl_iperf_stats::end_err
```

Err with UDP FIN or FINACK.

Definition at line 179 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

ts_start_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_start_ms
```

Start timestamp (ms).

Definition at line 181 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

ts_end_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_end_ms
```

End timestamp (ms).

Definition at line 183 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

bandwidth

```
uint32_t sl_iperf_stats::bandwidth
```

Rx or Tx cur bandwidth in bits/s.

Definition at line 185 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

finack_tot_len

```
uint32_t sl_iperf_stats::finack_tot_len
```

Total length of received bytes in Final ACK.

Definition at line 187 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

finack_duration_ms

> sl_iperf_ts_ms_t sl_iperf_stats::finack_duration_ms

Time duration in Final ACK.

Definition at line 189 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

finack_pkt

> uint32_t sl_iperf_stats::finack_pkt

Packet count in Final ACK.

Definition at line 191 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_conn

iPerf connection descriptor type definition

Public Attributes

int32_t	socket_id local socket id
int32_t	socket_id_clnt Accepted sock used by TCP server to rx.
sl_iperf_socket_addr_t	srv_addr Server sock addr IP.
sl_iperf_socket_addr_t	clnt_addr Client sock addr IP.
bool	run Server (rx'd) or client (tx'd) started.
uint8_t *	buff Buffer ptr to receive/transmit.
size_t	buff_size RX/TX buffer transmit.

Public Attribute Documentation

socket_id

```
int32_t sl_iperf_conn::socket_id
```

local socket id

Definition at line 197 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

socket_id_clnt

```
int32_t sl_iperf_conn::socket_id_clnt
```

Accepted sock used by TCP server to rx.

Definition at line 199 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

srv_addr

```
sl_iperf_socket_addr_t sl_iperf_conn::srv_addr
```

Server sock addr IP.

Definition at line 201 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

clnt_addr

```
sl_iperf_socket_addr_t sl_iperf_conn::clnt_addr
```

Client sock addr IP.

Definition at line 203 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

run

```
bool sl_iperf_conn::run
```

Server (rx'd) or client (tx'd) started.

Definition at line 205 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

buff

```
uint8_t* sl_iperf_conn::buff
```

Buffer ptr to receive/transmit.

Definition at line 207 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

buff_size

```
size_t sl_iperf_conn::buff_size
```

RX/TX buffer transmit.

Definition at line 209 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_log_str_buff

Log string buffer type definition.

Public Attributes

char *	pos	Position ptr.
char *	buff	Buff ptr.
size_t	size	Size.

Public Attribute Documentation

pos

```
char* sl_iperf_log_str_buff::pos
```

Position ptr.

Definition at line 249 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

buff

```
char* sl_iperf_log_str_buff::buff
```

Buff ptr.

Definition at line 251 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

size

```
size_t sl_iperf_log_str_buff::size
```

Size.

Definition at line 253 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_log

iPerf log type definition

Public Attributes

bool	colored Colored.
bool	buffered Buffered.
int32_t	last_res Last result.
int32_t(*	print Stdout and buffer printer.
sl_iperf_log_str_buff_t	buff Buff string instance.

Public Attribute Documentation

colored

```
bool sl_iperf_log::colored
```

Colored.

Definition at line 259 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

buffered

```
bool sl_iperf_log::buffered
```

Buffered.

Definition at line 261 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

last_res

```
int32_t sl_iperf_log::last_res
```

Last result.

Definition at line 263 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

print

```
int32_t(* sl_iperf_log::print) (struct sl_iperf_log *const log, const char *format,...)
```

Stdout and buffer printer.

Definition at line `265` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h`

buff

```
sl_iperf_log_str_buff_t sl_iperf_log::buff
```

Buff string instance.

Definition at line `267` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h`

sl_iperf_test

iPerf test descriptor

Public Attributes

sl_iperf_test_id_t	id Test ID.
sl_iperf_status_t	status Status.
sl_iperf_error_t	err Error.
sl_iperf_opt_t	opt Options.
sl_iperf_stats_t	statistic Statistics.
sl_iperf_conn_t	conn Connection.
void(*	cb Callback.
sl_iperf_log_t *	log Log object ptr.

Public Attribute Documentation

id

```
sl_iperf_test_id_t sl_iperf_test::id
```

Test ID.

Definition at line 276 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

status

```
sl_iperf_status_t sl_iperf_test::status
```

Status.

Definition at line 278 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

err

```
sl_iperf_error_t sl_iperf_test::err
```

Error.

Definition at line 280 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

opt

> sl_iperf_opt_t sl_iperf_test::opt

Options.

Definition at line 282 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

statistic

> sl_iperf_stats_t sl_iperf_test::statistic

Statistics.

Definition at line 284 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

conn

> sl_iperf_conn_t sl_iperf_test::conn

Connection.

Definition at line 286 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

cb

> void(* sl_iperf_test::cb) (struct sl_iperf_test *)

Callback.

Definition at line 288 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

log

> sl_iperf_log_t* sl_iperf_test::log

Log object ptr.

Definition at line 290 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_udp_datagram

iPerf UDP datagram structure type definition

Public Attributes

int32_t	id Packet id.
uint32_t	time_var_sec Time variable for sec.
uint32_t	time_var_usec Time variable for usec.
int32_t	id2 Packet id.

Public Attribute Documentation

id

```
int32_t sl_iperf_udp_datagram::id
```

Packet id.

Definition at line 302 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

time_var_sec

```
uint32_t sl_iperf_udp_datagram::time_var_sec
```

Time variable for sec.

Definition at line 304 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

time_var_usec

```
uint32_t sl_iperf_udp_datagram::time_var_usec
```

Time variable for usec.

Definition at line 306 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

id2

> int32_t sl_iperf_udp_datagram::id2

Packet id.

Definition at line 308 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_udp_srv_hdr

iPerf server header

Public Attributes

sl_iperf_udp_datagram_t	dtg Udp datagram.
int32_t	flags Server flag.
uint32_t	tot_len_u Tot bytes rx'd hi part.
uint32_t	tot_len_l Tot bytes rx'd low part.
uint32_t	stop_sec Stop time in sec.
uint32_t	stop_usec Stop time in usec.
uint32_t	lost_pkt_cnt Lost pkt cnt.
uint32_t	out_of_order_cnt Rx pkt out of order cnt.
uint32_t	packet_cnt Packet count.
uint32_t	jitter_sec Jitter hi.
uint32_t	jitter_usec Jitter low.

Public Attribute Documentation

dtg

```
sl_iperf_udp_datagram_t sl_iperf_udp_srv_hdr::dtg
```

Udp datagram.

Definition at line 314 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

flags

```
int32_t sl_iperf_udp_srv_hdr::flags
```

Server flag.

Definition at line 316 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

tot_len_u

```
uint32_t sl_iperf_udp_srv_hdr::tot_len_u
```

Tot bytes rx'd hi part.

Definition at line 318 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

tot_len_l

```
uint32_t sl_iperf_udp_srv_hdr::tot_len_l
```

Tot bytes rx'd low part.

Definition at line 320 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

stop_sec

```
uint32_t sl_iperf_udp_srv_hdr::stop_sec
```

Stop time in sec.

Definition at line 322 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

stop_usec

```
uint32_t sl_iperf_udp_srv_hdr::stop_usec
```

Stop time in usec.

Definition at line 324 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

lost_pkt_cnt

```
uint32_t sl_iperf_udp_srv_hdr::lost_pkt_cnt
```

Lost pkt cnt.

Definition at line 326 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

out_of_order_cnt

```
uint32_t sl_iperf_udp_srv_hdr::out_of_order_cnt
```

Rx pkt out of order cnt.

Definition at line 328 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

packet_cnt

```
uint32_t sl_iperf_udp_srv_hdr::packet_cnt
```

Packet count.

Definition at line 330 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

jitter_sec

```
uint32_t sl_iperf_udp_srv_hdr::jitter_sec
```

Jitter hi.

Definition at line 332 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

jitter_usec

```
uint32_t sl_iperf_udp_srv_hdr::jitter_usec
```

Jitter low.

Definition at line 334 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_udp_clnt_hdr_v1

iPerf CLient Header v1

Public Attributes

int32_t	flags Flags.
int32_t	num_threads Number of threads.
int32_t	port Port.
int32_t	buf_len Buffer Length.
int32_t	win_band Win band.
int32_t	amount Amount.

Public Attribute Documentation

flags

```
int32_t sl_iperf_udp_clnt_hdr_v1::flags
```

Flags.

Definition at line 340 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

num_threads

```
int32_t sl_iperf_udp_clnt_hdr_v1::num_threads
```

Number of threads.

Definition at line 342 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

port

```
int32_t sl_iperf_udp_clnt_hdr_v1::port
```

Port.

Definition at line 344 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

buf_len

```
int32_t sl_iperf_udp_clnt_hdr_v1::buf_len
```

Buffer Length.

Definition at line 346 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

win_band

```
int32_t sl_iperf_udp_clnt_hdr_v1::win_band
```

Win band.

Definition at line 348 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

amount

```
int32_t sl_iperf_udp_clnt_hdr_v1::amount
```

Amount.

Definition at line 350 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_ext

iPerf Client Header extended

Public Attributes

int32_t	type Type.
int32_t	length Length.
int16_t	u_flags Upper flags.
int16_t	l_flags Lower flags.
uint32_t	u_version Upper version.
uint32_t	l_version Lower version.
uint16_t	reserved Reserved.
uint16_t	tos Tos.
uint32_t	l_rate Lower rate.
uint32_t	u_rate Upper rate.
uint32_t	tcp_write_prefetch TCP write prefetch.

Public Attribute Documentation

type

```
int32_t sl_iperf_clnt_hdr_ext::type
```

Type.

Definition at line 356 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

length

```
int32_t sl_iperf_clnt_hdr_ext::length
```

Length.

Definition at line 358 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_flags

```
int16_t sl_iperf_clnt_hdr_ext::u_flags
```

Upper flags.

Definition at line 360 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

l_flags

```
int16_t sl_iperf_clnt_hdr_ext::l_flags
```

Lower flags.

Definition at line 362 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_version

```
uint32_t sl_iperf_clnt_hdr_ext::u_version
```

Upper version.

Definition at line 364 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

l_version

```
uint32_t sl_iperf_clnt_hdr_ext::l_version
```

Lower version.

Definition at line 366 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

reserved

```
uint16_t sl_iperf_clnt_hdr_ext::reserved
```

Reserved.

Definition at line 368 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

tos

```
uint16_t sl_iperf_clnt_hdr_ext::tos
```

Tos.

Definition at line 370 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

l_rate

```
uint32_t sl_iperf_clnt_hdr_ext::l_rate
```

Lower rate.

Definition at line 372 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_rate

```
uint32_t sl_iperf_clnt_hdr_ext::u_rate
```

Upper rate.

Definition at line 374 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

tcp_write_prefetch

```
uint32_t sl_iperf_clnt_hdr_ext::tcp_write_prefetch
```

TCP write prefetch.

Definition at line 376 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_isoch_payload

iPerf Client Isochronus payload

Public Attributes

uint32_t	burst_period period units microseconds
uint32_t	start_tv_sec Start sec.
uint32_t	start_tv_usec Start usec.
uint32_t	prev_frameid Previous frame ID.
uint32_t	frame_id Frame ID.
uint32_t	burst_size Burst size.
uint32_t	remaining Remaining.
uint32_t	reserved Reserved.

Public Attribute Documentation

burst_period

> uint32_t sl_iperf_clnt_hdr_isoch_payload::burst_period

period units microseconds

Definition at line `382` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h`

start_tv_sec

> uint32_t sl_iperf_clnt_hdr_isoch_payload::start_tv_sec

Start sec.

Definition at line `384` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h`

start_tv_usec

uint32_t sl_iperf_clnt_hdr_isoch_payload::start_tv_usec

Start usec.

Definition at line 386 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

prev_frameid

uint32_t sl_iperf_clnt_hdr_isoch_payload::prev_frameid

Previous frame ID.

Definition at line 388 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

frame_id

uint32_t sl_iperf_clnt_hdr_isoch_payload::frame_id

Frame ID.

Definition at line 390 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

burst_size

uint32_t sl_iperf_clnt_hdr_isoch_payload::burst_size

Burst size.

Definition at line 392 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

remaining

uint32_t sl_iperf_clnt_hdr_isoch_payload::remaining

Remaining.

Definition at line 394 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

reserved

uint32_t sl_iperf_clnt_hdr_isoch_payload::reserved

Reserved.

Definition at line 396 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_ext_starttime_fq

iPerf Client header extended FQ start time

Public Attributes

uint32_t	reserved	Reserved.
uint32_t	start_tv_sec	Start sec.
uint32_t	start_tv_usec	Start usec.
uint32_t	l_fq_rate	Lower FQ rate.
uint32_t	u_fq_rate	Upper FQ rate.

Public Attribute Documentation

reserved

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::reserved
```

Reserved.

Definition at line 402 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

start_tv_sec

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::start_tv_sec
```

Start sec.

Definition at line 404 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

start_tv_usec

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::start_tv_usec
```

Start usec.

Definition at line 406 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

l_fq_rate

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::l_fq_rate
```

Lower FQ rate.

Definition at line 408 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_fq_rate

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::u_fq_rate
```

Upper FQ rate.

Definition at line 410 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_clnt_hdr_ext_isoch_settings

iPerf Client Isochronus settings

Public Attributes

int32_t	l_fps Lower FPS.
int32_t	u_fps Upper FPS.
int32_t	l_mean Lower Mean.
int32_t	u_mean Upper Mean.
int32_t	l_variance Lower Variance.
int32_t	u_variance Upper Variance.
int32_t	l_burst_ipg Lower Burst IPG.
int32_t	u_burst_ipg Upper Burst IPG.

Public Attribute Documentation

l_fps

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_fps
```

Lower FPS.

Definition at line 416 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_fps

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_fps
```

Upper FPS.

Definition at line 418 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

l_mean

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_mean
```

Lower Mean.

Definition at line 420 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_mean

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_mean
```

Upper Mean.

Definition at line 422 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

l_variance

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_variance
```

Lower Variance.

Definition at line 424 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_variance

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_variance
```

Upper Variance.

Definition at line 426 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

l_burst_ipg

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_burst_ipg
```

Lower Burst IPG.

Definition at line 428 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

u_burst_ipg

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_burst_ipg
```

Upper Burst IPG.

Definition at line 430 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

sl_iperf_udp_clnt_hdr

iPerf Client UDP header

Public Attributes

sl_iperf_udp_datagram_t	dtg Datagram.
sl_iperf_clnt_hdr_v1_t	base Base v1 header.
sl_iperf_clnt_hdr_ext_t	extend Extended header.
sl_iperf_clnt_hdr_isoch_payload_t	isoch Isochronus payload.
sl_iperf_clnt_hdr_ext_starttime_fq_t	start_fq Extended start FQ.
sl_iperf_clnt_hdr_ext_isoch_settings_t	isoch_settings Isochronus settings.

Public Attribute Documentation

dtg

```
sl_iperf_udp_datagram_t sl_iperf_udp_clnt_hdr::dtg
```

Datagram.

Definition at line 436 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

base

```
sl_iperf_clnt_hdr_v1_t sl_iperf_udp_clnt_hdr::base
```

Base v1 header.

Definition at line 438 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

extend

```
sl_iperf_clnt_hdr_ext_t sl_iperf_udp_clnt_hdr::extend
```

Extended header.

Definition at line 440 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

isoch

> sl_iperf_clnt_hdr_isoch_payload_t sl_iperf_udp_clnt_hdr::isoch

Isochronus payload.

Definition at line 442 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

start_fq

> sl_iperf_clnt_hdr_ext_starttime_fq_t sl_iperf_udp_clnt_hdr::start_fq

Extended start FQ.

Definition at line 444 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

isoch_settings

> sl_iperf_clnt_hdr_ext_isoch_settings_t sl_iperf_udp_clnt_hdr::isoch_settings

Isochronus settings.

Definition at line 446 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/iperf/sl_iperf_types.h

Over-The-Air Device Firmware Upgrade (Alpha)

Over-The-Air Device Firmware Upgrade (Alpha)

Wi-SUN Over-The-Air Device Firmware Upgrade (OTA DFU) service is implemented for updating Wi-SUN device's firmware. The component uses the Gecko Bootloader API to perform firmware write, verify, and set to bootload operations.

A Trivial File Transfer Protocol (TFTP) client provides downloader solution to obtain a new Gecko Bootloader File (GBL) file from remote host Over-The-Air, using the Wi-SUN network. The connection to the TFTP remote host can be configured in component configuration file. The entire firmware upgrade session can be managed over CoAP. The service includes notification and status request capabilities.

Modules

Type definitions

Functions

void	sl_wisun_ota_dfu_init(void) Initialize the device firmware upgrade service.
sl_status_t	sl_wisun_ota_dfu_start_fw_update(void) Start firmware update.
sl_status_t	sl_wisun_ota_dfu_stop_fw_update(void) Stop firmware update.
sl_status_t	sl_wisun_ota_dfu_reboot_and_install(void) Reboot device.
uint32_t	sl_wisun_ota_dfu_get_fw_update_status(void) Get the status value.
const char *	sl_wisun_ota_dfu_get_fw_update_status_json_str(void) Get the status string in JSON format.
void	sl_wisun_ota_dfu_free_fw_update_status_json_str(const char *str) Free the status string buffer.
bool	sl_wisun_ota_dfu_get_fw_update_status_flag(const sl_wisun_ota_dfu_status_t status_flag) Get the status flag value.
void	sl_wisun_ota_dfu_error_hnd(const sl_wisun_ota_dfu_error_code_t error_code, sl_wisun_ota_dfu_error_ctx_t *const ctx) OTA DFU error handler (weak implementation)

Function Documentation

sl_wisun_ota_dfu_init

```
void sl_wisun_ota_dfu_init (void)
```

Initialize the device firmware upgrade service.

Parameters

N/A		

Initialize Wi-SUN OTA Device Firmware Upgrade service.

Definition at line 144 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_start_fw_update

```
sl_status_t sl_wisun_ota_dfu_start_fw_update (void)
```

Start firmware update.

Parameters

N/A		

Start firmware update by setting SL_WISUN_OTA_DFU_STATUS_FW_UPDATE_STARTED flag **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

Definition at line 152 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_stop_fw_update

```
sl_status_t sl_wisun_ota_dfu_stop_fw_update (void)
```

Stop firmware update.

Parameters

N/A		

Stop firmware update by setting SL_WISUN_OTA_DFU_STATUS_FW_UPDATE_STOPPED flag **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

Definition at line 160 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_reboot_and_install

```
sl_status_t sl_wisun_ota_dfu_reboot_and_install (void)
```

Reboot device.

Parameters

N/A		

Reboot device with calling corresponding gecko bootloader 'bootloader_rebootAndInstall' API This functions is available if auto-reboot mode is disabled. **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

Definition at line 170 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_get_fw_update_status

```
uint32_t sl_wisun_ota_dfu_get_fw_update_status (void)
```

Get the status value.

Parameters

N/A		

Returning the value of event flags **Returns**

- uint32_t Status

Definition at line 178 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_get_fw_update_status_json_str

```
const char * sl_wisun_ota_dfu_get_fw_update_status_json_str (void)
```

Get the status string in JSON format.

Parameters

N/A		

String buffer is allocated in heap by CoAP allocator **Returns**

- const char * Allocated string pointer on success, otherwise NULL

Definition at line 185 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_free_fw_update_status_json_str

```
void sl_wisun_ota_dfu_free_fw_update_status_json_str (const char *str)
```

Free the status string buffer.

Parameters

[in]	str	String ptr

Call CoAP free to release allocated memory

Definition at line 192 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_get_fw_update_status_flag

```
bool sl_wisun_ota_dfu_get_fw_update_status_flag (const sl_wisun_ota_dfu_status_t status_flag)
```

Get the status flag value.

Parameters

[in]	status_flag	Status flag enum

Bool representation of status variable bit value **Returns**

bool true if the flag is set, otherwise false

Definition at line 200 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_hnd

> void sl_wisun_ota_dfu_error_hnd (const sl_wisun_ota_dfu_error_code_t error_code, sl_wisun_ota_dfu_error_ctx_t *const ctx)

OTA DFU error handler (weak implementation)

Parameters

[in]	error_code	Error code
[in]	ctx	Error context with error details

Catch error in different stages of boot load.

Definition at line 208 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

Type definitions

Type definitions

Modules

sl_wisun_ota_dfu_error_ctx_fw_download

sl_wisun_ota_dfu_error_ctx_btl_fw_verify

sl_wisun_ota_dfu_error_ctx_btl_fw_set

sl_wisun_ota_dfu_error_ctx

Enumerations

enum sl_wisun_ota_dfu_status {

SL_WISUN_OTA_DFU_STATUS_FW_UPDATE_STARTED = 0
SL_WISUN_OTA_DFU_STATUS_FW_DOWNLOADED
SL_WISUN_OTA_DFU_STATUS_FW_VERIFIED
SL_WISUN_OTA_DFU_STATUS_FW_SET
SL_WISUN_OTA_DFU_STATUS_FW_UPDATE_STOPPED
SL_WISUN_OTA_DFU_STATUS_FW_DOWNLOAD_ERROR
SL_WISUN_OTA_DFU_STATUS_FW_VERIFY_ERROR
SL_WISUN_OTA_DFU_STATUS_FW_SET_ERROR

}

Wi-SUN OTA DFU Status enumeration.

enum sl_wisun_ota_dfu_error_code {

SL_WISUN_OTA_DFU_ERROR_FW_DOWNLOAD = 1001UL
SL_WISUN_OTA_DFU_ERROR_FW_VERIFY
SL_WISUN_OTA_DFU_ERROR_FW_SET

}

Wi-SUN OTA DFU error code enumeration.

Typedefs

typedef enum sl_wisun_ota_dfu_status — sl_wisun_ota_dfu_status_t
Wi-SUN OTA DFU Status enumeration.

typedef enum sl_wisun_ota_dfu_error_code — sl_wisun_ota_dfu_error_code_t
Wi-SUN OTA DFU error code enumeration.

typedef struct sl_wisun_ota_dfu_error_ctx_fw_download — sl_wisun_ota_dfu_error_ctx_fw_download_t
Wi-SUN OTA DFU download error context definition.

typedef struct sl_wisun_ota_dfu_error_ctx_btl_fw_verify	sl_wisun_ota_dfu_error_ctx_btl_fw_verify_t Wi-SUN OTA DFU verify error context definition.
typedef struct sl_wisun_ota_dfu_error_ctx_btl_fw_set	sl_wisun_ota_dfu_error_ctx_btl_fw_set_t Wi-SUN OTA DFU set error context definition.
typedef union sl_wisun_ota_dfu_error_ctx	sl_wisun_ota_dfu_error_ctx_t Wi-SUN OTA DFU error context definition.

Enumeration Documentation

sl_wisun_ota_dfu_status

> sl_wisun_ota_dfu_status

Wi-SUN OTA DFU Status enumeration.

Enumerator	
SL_WISUN_OTA_DFU_STATUS_FW_UPDATE_STARTED	Firmware update started.
SL_WISUN_OTA_DFU_STATUS_FW_DOWNLOADED	Firmware downloaded.
SL_WISUN_OTA_DFU_STATUS_FW_VERIFIED	Firmware verified.
SL_WISUN_OTA_DFU_STATUS_FW_SET	Firmware set.
SL_WISUN_OTA_DFU_STATUS_FW_UPDATE_STOPPED	
SL_WISUN_OTA_DFU_STATUS_FW_DOWNLOAD_ERROR	Firmware Download error.
SL_WISUN_OTA_DFU_STATUS_FW_VERIFY_ERROR	Firmware Verify error.
SL_WISUN_OTA_DFU_STATUS_FW_SET_ERROR	Firmware Set error.

Definition at line `73` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h`

sl_wisun_ota_dfu_error_code

> sl_wisun_ota_dfu_error_code

Wi-SUN OTA DFU error code enumeration.

Enumerator	
SL_WISUN_OTA_DFU_ERROR_FW_DOWNLOAD	Firmware downloaded error code.
SL_WISUN_OTA_DFU_ERROR_FW_VERIFY	Firmware verify error code.
SL_WISUN_OTA_DFU_ERROR_FW_SET	Firmware set error code.

Definition at line `93` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h`

Typedef Documentation

sl_wisun_ota_dfu_status_t

> typedef enum sl_wisun_ota_dfu_status sl_wisun_ota_dfu_status_t

Wi-SUN OTA DFU Status enumeration.

Definition at line 90 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_code_t

```
typedef enum sl_wisun_ota_dfu_error_code sl_wisun_ota_dfu_error_code_t
```

Wi-SUN OTA DFU error code enumeration.

Definition at line 100 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx_fw_download_t

```
typedef struct sl_wisun_ota_dfu_error_ctx_fw_download sl_wisun_ota_dfu_error_ctx_fw_download_t
```

Wi-SUN OTA DFU download error context definition.

Definition at line 110 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx_btl_fw_verify_t

```
typedef struct sl_wisun_ota_dfu_error_ctx_btl_fw_verify sl_wisun_ota_dfu_error_ctx_btl_fw_verify_t
```

Wi-SUN OTA DFU verify error context definition.

Definition at line 116 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx_btl_fw_set_t

```
typedef struct sl_wisun_ota_dfu_error_ctx_btl_fw_set sl_wisun_ota_dfu_error_ctx_btl_fw_set_t
```

Wi-SUN OTA DFU set error context definition.

Definition at line 122 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx_t

```
typedef union sl_wisun_ota_dfu_error_ctx sl_wisun_ota_dfu_error_ctx_t
```

Wi-SUN OTA DFU error context definition.

Definition at line 132 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx_fw_download

Wi-SUN OTA DFU download error context definition.

Public Attributes

int32_t	ret_val Return value of API call.
uint32_t	offset Offset of GBL file.
uint16_t	data_size Data size.

Public Attribute Documentation

ret_val

```
int32_t sl_wisun_ota_dfu_error_ctx_fw_download::ret_val
```

Return value of API call.

Definition at line 105 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

offset

```
uint32_t sl_wisun_ota_dfu_error_ctx_fw_download::offset
```

Offset of GBL file.

Definition at line 107 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

data_size

```
uint16_t sl_wisun_ota_dfu_error_ctx_fw_download::data_size
```

Data size.

Definition at line 109 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx_btl_fw_verify

Wi-SUN OTA DFU verify error context definition.

Public Attributes

int32_t	ret_val Return value of API call.

Public Attribute Documentation

ret_val

```
int32_t sl_wisun_ota_dfu_error_ctx_btl_fw_verify::ret_val
```

Return value of API call.

Definition at line 115 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx_btl_fw_set

Wi-SUN OTA DFU set error context definition.

Public Attributes

int32_t	ret_val Return value of API call.

Public Attribute Documentation

ret_val

```
int32_t sl_wisun_ota_dfu_error_ctx_btl_fw_set::ret_val
```

Return value of API call.

Definition at line 121 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

sl_wisun_ota_dfu_error_ctx

Wi-SUN OTA DFU error context definition.

Public Attributes

sl_wisun_ota_dfu_error_ctx_fw_download_t	download Download error context.
sl_wisun_ota_dfu_error_ctx_btl_fw_verify_t	verify Verify error context.
sl_wisun_ota_dfu_error_ctx_btl_fw_set_t	set Set error context.

Public Attribute Documentation

download

```
sl_wisun_ota_dfu_error_ctx_fw_download_t sl_wisun_ota_dfu_error_ctx::download
```

Download error context.

Definition at line 127 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

verify

```
sl_wisun_ota_dfu_error_ctx_btl_fw_verify_t sl_wisun_ota_dfu_error_ctx::verify
```

Verify error context.

Definition at line 129 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

set

```
sl_wisun_ota_dfu_error_ctx_btl_fw_set_t sl_wisun_ota_dfu_error_ctx::set
```

Set error context.

Definition at line 131 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/app/wisun/component/ota_dfu/sl_wisun_ota_dfu.h

Silicon Labs socket API (deprecated)

Silicon Labs socket API (deprecated)

Silicon Labs socket API is deprecated. This component is provided as a temporary solution to maintain a compatility with v1.7 and older.

Functions

sl_status_t sl_wisun_open_socket(sl_wisun_socket_protocol_t protocol, sl_wisun_socket_id_t *socket_id) SL_DEPRECATED_API_SDK_4_4
Open a socket.

sl_status_t sl_wisun_close_socket(sl_wisun_socket_id_t socket_id) SL_DEPRECATED_API_SDK_4_4
Close a socket.

sl_status_t sl_wisun_sendto_on_socket(sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *remote_address, uint16_t remote_port, uint16_t data_length, const uint8_t *data) SL_DEPRECATED_API_SDK_4_4
Write data to an unconnected socket.

sl_status_t sl_wisun_listen_on_socket(sl_wisun_socket_id_t socket_id) SL_DEPRECATED_API_SDK_4_4
Set a TCP socket to listening state.

sl_status_t sl_wisun_accept_on_socket(sl_wisun_socket_id_t socket_id, sl_wisun_socket_id_t *remote_socket_id, sl_wisun_ip_address_t *remote_address, uint16_t *remote_port) SL_DEPRECATED_API_SDK_4_4
Accept a pending connection request on a TCP socket.

sl_status_t sl_wisun_connect_socket(sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *remote_address, uint16_t remote_port) SL_DEPRECATED_API_SDK_4_4
Initiate a connection from a socket to a remote peer socket.

sl_status_t sl_wisun_bind_socket(sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *local_address, uint16_t local_port) SL_DEPRECATED_API_SDK_4_4
Bind a socket to a specific local address and/or a port number.

sl_status_t sl_wisun_send_on_socket(sl_wisun_socket_id_t socket_id, uint16_t data_length, const uint8_t *data) SL_DEPRECATED_API_SDK_4_4
Write data to a connected socket.

sl_status_t sl_wisun_receive_on_socket(sl_wisun_socket_id_t socket_id, sl_wisun_ip_address_t *remote_address, uint16_t *remote_port, uint16_t *data_length, uint8_t *data) SL_DEPRECATED_API_SDK_4_4
Read data from a socket.

sl_status_t sl_wisun_set_socket_option(sl_wisun_socket_id_t socket_id, sl_wisun_socket_option_t option, const sl_wisun_socket_option_data_t *option_data) SL_DEPRECATED_API_SDK_4_4
Set a socket option.

sl_status_t sl_wisun_get_socket_option(sl_wisun_socket_id_t socket_id, sl_wisun_socket_option_t option, sl_wisun_socket_option_data_t *option_data) SL_DEPRECATED_API_SDK_4_4
Get a socket option.

Function Documentation

sl_wisun_open_socket

```
sl_status_t sl_wisun_open_socket (sl_wisun_socket_protocol_t protocol, sl_wisun_socket_id_t *socket_id)
```

SL_DEPRECATED_API_SDK_4_4

Open a socket.

Parameters

[in]	protocol	Protocol type of the socket
[out]	socket_id	ID of the opened socket

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function opens a socket. Up to 10 sockets may be open at any given time, including those opened implicitly via sl_wisun_accept_on_socket().

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See socket() for a replacement.

Definition at line 55 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_close_socket

sl_status_t sl_wisun_close_socket (sl_wisun_socket_id_t socket_id) SL_DEPRECATED_API_SDK_4_4

Close a socket.

Parameters

[in]	socket_id	ID of the socket

Returns

- SL_STATUS_OK if successful, an error code otherwise

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See close() for a replacement.

Definition at line 67 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_sendto_on_socket

sl_status_t sl_wisun_sendto_on_socket (sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *remote_address, uint16_t remote_port, uint16_t data_length, const uint8_t *data) SL_DEPRECATED_API_SDK_4_4

Write data to an unconnected socket.

Parameters

[in]	socket_id	ID of the socket
[in]	remote_address	IP address of the remote peer
[in]	remote_port	Port number of the remote peer
[in]	data_length	Amount of data to write
[in]	data	Pointer to the data

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function initiates a data transmission to a remote peer and can only be used on an unconnected UDP or ICMP socket. Completion of the transmission is indicated with a SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID event. The function takes

a copy of the data, so the caller may free the resource when the function returns.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See sendto() for a replacement.

Definition at line 88 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_listen_on_socket

```
sl_status_t sl_wisun_listen_on_socket (sl_wisun_socket_id_t socket_id) SL_DEPRECATED_API_SDK_4_4
```

Set a TCP socket to listening state.

Parameters

[in]	socket_id	ID of the socket

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets a TCP socket to listening state, allowing it to act as a server socket, i.e., to receive connection requests from clients. Connection requests are indicated with SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID events and accepted using sl_wisun_accept_on_socket(). This function can only be used on an unconnected TCP socket.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See listen() for a replacement.

Definition at line 110 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_accept_on_socket

```
sl_status_t sl_wisun_accept_on_socket (sl_wisun_socket_id_t socket_id, sl_wisun_socket_id_t *remote_socket_id,
sl_wisun_ip_address_t *remote_address, uint16_t *remote_port) SL_DEPRECATED_API_SDK_4_4
```

Accept a pending connection request on a TCP socket.

Parameters

[in]	socket_id	ID of the socket on listening state
[out]	remote_socket_id	ID of the new connected socket
[out]	remote_address	IP address of the remote peer
[out]	remote_port	Port number of the remote peer

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function accepts a pending connection request from a remote peer and creates a new connected TCP socket for the connection. To decline a connection request, the request must be accepted and then closed using sl_wisun_close_socket(). The function can only be used on a TCP socket in listening state.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See accept() for a replacement.

Definition at line 130 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_connect_socket

```
sl_status_t sl_wisun_connect_socket (sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *remote_address,
uint16_t remote_port) SL_DEPRECATED_API_SDK_4_4
```

Initiate a connection from a socket to a remote peer socket.

Parameters

[in]	socket_id	ID of the socket
[in]	remote_address	IP address of the remote peer
[in]	remote_port	Port number of the remote peer

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function initiates a connection from a local socket to to a remote peer socket. The result of the connection is indicated with a SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID event. Connecting a socket is mandatory for TCP client sockets but may be also used on other types of sockets. A connected socket can only receive and transmit data with the designated peer. This function can only be used on an unconnected TCP or UDP socket.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See connect() for a replacement.

Definition at line 154 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_bind_socket

> sl_status_t sl_wisun_bind_socket (sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *local_address, uint16_t local_port) SL_DEPRECATED_API_SDK_4_4

Bind a socket to a specific local address and/or a port number.

Parameters

[in]	socket_id	ID of the socket
[in]	local_address	Local IP address to use on the socket. NULL if not bound.
[in]	local_port	Local port number to use on the socket. Zero if not bound.

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function binds the local address and/or the port of a socket. When the local address is bound, the socket will only accept traffic sent to the specified address and the transmitted packets will use the address as the source address. If not bound, the socket will accept data sent to any valid address of the device. The source address is selected by the stack. Binding the local port number sets the port number for received and transmitted packets. If not bound, the stack will select a port number automatically. This function can only be used on an unconnected TCP or UDP socket. Once bound to a specific address and/or port, the value cannot be changed.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See bind() for a replacement.

Definition at line 180 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_send_on_socket

> sl_status_t sl_wisun_send_on_socket (sl_wisun_socket_id_t socket_id, uint16_t data_length, const uint8_t *data) SL_DEPRECATED_API_SDK_4_4

Write data to a connected socket.

Parameters

[in]	socket_id	ID of the socket
[in]	data_length	Amount of data to write
[in]	data	Pointer to the data

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function initiates transmission of data to a connected remote peer and can only be used on a connected socket. Completion of the transmission is indicated with a SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID event. The function takes a copy of the data, so the caller may free the resource when the function returns.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See send() for a replacement.

Definition at line 201 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_receive_on_socket

> sl_status_t sl_wisun_receive_on_socket (sl_wisun_socket_id_t socket_id, sl_wisun_ip_address_t *remote_address, uint16_t *remote_port, uint16_t *data_length, uint8_t *data) SL_DEPRECATED_API_SDK_4_4

Read data from a socket.

Parameters

[in]	socket_id	ID of the socket
[out]	remote_address	IP address of the remote peer
[out]	remote_port	Port number of the remote peer
[inout]	data_length	Amount of data to read on input, amount of data read on output
[in]	data	Pointer to where the read data is stored

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function reads buffered data from a socket. When reading data from a UDP or ICMP socket, the entire packet must be read. Any data left unread is discarded after this call. TCP sockets allow reading only a part of the buffered data.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See recvfrom() for a replacement.

Definition at line 223 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_set_socket_option

> sl_status_t sl_wisun_set_socket_option (sl_wisun_socket_id_t socket_id, sl_wisun_socket_option_t option, const sl_wisun_socket_option_data_t *option_data) SL_DEPRECATED_API_SDK_4_4

Set a socket option.

Parameters

[in]	socket_id	ID of the socket

[in]	option	Socket option to set • SL_WISUN_SOCKET_OPTION_EVENT_MODE: Event mode • SL_WISUN_SOCKET_OPTION_MULTICAST_GROUP: Multicast group • SL_WISUN_SOCKET_OPTION_SEND_BUFFER_LIMIT: Tx buffer limit • SL_WISUN_SOCKET_OPTION_EDFE_MODE: Enable/disable EDFE mode • SL_WISUN_SOCKET_OPTION_UNICAST_HOP_LIMIT: Socket unicast hop limit • SL_WISUN_SOCKET_OPTION_MULTICAST_HOP_LIMIT: Socket multicast hop limit
[in]	option_data	Socket option specific data

Returns

- SL_STATUS_OK if successful, an error code otherwise

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See setsockopt() for a replacement.

Definition at line 246 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

sl_wisun_get_socket_option

> sl_status_t sl_wisun_get_socket_option (sl_wisun_socket_id_t socket_id, sl_wisun_socket_option_t option, sl_wisun_socket_option_data_t *option_data) SL_DEPRECATED_API_SDK_4_4

Get a socket option.

Parameters

[in]	socket_id	ID of the socket
[in]	option	Socket option to get • SL_WISUN_SOCKET_OPTION_SEND_BUFFER_LIMIT: Send buffer limit
[out]	option_data	Socket option specific data

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function retrieves the value of a socket option.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack. See getsockopt() for a replacement.

Definition at line 264 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/plugin/sl_wisun_legacy_socket_wrapper.h

Wi-SUN Stack Plugin

Wi-SUN Stack Plugin

The Wi-SUN stack plugin components are software modules tightly linked to the stack that provide means to customize it: debug, manufacturing, or Wi-SUN-specific optional features. They can have significant impact on key capabilities and memory footprint.

Modules

Stack Trace and Debug

Stack Trace and Debug

The Trace and Debug component adds tracing capabilities to the stack and stack plugin components. The component provides APIs to configure or filter out the debug traces output.

Functions

sl_status_t sl_wisun_set_trace_level(uint8_t group_count, sl_wisun_trace_group_config_t *trace_config)
Set the trace level.

sl_status_t sl_wisun_set_trace_filter(uint8_t filter[SL_WISUN_FILTER_BITFIELD_SIZE])
Set the trace filter.

Function Documentation

sl_wisun_set_trace_level

```
sl_status_t sl_wisun_set_trace_level (uint8_t group_count, sl_wisun_trace_group_config_t *trace_config)
```

Set the trace level.

Parameters

[in]	group_count	Number of groups to configure. If 0, enable all levels for all groups. Maximum SL_WISUN_TRACE_GROUP_COUNT.
[in]	trace_config	Table with group_count element filled. It indicates the trace level to be displayed for each group.

Returns

- One of the following:
  - SL_STATUS_NOT_AVAILABLE if the Stack Trace and Debug component is not installed
  - SL_STATUS_OK if successful, an error code otherwise

Definition at line `59` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_trace_api.h`

sl_wisun_set_trace_filter

```
sl_status_t sl_wisun_set_trace_filter (uint8_t filter[SL_WISUN_FILTER_BITFIELD_SIZE])
```

Set the trace filter.

Parameters

[in]	filter	Bit mask of trace group IDs. First byte of the array represents IDs 0 - 7, with bit 0 being ID 0. Second byte represents IDs 8 - 15 and so forth. If a bit is set, the corresponding trace group ID is selected for tracing. 0 means the particular trace group ID is disabled. Bit enumeration is defined in sl_wisun_trace_group_t.

Indicate which trace group will be displayed.

Returns

One of the following:

- SL_STATUS_NOT_AVAILABLE if the Stack Trace and Debug component is not installed
- SL_STATUS_OK if successful, an error code otherwise

Definition at line `74` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_trace_api.h`

RF Test

RF Test

The RF Test component provides low-level APIs to produce an RF tone or a modulated packet. The RF Test component cannot run simultaneously with the Wi-SUN stack and is only meant for production calibration.

Functions

sl_status_t	sl_wisun_start_stream(uint16_t channel) Start transmitting a random stream of characters to enable the measurement of radio modulation.
sl_status_t	sl_wisun_stop_stream() Stop a previously started stream of characters.
sl_status_t	sl_wisun_start_tone(uint16_t channel) Start transmitting an unmodulated tone.
sl_status_t	sl_wisun_stop_tone() Stop a previously started tone.
sl_status_t	sl_wisun_set_test_tx_power(int8_t tx_power) Set transmit power.
bool	sl_wisun_is_running_rf_test() Return the current status of the RF test plugin.

Function Documentation

sl_wisun_start_stream

```
sl_status_t sl_wisun_start_stream (uint16_t channel)
```

Start transmitting a random stream of characters to enable the measurement of radio modulation.

Parameters

[in]	channel	Name of the Wi-SUN network as a zero-terminated string

Returns

- One of the following:
  - SL_STATUS_OK if the stream transmission started successfully.
  - SL_STATUS_NOT_READY if called before the stack initialization.
  - SL_STATUS_BUSY if a test is already running.
  - SL_STATUS_NETWORK_UP if a connection is already established or in progress.
  - SL_STATUS_INVALID_PARAMETER if an invalid channel is configured.

Transmit a PN9 bytes sequence. See RAIL_StartTxStream() for more information.

Definition at line `55` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_rf_test.h`

sl_wisun_stop_stream

sl_status_t sl_wisun_stop_stream ()

Stop a previously started stream of characters.

Parameters

[in]	channel	Name of the Wi-SUN network as a zero-terminated string

Returns

- One of the following:
  - SL_STATUS_OK if the stream transmission stopped successfully.
  - SL_STATUS_INVALID_STATE if while not transmitting a stream.

See RAIL_StopTxStream() for more information.

Definition at line 68 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_rf_test.h

sl_wisun_start_tone

sl_status_t sl_wisun_start_tone (uint16_t channel)

Start transmitting an unmodulated tone.

Parameters

[in]	channel	Name of the Wi-SUN network as a zero-terminated string

Returns

- One of the following:
  - SL_STATUS_OK if the stream transmission started successfully.
  - SL_STATUS_NOT_READY if called before the stack initialization.
  - SL_STATUS_BUSY if a test is already running.
  - SL_STATUS_NETWORK_UP if a connection is already established or in progress.
  - SL_STATUS_INVALID_PARAMETER if an invalid channel is configured.

Transmit a PN9 bytes sequence. See RAIL_StartTxStream() for more information.

Definition at line 84 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_rf_test.h

sl_wisun_stop_tone

sl_status_t sl_wisun_stop_tone ()

Stop a previously started tone.

Parameters

[in]	channel	Name of the Wi-SUN network as a zero-terminated string

Returns

- One of the following:
  - SL_STATUS_OK if the tone stopped successfully.
  - SL_STATUS_INVALID_STATE if while not transmitting a tone.

See RAIL_StopTxStream() for more information.

Definition at line 96 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_rf_test.h

sl_wisun_set_test_tx_power

```
sl_status_t sl_wisun_set_test_tx_power (int8_t tx_power)
```

Set transmit power.

Parameters

[in]	tx_power	Transmit power in units of dBm, can be negative.

Returns

- always SL_STATUS_OK

Definition at line 104 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_rf_test.h

sl_wisun_is_running_rf_test

```
bool sl_wisun_is_running_rf_test ()
```

Return the current status of the RF test plugin.

Returns

- One of the following:
  - True if a test is running.
  - False otherwise.

Definition at line 113 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_rf_test.h

Wi-SUN Stack API

Wi-SUN Stack API

Wi-SUN Stack API is based on requests from the application to the stack and events from the stack to the application.

Requests are made using function calls, where a function call either performs the required action immediately or initiates an internal operation within the stack, which terminates with an event. All events contain a status code, indicating the result of the requested operation. Events are also used by the stack to notify the application of any important information, such as the state of the connection.

The application is expected to override sl_wisun_on_event() to handle events from the stack. Because all events share a common header, the function may be implemented as a switch statement. The event-specific data can be accessed through the sl_wisun_evt_t::evt union.

```
void sl_wisun_on_event(sl_wisun_evt_t *evt)
{
  switch (evt->header.id) {
    case SL_WISUN_MSG_CONNECTED_IND_ID:
      handle_connected_event(evt->evt.connected);
      break;
    default:
      break;
  }
}
```

The API is thread-safe, which means can be called from multiple RTOS tasks. The stack guarantees that only a single request is executed at a time and that requests are handled in the order they were made. Event callback is executed in a different context than the request, so the API functions may be called from the event callback.

Modules

Wi-SUN API events

Wi-SUN API type definitions

Socket API

Callbacks

void sl_wisun_on_event(sl_wisun_evt_t *evt)
Callback handler for a single event.

Functions

sl_status_t sl_wisun_join(const uint8_t *name, sl_wisun_phy_config_t *phy_config)
Initiate a connection to a Wi-SUN network.

sl_status_t sl_wisun_get_ip_address(sl_wisun_ip_address_type_t address_type, in6_addr_t *address)
Read an IP address.

sl_status_t sl_wisun_disconnect()
Disconnect from the Wi-SUN network.

sl_status_t sl_wisun_set_trusted_certificate(uint16_t certificate_options, uint16_t certificate_length, const uint8_t *certificate)
Set a trusted certificate used to verify the authentication server certificate.

sl_status_t sl_wisun_set_device_certificate(uint16_t certificate_options, uint16_t certificate_length, const uint8_t *certificate)
Set the device certificate used to authenticate to the authentication server.

sl_status_t sl_wisun_set_device_private_key(uint16_t key_options, uint16_t key_length, const uint8_t *key)
Set the private key of the device certificate.

sl_status_t sl_wisun_get_statistics(sl_wisun_statistics_type_t statistics_type, sl_wisun_statistics_t *statistics)
Read a set of statistics.

sl_status_t sl_wisun_set_tx_power(int8_t tx_power)
Set the maximum TX power.

sl_status_t sl_wisun_set_allowed_channel_mask(const sl_wisun_channel_mask_t *channel_mask)
Set a mask of operating channels.

sl_status_t sl_wisun_set_channel_mask(const sl_wisun_channel_mask_t *channel_mask)
Set a mask of operating channels.

sl_status_t sl_wisun_allow_mac_address(const sl_wisun_mac_address_t *address)
Add a MAC address to the list of allowed addresses.

sl_status_t sl_wisun_deny_mac_address(const sl_wisun_mac_address_t *address)
Add a MAC address to the list of denied addresses.

sl_status_t sl_wisun_get_join_state(sl_wisun_join_state_t *join_state)
Get the current join state.

sl_status_t sl_wisun_clear_credential_cache()
Clear the credential cache.

sl_status_t sl_wisun_get_mac_address(sl_wisun_mac_address_t *address)
Get the current device MAC address in use.

sl_status_t sl_wisun_set_mac_address(const sl_wisun_mac_address_t *address)
Set the device MAC address to be used.

sl_status_t sl_wisun_reset_statistics(sl_wisun_statistics_type_t statistics_type)
Reset a set of statistics in the stack.

sl_status_t sl_wisun_get_neighbor_count(uint8_t *neighbor_count)
Get the number of RPL neighbors (parents and children).

sl_status_t sl_wisun_get_neighbors(uint8_t *neighbor_count, sl_wisun_mac_address_t *neighbor_mac_addresses)
Get a list of RPL neighbor (parents and children) MAC addresses.

sl_status_t sl_wisun_get_neighbor_info(const sl_wisun_mac_address_t *neighbor_mac_address, sl_wisun_neighbor_info_t *neighbor_info)
Get information about a RPL neighbor (parent or child).

sl_status_t sl_wisun_set_unicast_settings(uint8_t dwell_interval_ms)
Set unicast settings.

sl_status_t sl_wisun_set_device_private_key_id(uint32_t key_id)
Set the private key of the device certificate.

sl_status_t sl_wisun_set_regulation(sl_wisun_regulation_t regulation)
Set the regional regulation.

sl_status_t sl_wisun_set_regulation_tx_thresholds(int8_t warning_threshold, int8_t alert_threshold)
Set the thresholds for transmission duration level event.

sl_status_t sl_wisun_set_advert_fragment_duration(uint32_t fragment_duration_ms)
Set the async transmission fragmentation parameters.

sl_status_t sl_wisun_set_unicast_tx_mode(uint8_t mode)
Enable an algorithm that trades off unicast communication reliability for latency.

sl_status_t sl_wisun_set_device_type(sl_wisun_device_type_t device_type)
Set the device type.

sl_status_t sl_wisun_config_mode_switch(uint8_t mode, uint8_t phy_mode_id, const sl_wisun_mac_address_t *neighbor_address, bool reserved)
Set the mode switch configuration.

sl_status_t sl_wisun_set_mode_switch(uint8_t mode, uint8_t phy_mode_id, const sl_wisun_mac_address_t *neighbor_address)
Set the PHY mode switch configuration.

sl_status_t sl_wisun_set_connection_parameters(const sl_wisun_connection_params_t *params)
Configure the FFN parameter set.

sl_status_t sl_wisun_set_pom_ie(uint8_t phy_mode_id_count, uint8_t phy_mode_ids[SL_WISUN_MAX_PHY_MODE_ID_COUNT], uint8_t is_mdr_command_capable)
Set the POM-IE configuration.

sl_status_t sl_wisun_get_pom_ie(uint8_t *phy_mode_id_count, uint8_t *phy_mode_ids, uint8_t *is_mdr_command_capable)
Get the POM-IE configuration.

sl_status_t sl_wisun_get_stack_version(uint8_t *major, uint8_t *minor, uint8_t *patch, uint16_t *build)
Get the Wi-SUN stack version.

sl_status_t sl_wisun_set_lfn_parameters(const sl_wisun_lfn_params_t *params)
Configure the LFN parameter set.

sl_status_t sl_wisun_set_lfn_support(uint8_t lfn_limit)
Set the maximum number of LFN children.

sl_status_t sl_wisun_set_pti_state(bool pti_state)
Set the PTI state.

sl_status_t sl_wisun_trigger_frame(sl_wisun_frame_type_t frame_type)
Trigger the transmission of a frame (FAN Discovery, RPL).

sl_status_t sl_wisun_set_security_state(uint32_t security_state)
Set the security state.

sl_status_t sl_wisun_get_network_info(sl_wisun_network_info_t *network_info)
Get the Wi-SUN network information.

sl_status_t sl_wisun_get_rpl_info(sl_wisun_rpl_info_t *rpl_info)
Get RPL information.

sl_status_t sl_wisun_get_excluded_channel_mask(sl_wisun_channel_mask_type_t type, sl_wisun_channel_mask_t *channel_mask, uint8_t *channel_count)
Get the mask of channels excluded from channel plan.

Callbacks Documentation

sl_wisun_on_event

```
void sl_wisun_on_event (sl_wisun_evt_t *evt)
```

Callback handler for a single event.

Parameters

N/A	evt	The event to be handled

This function is called when the stack sends an event to the application. The application can declare its own version this function to customize event handling. The default implementation discards all events.

See Also

- Wi-SUN API events

Definition at line `98` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h`

Function Documentation

sl_wisun_join

```
sl_status_t sl_wisun_join (const uint8_t *name, sl_wisun_phy_config_t *phy_config)
```

Initiate a connection to a Wi-SUN network.

Parameters

[in]	name	Name of the Wi-SUN network as a zero-terminated string
[in]	phy_config	Pointer to PHY configuration

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function initiates connection to a Wi-SUN network. Completion of the request is indicated with a SL_WISUN_MSG_CONNECTED_IND_ID event.

Definition at line `114` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h`

sl_wisun_get_ip_address

```
sl_status_t sl_wisun_get_ip_address (sl_wisun_ip_address_type_t address_type, in6_addr_t *address)
```

Read an IP address.

Parameters

[in]	address_type	Type of the IP address to read • SL_WISUN_IP_ADDRESS_TYPE_LINK_LOCAL: Link-local IPv6 address of the device • SL_WISUN_IP_ADDRESS_TYPE_GLOBAL: Global unicast IPv6 address of the device • SL_WISUN_IP_ADDRESS_TYPE_BORDER_ROUTER: Global unicast IPv6 address of the border router • SL_WISUN_IP_ADDRESS_TYPE_PRIMARY_PARENT: Link-local IPv6 address of the primary parent • SL_WISUN_IP_ADDRESS_TYPE_SECONDARY_PARENT: Link-local IPv6 address of the secondary parent
[out]	address	IP address to read

Returns

- SL_STATUS_OK if successful, an error code otherwise.

Definition at line 128 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_disconnect

```
sl_status_t sl_wisun_disconnect ()
```

Disconnect from the Wi-SUN network.

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function disconnects an active connection or cancels an ongoing connection attempt.

Definition at line 139 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_trusted_certificate

```
sl_status_t sl_wisun_set_trusted_certificate (uint16_t certificate_options, uint16_t certificate_length, const uint8_t *certificate)
```

Set a trusted certificate used to verify the authentication server certificate.

Parameters

[in]	certificate_options	Options for the certificate • SL_WISUN_CERTIFICATE_OPTION_APPEND: Append the certificate to the list of trusted certificates instead of replacing the previous entries • SL_WISUN_CERTIFICATE_OPTION_IS_REF: The application guarantees the certificate data will remain in scope and can therefore be referenced instead of copied
[in]	certificate_length	Size of the certificate data
[in]	certificate	Pointer to the certificate data

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 153 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_device_certificate

```
sl_status_t sl_wisun_set_device_certificate (uint16_t certificate_options, uint16_t certificate_length, const uint8_t *certificate)
```

Set the device certificate used to authenticate to the authentication server.

Parameters

[in]	certificate_options	Options for the certificate. • SL_WISUN_CERTIFICATE_OPTION_APPEND: Append the certificate to the list of device certificates instead of replacing the previous entries • SL_WISUN_CERTIFICATE_OPTION_IS_REF: The application guarantees the certificate data will remain in scope and can therefore be referenced instead of copied • SL_WISUN_CERTIFICATE_OPTION_HAS_KEY: The certificate has a private key
[in]	certificate_length	Size of the certificate data
[in]	certificate	Pointer to the certificate data

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 170 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_device_private_key

```
sl_status_t sl_wisun_set_device_private_key (uint16_t key_options, uint16_t key_length, const uint8_t *key)
```

Set the private key of the device certificate.

Parameters

[in]	key_options	Options for the private key • SL_WISUN_PRIVATE_KEY_OPTION_IS_REF: The application guarantees the private key data will remain in scope and can therefore be referenced instead of copied
[in]	key_length	Size of the private key data
[in]	key	Pointer to the private key data

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 184 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_statistics

```
sl_status_t sl_wisun_get_statistics (sl_wisun_statistics_type_t statistics_type, sl_wisun_statistics_t *statistics)
```

Read a set of statistics.

Parameters

[in]	statistics_type	Type of statistics to read • SL_WISUN_STATISTICS_TYPE_PHY: PHY/RF statistics • SL_WISUN_STATISTICS_TYPE_MAC: MAC statistics • SL_WISUN_STATISTICS_TYPE_FHSS: Frequency hopping statistics • SL_WISUN_STATISTICS_TYPE_WISUN: Wi-SUN statistics • SL_WISUN_STATISTICS_TYPE_NETWORK: 6LoWPAN/IP stack statistics • SL_WISUN_STATISTICS_TYPE_REGULATION: Regional regulation statistics • SL_WISUN_STATISTICS_TYPE_HEAP: Heap usage statistics
[out]	statistics	Set of statistics read

Returns

- SL_STATUS_OK if successful, an error code otherwise.

This function reads a set of statistics from the stack. Statistics are cumulative and reset when a connection is initiated or by calling sl_wisun_reset_statistics().

Definition at line 206 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_tx_power

```
sl_status_t sl_wisun_set_tx_power (int8_t tx_power)
```

Set the maximum TX power.

Parameters

[in]	tx_power	TX power in dBm

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the maximum TX power. The device may use a lower value based on internal decision making or hardware limitations but will never exceed the given value.

Definition at line 219 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_allowed_channel_mask

```
sl_status_t sl_wisun_set_allowed_channel_mask (const sl_wisun_channel_mask_t *channel_mask)
```

Set a mask of operating channels.

Parameters

[in]	channel_mask	Mask of operating channels

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets a mask of channels the device is allowed to operate in for unicast frequency hopping. By default, all channels in the channel plan are allowed. The mask can only be used to further restrict the channels. Channels outside the channel plan or channels internally excluded are ignored. This mask will be used in the following connections.

Warnings

-

By comparison to the Wi-SUN FAN specification, the channel mask logic is inverted. The specification references a mask of excluded channels.

Definition at line 237 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_channel_mask

```
sl_status_t sl_wisun_set_channel_mask (const sl_wisun_channel_mask_t *channel_mask)
```

Set a mask of operating channels.

Parameters

[in]	channel_mask	Mask of operating channels

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets a mask of channels the device is allowed to operate in for unicast frequency hopping. By default, all channels in the channel plan are allowed. The mask can only be used to further restrict the channels. Channels outside the channel plan or channels internally excluded are ignored. This mask will be used in the following connections.

Definition at line 251 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_allow_mac_address

```
sl_status_t sl_wisun_allow_mac_address (const sl_wisun_mac_address_t *address)
```

Add a MAC address to the list of allowed addresses.

Parameters

[in]	address	MAC address • **sl_wisun_broadcast_mac**: allow all MAC addresses • **unicast address**: allow the given MAC address

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function adds a MAC address to the list of allowed addresses. When the first address is added to the list, the list of denied addresses is cleared and the device will start preventing communication with any device whose MAC address does not match any of addresses on the list. By default, all MAC addresses are allowed. Up to 10 MAC addresses may be added to the list. The access list affects only directly connected nodes such as parents, children, and neighbors.

Definition at line 269 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_deny_mac_address

```
sl_status_t sl_wisun_deny_mac_address (const sl_wisun_mac_address_t *address)
```

Add a MAC address to the list of denied addresses.

Parameters

[in]	address	MAC address • **sl_wisun_broadcast_mac**: deny all MAC addresses • **unicast address**: deny the given MAC address

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function adds a MAC address to the list of denied addresses. When the first address is added to the list, the list of allowed addresses is cleared and the device will start preventing communication with any device whose MAC address matches any of the addresses on the list. By default, all MAC addresses are allowed. Up to 10 MAC addresses may be added to the list. The access list affects only directly connected nodes such as parents, children, and neighbors.

Definition at line `287` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h`

sl_wisun_get_join_state

```
sl_status_t sl_wisun_get_join_state (sl_wisun_join_state_t *join_state)
```

Get the current join state.

Parameters

[out]	join_state	Join state

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function retrieves the current state of the connection process. The function can only be used once a connection has been initiated.

Definition at line `298` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h`

sl_wisun_clear_credential_cache

```
sl_status_t sl_wisun_clear_credential_cache ()
```

Clear the credential cache.

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function clears the cached authentication credentials stored in non-volatile storage. The function is intended for test purposes. Note that clearing the credential cache may prevent the node from reconnecting to the same parent until the corresponding cache entry has expired on the parent.

Definition at line `310` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h`

sl_wisun_get_mac_address

```
sl_status_t sl_wisun_get_mac_address (sl_wisun_mac_address_t *address)
```

Get the current device MAC address in use.

Parameters

[out]	address	MAC address

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 318 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_mac_address

```
sl_status_t sl_wisun_set_mac_address (const sl_wisun_mac_address_t *address)
```

Set the device MAC address to be used.

Parameters

[in]	address	MAC address

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the MAC address for use in the following connections. By default, the device will use the built-in unique device MAC address. The address is reset to the built-in value upon power up.

Definition at line 330 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_reset_statistics

```
sl_status_t sl_wisun_reset_statistics (sl_wisun_statistics_type_t statistics_type)
```

Reset a set of statistics in the stack.

Parameters

[in]	statistics_type	Type of statistics to reset • SL_WISUN_STATISTICS_TYPE_PHY: PHY/RF statistics • SL_WISUN_STATISTICS_TYPE_MAC: MAC statistics • SL_WISUN_STATISTICS_TYPE_FHSS: Frequency hopping statistics • SL_WISUN_STATISTICS_TYPE_WISUN: Wi-SUN statistics • SL_WISUN_STATISTICS_TYPE_NETWORK: 6LoWPAN/IP stack statistics

Returns

- SL_STATUS_OK if successful, an error code otherwise.

Definition at line 343 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_neighbor_count

```
sl_status_t sl_wisun_get_neighbor_count (uint8_t *neighbor_count)
```

Get the number of RPL neighbors (parents and children).

Parameters

[out]	neighbor_count	Number of neighbors

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 351 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_neighbors

```
sl_status_t sl_wisun_get_neighbors (uint8_t *neighbor_count, sl_wisun_mac_address_t *neighbor_mac_addresses)
```

Get a list of RPL neighbor (parents and children) MAC addresses.

Parameters

[inout]	neighbor_count	Maximum number of neighbors to read on input, number of neighbors read on output
[out]	neighbor_mac_addresses	Pointer to memory where to store neighbor MAC addresses

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 362 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_neighbor_info

```
sl_status_t sl_wisun_get_neighbor_info (const sl_wisun_mac_address_t *neighbor_mac_address, sl_wisun_neighbor_info_t *neighbor_info)
```

Get information about a RPL neighbor (parent or child).

Parameters

[in]	neighbor_mac_address	Pointer to neighbor MAC address
[out]	neighbor_info	Pointer to where the read information is stored

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 372 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_unicast_settings

```
sl_status_t sl_wisun_set_unicast_settings (uint8_t dwell_interval_ms)
```

Set unicast settings.

Parameters

[in]	dwell_interval_ms	Unicast Dwell Interval (15-255 ms)

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the parameters for unicast channel hopping to be used in the following connections. The Unicast Dwell Interval specifies the duration which the node will listen to a channel within its listening schedule. The default value is 255 ms.

Definition at line 386 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_device_private_key_id

```
sl_status_t sl_wisun_set_device_private_key_id (uint32_t key_id)
```

Set the private key of the device certificate.

Parameters

[in]	key_id	Key ID of the private key

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the device private key using a key identifier, referencing a key stored in PSA cryptography module. The corresponding device certificate must still be set using sl_wisun_set_device_certificate(). The stored key must have correct PSA key attributes, see the Wi-SUN FAN Security Concepts and Design Considerations document for details.

Definition at line 401 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_regulation

```
sl_status_t sl_wisun_set_regulation (sl_wisun_regulation_t regulation)
```

Set the regional regulation.

Parameters

[in]	regulation	Regional regulation

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the regional regulation for use in the following connections. The selected regional regulation will impact both the Wi-SUN stack performance and its behavior. See regulation standards for details. No regulation is set by default.

Definition at line 414 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_regulation_tx_thresholds

```
sl_status_t sl_wisun_set_regulation_tx_thresholds (int8_t warning_threshold, int8_t alert_threshold)
```

Set the thresholds for transmission duration level event.

Parameters

[in]	warning_threshold	Warning threshold in percent or -1 to disable
[in]	alert_threshold	Alert threshold in percent or -1 to disable

Returns

SL_STATUS_OK if successful, an error code otherwise

This function sets the thresholds for transmission duration level event. When set and when a regional regulation is enabled using sl_wisun_set_regulation(), a SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID event is sent when one of the configured thresholds is exceeded. This can be used by the application to prevent exceeding the total transmission duration allowed in the regional regulation. Thresholds are defined as a percentage of the maximum transmission duration permitted by the regional regulation.

Definition at line 431 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_advert_fragment_duration

```
sl_status_t sl_wisun_set_advert_fragment_duration (uint32_t fragment_duration_ms)
```

Set the async transmission fragmentation parameters.

Parameters

[in]	fragment_duration_ms	Max duration of a fragment in ms (min 500 ms)

Returns

- SL_STATUS_OK if successful, an error code otherwise

Async transmissions, such as Wi-SUN PAN advertisement packets, are sent to every allowed operating channel and may therefore block broadcast and unicast traffic. This impact can be reduced by splitting the channel list into fragments based on the maximum transmission duration and by forcing a delay between the fragments, allowing other traffic to occur. This function sets the maximum duration of a PA, PAS, PC, and PCS advertisement period fragments. A small value trades off longer connection times for shorter latencies. Setting the duration to SL_WISUN_ADVERT_FRAGMENT_DISABLE disables async transmission fragmentation.

By default, the maximum fragment duration is set to 500 ms.

Definition at line 452 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_unicast_tx_mode

```
sl_status_t sl_wisun_set_unicast_tx_mode (uint8_t mode)
```

Enable an algorithm that trades off unicast communication reliability for latency.

Parameters

[in]	mode	Transmission mode to use • SL_WISUN_UNICAST_TX_MODE_DEFAULT: Default transmission mode. • SL_WISUN_UNICAST_TX_MODE_SLOT: High reliability, high latency.

Returns

- SL_STATUS_OK if successful, an error code otherwise

Enable an algorithm that trades off unicast communication reliability for latency. The mechanism is only effective when all the neighbors are enabled. Enabling this option is detrimental when used with unaware Wi-SUN devices.

Definition at line 467 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_device_type

sl_status_t sl_wisun_set_device_type (sl_wisun_device_type_t device_type)

Set the device type.

Parameters

[in]	device_type	Type of the device

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the operational mode of the node.

Definition at line 477 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_config_mode_switch

sl_status_t sl_wisun_config_mode_switch (uint8_t mode, uint8_t phy_mode_id, const sl_wisun_mac_address_t *neighbor_address, bool reserved)

Set the mode switch configuration.

Parameters

[in]	mode	Mode switch configuration of the neighbor. If set to SL_WISUN_MODE_SWITCH_DEFAULT, the configuration of the neighbor is reset back to the default mode switch behavior.
[in]	phy_mode_id	PhyModeId to use when mode is set to SL_WISUN_MODE_SWITCH_ENABLED, ignored otherwise.
[in]	neighbor_address	MAC address of the neighbor to configure. If set to sl_wisun_broadcast_mac, configures the default mode switch behavior for all non-configured neighbors.
[in]	reserved	Reserved for future use, set to false.

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 495 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_mode_switch

sl_status_t sl_wisun_set_mode_switch (uint8_t mode, uint8_t phy_mode_id, const sl_wisun_mac_address_t *neighbor_address)

Set the PHY mode switch configuration.

Parameters

[in]	mode	Mode switch configuration of the neighbor. If set to SL_WISUN_MODE_SWITCH_DEFAULT, the configuration of the neighbor is reset back to the default mode switch behavior.
[in]	phy_mode_id	PhyModeId to use when mode is set to SL_WISUN_MODE_SWITCH_ENABLED, ignored otherwise.
[in]	neighbor_address	MAC address of the neighbor to configure. If set to sl_wisun_broadcast_mac, configures the default mode switch behavior for all non-configured neighbors.

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 515 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_connection_parameters

```
sl_status_t sl_wisun_set_connection_parameters (const sl_wisun_connection_params_t *params)
```

Configure the FFN parameter set.

Parameters

[in]	params	Parameter set to use

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the FFN parameter set. These parameters impact connection time, bandwidth usage, and latency. Use of a predefined parameter set is recommended (Predefined FFN parameter sets). The function must be called before initiating a connection.

Definition at line 530 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_pom_ie

```
sl_status_t sl_wisun_set_pom_ie (uint8_t phy_mode_id_count, uint8_t
phy_mode_ids[SL_WISUN_MAX_PHY_MODE_ID_COUNT], uint8_t is_mdr_command_capable)
```

Set the POM-IE configuration.

Parameters

[in]	phy_mode_id_count	Number of PhyModeId to configure
[in]	phy_mode_ids	List of phy_mode_id_count PhyModeId. It must contain the base operating mode.
[in]	is_mdr_command_capable	Indicate if the device supports MAC mode switch. Feature currently unsupported, must be set to 0.

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the PHY operating mode information advertised to neighboring nodes. By default the PhyModeId list contains the first fifteen PhyModeId listed in radio multi-PHY configuration, MAC mode switch is disabled.

Definition at line 546 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_pom_ie

```
sl_status_t sl_wisun_get_pom_ie (uint8_t *phy_mode_id_count, uint8_t *phy_mode_ids, uint8_t *is_mdr_command_capable)
```

Get the POM-IE configuration.

Parameters

[out]	phy_mode_id_count	Number of PhyModeId retrieved

[out]	phy_mode_ids	List of phy_mode_id_count PhyModeId. Caller must allocate space for at least SL_WISUN_MAX_PHY_MODE_ID_COUNT entries.
[out]	is_mdr_command_capable	Set to 1 if the device supports MAC mode switch, 0 otherwise

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function retrieves the PHY operating mode information advertised to neighboring nodes.

Definition at line 563 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_stack_version

```
sl_status_t sl_wisun_get_stack_version (uint8_t *major, uint8_t *minor, uint8_t *patch, uint16_t *build)
```

Get the Wi-SUN stack version.

Parameters

[out]	major	Wi-SUN stack version major
[out]	minor	Wi-SUN stack version minor
[out]	patch	Wi-SUN stack version patch
[out]	build	Build number, set to 0 in public versions

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 576 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_lfn_parameters

```
sl_status_t sl_wisun_set_lfn_parameters (const sl_wisun_lfn_params_t *params)
```

Configure the LFN parameter set.

Parameters

[in]	params	Parameter set to use

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the LFN parameter set. These parameters impact connection time, bandwidth usage, power consumption, and latency. Use of a predefined parameter set is recommended (Predefined LFN parameter sets). The function must be called before initiating a connection.

Definition at line 593 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_lfn_support

```
sl_status_t sl_wisun_set_lfn_support (uint8_t lfn_limit)
```

Set the maximum number of LFN children.

Parameters

[in]	lfn_limit	Maximum number of LFN children [0, 10]

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the maximum number of LFN children this node can parent.

Definition at line 604 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_pti_state

```
sl_status_t sl_wisun_set_pti_state (bool pti_state)
```

Set the PTI state.

Parameters

[in]	pti_state	PTI state • **true**: PTI is enabled • **false**: PTI is disabled

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets Packet Trace Interface (PTI) state. PTI is enabled by default.

Definition at line 617 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_trigger_frame

```
sl_status_t sl_wisun_trigger_frame (sl_wisun_frame_type_t frame_type)
```

Trigger the transmission of a frame (FAN Discovery, RPL).

Parameters

[in]	frame_type	Type of frame

Returns

- SL_STATUS_OK if successful, an error code otherwise

No frame is transmitted if the associated Trickle timer is not started, if exists.

Definition at line 628 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_set_security_state

```
sl_status_t sl_wisun_set_security_state (uint32_t security_state)
```

Set the security state.

Parameters

[in]	security_state	Security state • 0: Security is disabled

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 637 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_network_info

```
sl_status_t sl_wisun_get_network_info (sl_wisun_network_info_t *network_info)
```

Get the Wi-SUN network information.

Parameters

[out]	network_info	Pointer to network information

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 645 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_rpl_info

```
sl_status_t sl_wisun_get_rpl_info (sl_wisun_rpl_info_t *rpl_info)
```

Get RPL information.

Parameters

[out]	rpl_info	Pointer to RPL information

Returns

- SL_STATUS_OK if successful, an error code otherwise

Definition at line 653 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

sl_wisun_get_excluded_channel_mask

```
sl_status_t sl_wisun_get_excluded_channel_mask (sl_wisun_channel_mask_type_t type, sl_wisun_channel_mask_t
*channel_mask, uint8_t *channel_count)
```

Get the mask of channels excluded from channel plan.

Parameters

[in]	type	Type of channel mask
[out]	channel_mask	Pointer to mask
[out]	channel_count	Number of channels in mask

Returns

-

SL_STATUS_OK if successful, an error code otherwise

Definition at line 663 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_api.h

Wi-SUN API events

Wi-SUN API events

Modules

Enumerations

enum sl_wisun_msg_ind_id_t {

SL_WISUN_MSG_CONNECTED_IND_ID = 0×81
SL_WISUN_MSG_SOCKET_DATA_IND_ID = 0×82
SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID = 0×83
SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID = 0×84
SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID = 0×85
SL_WISUN_MSG_SOCKET_CLOSING_IND_ID = 0×86
SL_WISUN_MSG_DISCONNECTED_IND_ID = 0×87
SL_WISUN_MSG_CONNECTION_LOST_IND_ID = 0×88
SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID = 0×89
SL_WISUN_MSG_ERROR_IND_ID = 0×8A
SL_WISUN_MSG_JOIN_STATE_IND_ID = 0×8B
SL_WISUN_MSG_NETWORK_UPDATE_IND_ID = 0×8C
SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID = 0×8D
SL_WISUN_MSG_MODE_SWITCH_FALLBACK_IND_ID = 0×8E
SL_WISUN_MSG_RX_FRAME_IND_ID = 0×8F
SL_WISUN_MSG_LFN_WAKE_UP_IND_ID = 0×90
SL_WISUN_MSG_LFN_MULTICAST_REG_IND_ID = 0×91

}

Wi-SUN Message API indication IDs.

Enumeration Documentation

sl_wisun_msg_ind_id_t

> sl_wisun_msg_ind_id_t

Wi-SUN Message API indication IDs.

Enumerator	
SL_WISUN_MSG_CONNECTED_IND_ID	This indication is sent when a connection request has been completed.
SL_WISUN_MSG_SOCKET_DATA_IND_ID	This indication is sent when data has been received on a socket.
SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID	This indication is sent when there is buffered data available on a socket.
SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID	This indication is sent when a socket connect request has been completed.
SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID	This indication is sent when there is a socket connection request waiting.
SL_WISUN_MSG_SOCKET_CLOSING_IND_ID	This event is sent when a socket is waiting to be closed.
SL_WISUN_MSG_DISCONNECTED_IND_ID	This event is sent when a disconnection request has been completed.
SL_WISUN_MSG_CONNECTION_LOST_IND_ID	This event is sent when a connection to Wi-SUN network has been lost and the device is trying to regain the connection.
SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID	This event is sent when part of the buffered socket data has been sent.
SL_WISUN_MSG_ERROR_IND_ID	This event is sent when an internal stack error has occurred.
SL_WISUN_MSG_JOIN_STATE_IND_ID	This event is sent when the join state changes.
SL_WISUN_MSG_NETWORK_UPDATE_IND_ID	This event is sent when the network has been updated.
SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID	This event is sent when regional regulation transmission level changes.
SL_WISUN_MSG_MODE_SWITCH_FALLBACK_IND_ID	This event is sent when the mode switch is disabled.

SL_WISUN_MSG_RX_FRAME_IND_ID	This event is sent on frame receptions.
SL_WISUN_MSG_LFN_WAKE_UP_IND_ID	This event is sent on LFN Wake Up.
SL_WISUN_MSG_LFN_MULTICAST_REG_IND_ID	Indicate a multicast group registration finishes.

Definition at line 43 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_evt_t

Wi-SUN event definitions.

This structure contains a Wi-SUN API event and its associated data.

Public Attributes

sl_wisun_msg_header_t	header Common event header.
sl_wisun_msg_connected_ind_body_t	connected SL_WISUN_MSG_CONNECTED_IND_ID event data
sl_wisun_msg_socket_data_ind_body_t	socket_data SL_WISUN_MSG_SOCKET_DATA_IND_ID event data
sl_wisun_msg_socket_data_available_ind_body_t	socket_data_available SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID event data
sl_wisun_msg_socket_connected_ind_body_t	socket_connected SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID event data
sl_wisun_msg_socket_connection_available_ind_body_t	socket_connection_available SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID event data
sl_wisun_msg_socket_closing_ind_body_t	socket_closing SL_WISUN_MSG_SOCKET_CLOSING_IND_ID event data
sl_wisun_msg_disconnected_ind_body_t	disconnected SL_WISUN_MSG_DISCONNECTED_IND_ID event data
sl_wisun_msg_connection_lost_ind_body_t	connection_lost SL_WISUN_MSG_CONNECTION_LOST_IND_ID event data
sl_wisun_msg_socket_data_sent_ind_body_t	socket_data_sent SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID event data
sl_wisun_msg_error_ind_body_t	error SL_WISUN_MSG_ERROR_IND_ID event data
sl_wisun_msg_join_state_ind_body_t	join_state SL_WISUN_MSG_JOIN_STATE_IND_ID event data
sl_wisun_msg_network_update_ind_body_t	network_update SL_WISUN_MSG_NETWORK_UPDATE_IND_ID event data
sl_wisun_msg_regulation_tx_level_in_	

d_body_t	regulation_tx_level SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID event data
sl_wisun_msg_mode_switch_fallback_ind_body_t	mode_switch_fallback SL_WISUN_MSG_MODE_SWITCH_FALLBACK_IND_ID event data
sl_wisun_msg_rx_frame_ind_body_t	rx_frame SL_WISUN_MSG_RX_FRAME_IND_ID event data
sl_wisun_msg_lfn_wake_up_ind_body_t	lfn_wake_up SL_WISUN_MSG_LFN_WAKE_UP_IND_ID event data
sl_wisun_msg_lfn_multicast_reg_ind_body_t	lfn_multicast_reg SL_WISUN_MSG_LFN_MULTICAST_REG_IND_ID event data
union sl_wisun_evt_t::@0	evt Event-specific data.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_evt_t::header
```

Common event header.

This structure contains common information for all events. ID of the event is stored in the sl_wisun_msg_header_t.id field and is one of the values of sl_wisun_msg_ind_id_t. The other fields can be ignored.

Definition at line 568 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

connected

```
sl_wisun_msg_connected_ind_body_t sl_wisun_evt_t::connected
```

SL_WISUN_MSG_CONNECTED_IND_ID event data

Definition at line 573 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_data

```
sl_wisun_msg_socket_data_ind_body_t sl_wisun_evt_t::socket_data
```

SL_WISUN_MSG_SOCKET_DATA_IND_ID event data

Definition at line 575 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_data_available

sl_wisun_msg_socket_data_available_ind_body_t sl_wisun_evt_t::socket_data_available

SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID event data

Definition at line 577 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_connected

sl_wisun_msg_socket_connected_ind_body_t sl_wisun_evt_t::socket_connected

SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID event data

Definition at line 579 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_connection_available

sl_wisun_msg_socket_connection_available_ind_body_t sl_wisun_evt_t::socket_connection_available

SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID event data

Definition at line 581 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_closing

sl_wisun_msg_socket_closing_ind_body_t sl_wisun_evt_t::socket_closing

SL_WISUN_MSG_SOCKET_CLOSING_IND_ID event data

Definition at line 583 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

disconnected

sl_wisun_msg_disconnected_ind_body_t sl_wisun_evt_t::disconnected

SL_WISUN_MSG_DISCONNECTED_IND_ID event data

Definition at line 585 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

connection_lost

sl_wisun_msg_connection_lost_ind_body_t sl_wisun_evt_t::connection_lost

SL_WISUN_MSG_CONNECTION_LOST_IND_ID event data

Definition at line 587 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_data_sent

> sl_wisun_msg_socket_data_sent_ind_body_t sl_wisun_evt_t::socket_data_sent

SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID event data

Definition at line 589 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

error

> sl_wisun_msg_error_ind_body_t sl_wisun_evt_t::error

SL_WISUN_MSG_ERROR_IND_ID event data

Definition at line 591 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

join_state

> sl_wisun_msg_join_state_ind_body_t sl_wisun_evt_t::join_state

SL_WISUN_MSG_JOIN_STATE_IND_ID event data

Definition at line 593 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

network_update

> sl_wisun_msg_network_update_ind_body_t sl_wisun_evt_t::network_update

SL_WISUN_MSG_NETWORK_UPDATE_IND_ID event data

Definition at line 595 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

regulation_tx_level

> sl_wisun_msg_regulation_tx_level_ind_body_t sl_wisun_evt_t::regulation_tx_level

SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID event data

Definition at line 597 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

mode_switch_fallback

> sl_wisun_msg_mode_switch_fallback_ind_body_t sl_wisun_evt_t::mode_switch_fallback

SL_WISUN_MSG_MODE_SWITCH_FALLBACK_IND_ID event data

Definition at line 599 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

rx_frame

sl_wisun_msg_rx_frame_ind_body_t sl_wisun_evt_t::rx_frame

SL_WISUN_MSG_RX_FRAME_IND_ID event data

Definition at line 601 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

lfn_wake_up

sl_wisun_msg_lfn_wake_up_ind_body_t sl_wisun_evt_t::lfn_wake_up

SL_WISUN_MSG_LFN_WAKE_UP_IND_ID event data

Definition at line 603 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

lfn_multicast_reg

sl_wisun_msg_lfn_multicast_reg_ind_body_t sl_wisun_evt_t::lfn_multicast_reg

SL_WISUN_MSG_LFN_MULTICAST_REG_IND_ID event data

Definition at line 605 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

evt

union sl_wisun_evt_t::@0 sl_wisun_evt_t::evt

Event-specific data.

This structure contains the event-specific data.

Definition at line 606 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_connected_ind

sl_wisun_msg_connected_ind

Modules

sl_wisun_msg_connected_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_connected_ind_body_t::status
```

Status of the indication.

Definition at line 90 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_connected_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_connected_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_connected_ind_t::header
```

Common message header.

Definition at line 98 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_connected_ind_body_t sl_wisun_msg_connected_ind_t::body
```

Indication message body.

Definition at line 100 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_network_update_ind

sl_wisun_msg_network_update_ind

Modules

sl_wisun_msg_network_update_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	flags Bit mask indicating the changes.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_network_update_ind_body_t::status
```

Status of the indication.

Definition at line 115 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

flags

```
uint32_t sl_wisun_msg_network_update_ind_body_t::flags
```

Bit mask indicating the changes.

- bit SL_WISUN_NETWORK_UPDATE_FLAGS_GLOBAL_IP: Global IP address has changed
- bit SL_WISUN_NETWORK_UPDATE_FLAGS_PRIMARY_PARENT: primary parent has changed
- bit SL_WISUN_NETWORK_UPDATE_FLAGS_SECONDARY_PARENT: secondary parent has changed

Definition at line 120 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_network_update_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_network_update_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_network_update_ind_t::header
```

Common message header.

Definition at line 128 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_network_update_ind_body_t sl_wisun_msg_network_update_ind_t::body
```

Indication message body.

Definition at line 130 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_data_ind

sl_wisun_msg_socket_data_ind

Modules

sl_wisun_msg_socket_data_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
int32_t	socket_id ID of the socket.
in6_addr_t	remote_address IP address of the sender.
uint16_t	remote_port Port number of the sender.
uint16_t	data_length Amount of received data.
uint8_t	data Received data.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_socket_data_ind_body_t::status
```

Status of the indication.

Definition at line 145 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_id

```
int32_t sl_wisun_msg_socket_data_ind_body_t::socket_id
```

ID of the socket.

Definition at line 147 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

remote_address

```
in6_addr_t sl_wisun_msg_socket_data_ind_body_t::remote_address
```

IP address of the sender.

Definition at line 149 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

remote_port

```
uint16_t sl_wisun_msg_socket_data_ind_body_t::remote_port
```

Port number of the sender.

Definition at line 151 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

data_length

```
uint16_t sl_wisun_msg_socket_data_ind_body_t::data_length
```

Amount of received data.

Definition at line 153 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

data

```
uint8_t sl_wisun_msg_socket_data_ind_body_t::data[]
```

Received data.

Definition at line 155 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_data_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_socket_data_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_data_ind_t::header
```

Common message header.

Definition at line 163 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_socket_data_ind_body_t sl_wisun_msg_socket_data_ind_t::body
```

Indication message body.

Definition at line 165 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_data_available_ind

sl_wisun_msg_socket_data_available_ind

Modules

sl_wisun_msg_socket_data_available_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	socket_id ID of the socket.
uint16_t	data_length Amount of data that can be read.
uint16_t	reserved Reserved, set to zero.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_socket_data_available_ind_body_t::status
```

Status of the indication.

Definition at line 180 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_id

```
uint32_t sl_wisun_msg_socket_data_available_ind_body_t::socket_id
```

ID of the socket.

Definition at line 182 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

data_length

```
uint16_t sl_wisun_msg_socket_data_available_ind_body_t::data_length
```

Amount of data that can be read.

Definition at line 184 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

reserved

uint16_t sl_wisun_msg_socket_data_available_ind_body_t::reserved

Reserved, set to zero.

Definition at line 186 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_data_available_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_socket_data_available_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_data_available_ind_t::header
```

Common message header.

Definition at line 194 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_socket_data_available_ind_body_t sl_wisun_msg_socket_data_available_ind_t::body
```

Indication message body.

Definition at line 196 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_connected_ind

sl_wisun_msg_socket_connected_ind

Modules

sl_wisun_msg_socket_connected_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	socket_id ID of the socket.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_socket_connected_ind_body_t::status
```

Status of the indication.

Definition at line 211 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_id

```
uint32_t sl_wisun_msg_socket_connected_ind_body_t::socket_id
```

ID of the socket.

Definition at line 213 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_connected_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_socket_connected_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_connected_ind_t::header
```

Common message header.

Definition at line 221 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_socket_connected_ind_body_t sl_wisun_msg_socket_connected_ind_t::body
```

Indication message body.

Definition at line 223 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_connection_available_ind

sl_wisun_msg_socket_connection_available_ind

Modules

sl_wisun_msg_socket_connection_available_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	socket_id ID of the socket.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_socket_connection_available_ind_body_t::status
```

Status of the indication.

Definition at line 238 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_id

```
uint32_t sl_wisun_msg_socket_connection_available_ind_body_t::socket_id
```

ID of the socket.

Definition at line 240 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_connection_available_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_socket_connection_available_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_connection_available_ind_t::header
```

Common message header.

Definition at line 248 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_socket_connection_available_ind_body_t sl_wisun_msg_socket_connection_available_ind_t::body
```

Indication message body.

Definition at line 250 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_closing_ind

sl_wisun_msg_socket_closing_ind

Modules

sl_wisun_msg_socket_closing_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	socket_id ID of the socket.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_socket_closing_ind_body_t::status
```

Status of the indication.

Definition at line 265 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_id

```
uint32_t sl_wisun_msg_socket_closing_ind_body_t::socket_id
```

ID of the socket.

Definition at line 267 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_closing_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_socket_closing_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_closing_ind_t::header
```

Common message header.

Definition at line 275 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_socket_closing_ind_body_t sl_wisun_msg_socket_closing_ind_t::body
```

Indication message body.

Definition at line 277 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_disconnected_ind

sl_wisun_msg_disconnected_ind

Modules

sl_wisun_msg_disconnected_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.

Public Attribute Documentation

status

> uint32_t sl_wisun_msg_disconnected_ind_body_t::status

Status of the indication.

Definition at line 292 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_disconnected_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_disconnected_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_disconnected_ind_t::header
```

Common message header.

Definition at line 300 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_disconnected_ind_body_t sl_wisun_msg_disconnected_ind_t::body
```

Indication message body.

Definition at line 302 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_connection_lost_ind

sl_wisun_msg_connection_lost_ind

Modules

sl_wisun_msg_connection_lost_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_connection_lost_ind_body_t::status
```

Status of the indication.

Definition at line 317 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_connection_lost_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_connection_lost_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_connection_lost_ind_t::header
```

Common message header.

Definition at line 325 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_connection_lost_ind_body_t sl_wisun_msg_connection_lost_ind_t::body
```

Indication message body.

Definition at line 327 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_data_sent_ind

sl_wisun_msg_socket_data_sent_ind

Modules

sl_wisun_msg_socket_data_sent_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
int32_t	socket_id ID of the socket.
uint32_t	socket_space_left Amount of free space in the transmission buffer.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_socket_data_sent_ind_body_t::status
```

Status of the indication.

Definition at line 342 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_id

```
int32_t sl_wisun_msg_socket_data_sent_ind_body_t::socket_id
```

ID of the socket.

Definition at line 344 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

socket_space_left

```
uint32_t sl_wisun_msg_socket_data_sent_ind_body_t::socket_space_left
```

Amount of free space in the transmission buffer.

Definition at line 346 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_socket_data_sent_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_socket_data_sent_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_data_sent_ind_t::header
```

Common message header.

Definition at line 354 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_socket_data_sent_ind_body_t sl_wisun_msg_socket_data_sent_ind_t::body
```

Indication message body.

Definition at line 356 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_error_ind

sl_wisun_msg_error_ind

Modules

sl_wisun_msg_error_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_error_ind_body_t::status
```

Status of the indication.

Definition at line 371 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_error_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_error_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_error_ind_t::header
```

Common message header.

Definition at line 379 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_error_ind_body_t sl_wisun_msg_error_ind_t::body
```

Indication message body.

Definition at line 381 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_join_state_ind

sl_wisun_msg_join_state_ind

Modules

sl_wisun_msg_join_state_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	join_state Join state.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_join_state_ind_body_t::status
```

Status of the indication.

Definition at line 396 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

join_state

```
uint32_t sl_wisun_msg_join_state_ind_body_t::join_state
```

Join state.

Definition at line 398 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_join_state_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_join_state_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_join_state_ind_t::header
```

Common message header.

Definition at line 406 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_join_state_ind_body_t sl_wisun_msg_join_state_ind_t::body
```

Indication message body.

Definition at line 408 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_regulation_tx_level_ind

sl_wisun_msg_regulation_tx_level_ind

Modules

sl_wisun_msg_regulation_tx_level_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	tx_duration_ms Sum of transmission durations during last hour in milliseconds.
uint8_t	tx_level Transmission level, one value of sl_wisun_regulation_tx_level_t.
uint8_t	reserved Reserved.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_regulation_tx_level_ind_body_t::status
```

Status of the indication.

Definition at line 423 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

tx_duration_ms

```
uint32_t sl_wisun_msg_regulation_tx_level_ind_body_t::tx_duration_ms
```

Sum of transmission durations during last hour in milliseconds.

Definition at line 425 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

tx_level

```
uint8_t sl_wisun_msg_regulation_tx_level_ind_body_t::tx_level
```

Transmission level, one value of sl_wisun_regulation_tx_level_t.

Definition at line 427 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

reserved

uint8_t sl_wisun_msg_regulation_tx_level_ind_body_t::reserved[3]

Reserved.

Definition at line 429 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_regulation_tx_level_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_regulation_tx_level_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_regulation_tx_level_ind_t::header
```

Common message header.

Definition at line 437 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_regulation_tx_level_ind_body_t sl_wisun_msg_regulation_tx_level_ind_t::body
```

Indication message body.

Definition at line 439 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_mode_switch_fallback_level_ind

sl_wisun_mode_switch_fallback_level_ind

Modules

sl_wisun_msg_mode_switch_fallback_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
sl_wisun_mac_address_t	address MAC address of the peer triggering the fallback.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_mode_switch_fallback_ind_body_t::status
```

Status of the indication.

Definition at line 454 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

address

```
sl_wisun_mac_address_t sl_wisun_msg_mode_switch_fallback_ind_body_t::address
```

MAC address of the peer triggering the fallback.

Definition at line 456 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_mode_switch_fallback_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_mode_switch_fallback_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_mode_switch_fallback_ind_t::header
```

Common message header.

Definition at line 464 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_mode_switch_fallback_ind_body_t sl_wisun_msg_mode_switch_fallback_ind_t::body
```

Indication message body.

Definition at line 466 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_rx_frame_ind

sl_wisun_msg_rx_frame_ind

Modules

sl_wisun_msg_rx_frame_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint64_t	timestamp_us Timestamp in microseconds.
uint16_t	length Frame length in bytes.
uint8_t	frame Received frame.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_rx_frame_ind_body_t::status
```

Status of the indication.

Definition at line 481 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

timestamp_us

```
uint64_t sl_wisun_msg_rx_frame_ind_body_t::timestamp_us
```

Timestamp in microseconds.

Definition at line 483 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

length

```
uint16_t sl_wisun_msg_rx_frame_ind_body_t::length
```

Frame length in bytes.

Definition at line 485 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

frame

> uint8_t sl_wisun_msg_rx_frame_ind_body_t::frame[]

Received frame.

Definition at line 487 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_rx_frame_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_rx_frame_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_rx_frame_ind_t::header
```

Common message header.

Definition at line 495 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_rx_frame_ind_body_t sl_wisun_msg_rx_frame_ind_t::body
```

Indication message body.

Definition at line 497 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_lfn_wake_up_ind

sl_wisun_msg_lfn_wake_up_ind

Modules

sl_wisun_msg_lfn_wake_up_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
uint32_t	wup_duration_us Expected wake-up duration in microseconds.
uint64_t	next_wup_us Expected time to next wake-up in microseconds.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_lfn_wake_up_ind_body_t::status
```

Status of the indication.

Definition at line 512 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

wup_duration_us

```
uint32_t sl_wisun_msg_lfn_wake_up_ind_body_t::wup_duration_us
```

Expected wake-up duration in microseconds.

Definition at line 514 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

next_wup_us

```
uint64_t sl_wisun_msg_lfn_wake_up_ind_body_t::next_wup_us
```

Expected time to next wake-up in microseconds.

Definition at line 516 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_lfn_wake_up_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_lfn_wake_up_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_lfn_wake_up_ind_t::header
```

Common message header.

Definition at line 524 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_lfn_wake_up_ind_body_t sl_wisun_msg_lfn_wake_up_ind_t::body
```

Indication message body.

Definition at line 526 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_lfn_multicast_reg_ind

sl_wisun_msg_lfn_multicast_reg_ind

Modules

sl_wisun_msg_lfn_multicast_reg_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.
in6_addr_t	ip_address Registered multicast IP address.

Public Attribute Documentation

status

```
uint32_t sl_wisun_msg_lfn_multicast_reg_ind_body_t::status
```

Status of the indication.

Definition at line 541 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

ip_address

```
in6_addr_t sl_wisun_msg_lfn_multicast_reg_ind_body_t::ip_address
```

Registered multicast IP address.

Definition at line 543 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

sl_wisun_msg_lfn_multicast_reg_ind_t

Indication message.

Public Attributes

sl_wisun_msg_header_t	header Common message header.
sl_wisun_msg_lfn_multicast_reg_ind_body_t	body Indication message body.

Public Attribute Documentation

header

```
sl_wisun_msg_header_t sl_wisun_msg_lfn_multicast_reg_ind_t::header
```

Common message header.

Definition at line 551 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

body

```
sl_wisun_msg_lfn_multicast_reg_ind_body_t sl_wisun_msg_lfn_multicast_reg_ind_t::body
```

Indication message body.

Definition at line 553 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_events.h

Wi-SUN API type definitions

Modules

Enumerations

enum sl_wisun_device_type_t {

SL_WISUN_ROUTER = 0
SL_WISUN_LFN = 1
SL_WISUN_BORDER_ROUTER = 2

}

Enumerations for device type.

enum sl_wisun_network_size_t {

SL_WISUN_NETWORK_SIZE_AUTOMATIC = 0
SL_WISUN_NETWORK_SIZE_SMALL = 1
SL_WISUN_NETWORK_SIZE_MEDIUM = 2
SL_WISUN_NETWORK_SIZE_LARGE = 3
SL_WISUN_NETWORK_SIZE_TEST = 4
SL_WISUN_NETWORK_SIZE_CERTIFICATION = 5

}

Enumerations for network size.

enum sl_wisun_ip_address_type_t {

SL_WISUN_IP_ADDRESS_TYPE_LINK_LOCAL = 0
SL_WISUN_IP_ADDRESS_TYPE_GLOBAL = 1
SL_WISUN_IP_ADDRESS_TYPE_BORDER_ROUTER = 2
SL_WISUN_IP_ADDRESS_TYPE_PRIMARY_PARENT = 3
SL_WISUN_IP_ADDRESS_TYPE_SECONDARY_PARENT = 4

}

Enumerations for IP address type.

enum sl_wisun_certificate_option_t {

SL_WISUN_CERTIFICATE_OPTION_NONE = 0
SL_WISUN_CERTIFICATE_OPTION_APPEND = 1
SL_WISUN_CERTIFICATE_OPTION_IS_REF = 2
SL_WISUN_CERTIFICATE_OPTION_HAS_KEY = 4

}

Enumerations for certificate options.

enum sl_wisun_private_key_option_t {

SL_WISUN_PRIVATE_KEY_OPTION_NONE = 0
SL_WISUN_PRIVATE_KEY_OPTION_IS_REF = 1

}

Enumerations for private key options.

enum sl_wisun_statistics_type_t {

SL_WISUN_STATISTICS_TYPE_PHY = 0
SL_WISUN_STATISTICS_TYPE_MAC = 1
SL_WISUN_STATISTICS_TYPE_FHSS = 2
SL_WISUN_STATISTICS_TYPE_WISUN = 3
SL_WISUN_STATISTICS_TYPE_NETWORK = 4
SL_WISUN_STATISTICS_TYPE_REGULATION = 5
SL_WISUN_STATISTICS_TYPE_HEAP = 6

}

Enumerations for statistics type.

enum sl_wisun_regulatory_domain_t {

SL_WISUN_REGULATORY_DOMAIN_WW = 0
SL_WISUN_REGULATORY_DOMAIN_NA = 1
SL_WISUN_REGULATORY_DOMAIN_JP = 2
SL_WISUN_REGULATORY_DOMAIN_EU = 3
SL_WISUN_REGULATORY_DOMAIN_CN = 4
SL_WISUN_REGULATORY_DOMAIN_IN = 5
SL_WISUN_REGULATORY_DOMAIN_MX = 6
SL_WISUN_REGULATORY_DOMAIN_BZ = 7
SL_WISUN_REGULATORY_DOMAIN_AZ = 8
SL_WISUN_REGULATORY_DOMAIN_NZ = 8
SL_WISUN_REGULATORY_DOMAIN_KR = 9
SL_WISUN_REGULATORY_DOMAIN_PH = 10
SL_WISUN_REGULATORY_DOMAIN_MY = 11
SL_WISUN_REGULATORY_DOMAIN_HK = 12
SL_WISUN_REGULATORY_DOMAIN_SG = 13
SL_WISUN_REGULATORY_DOMAIN_TH = 14
SL_WISUN_REGULATORY_DOMAIN_VN = 15
SL_WISUN_REGULATORY_DOMAIN_APP = 255

}

Enumerations for regulatory domain.

enum sl_wisun_operating_class_t {

SL_WISUN_OPERATING_CLASS_1 = 1
SL_WISUN_OPERATING_CLASS_2 = 2
SL_WISUN_OPERATING_CLASS_3 = 3
SL_WISUN_OPERATING_CLASS_4 = 4
SL_WISUN_OPERATING_CLASS_5 = 5
SL_WISUN_OPERATING_CLASS_APP = 255

}

Enumerations for operating class.

enum sl_wisun_operating_mode_t {

SL_WISUN_OPERATING_MODE_1A = 0x1a
SL_WISUN_OPERATING_MODE_1B = 0x1b
SL_WISUN_OPERATING_MODE_2A = 0x2a
SL_WISUN_OPERATING_MODE_2B = 0x2b
SL_WISUN_OPERATING_MODE_3 = 0x03
SL_WISUN_OPERATING_MODE_4A = 0x4a
SL_WISUN_OPERATING_MODE_4B = 0x4b
SL_WISUN_OPERATING_MODE_5 = 0x05

}

Enumerations for operating mode.

enum sl_wisun_multicast_group_action_t {

SL_WISUN_MULTICAST_GROUP_ACTION_JOIN = 0
SL_WISUN_MULTICAST_GROUP_ACTION_LEAVE = 1

}

Enumerations for multicast group action.

enum sl_wisun_channel_spacing_t {

SL_WISUN_CHANNEL_SPACING_100KHZ = 0
SL_WISUN_CHANNEL_SPACING_200KHZ = 1
SL_WISUN_CHANNEL_SPACING_400KHZ = 2
SL_WISUN_CHANNEL_SPACING_600KHZ = 3
SL_WISUN_CHANNEL_SPACING_250KHZ = 4
SL_WISUN_CHANNEL_SPACING_800KHZ = 5
SL_WISUN_CHANNEL_SPACING_1200KHZ = 6

}

Enumerations for channel spacing.

enum sl_wisun_join_state_t {

SL_WISUN_JOIN_STATE_DISCONNECTED = 0
SL_WISUN_JOIN_STATE_SELECT_PAN = 1
SL_WISUN_JOIN_STATE_AUTHENTICATE = 2
SL_WISUN_JOIN_STATE_ACQUIRE_PAN_CONFIG = 3
SL_WISUN_JOIN_STATE_CONFIGURE_ROUTING = 4
SL_WISUN_JOIN_STATE_OPERATIONAL = 5
SL_WISUN_JOIN_STATE_PARENT_SELECT = 41
SL_WISUN_JOIN_STATE_DHCP = 42
SL_WISUN_JOIN_STATE_EARO = 43
SL_WISUN_JOIN_STATE_DAO = 44

}

Enumerations for join state.

enum sl_wisun_network_update_flags_t {

SL_WISUN_NETWORK_UPDATE_FLAGS_GLOBAL_IP = 0
SL_WISUN_NETWORK_UPDATE_FLAGS_PRIMARY_PARENT = 1
SL_WISUN_NETWORK_UPDATE_FLAGS_SECONDARY_PARENT = 2

}

Enumerations for network update flags.

enum sl_wisun_phy_config_type_t {

SL_WISUN_PHY_CONFIG_FAN10 = 0
SL_WISUN_PHY_CONFIG_FAN11 = 1
SL_WISUN_PHY_CONFIG_EXPLICIT = 2
SL_WISUN_PHY_CONFIG_IDS = 3
SL_WISUN_PHY_CONFIG_CUSTOM_FSK = 4
SL_WISUN_PHY_CONFIG_CUSTOM_OFDM = 5
SL_WISUN_PHY_CONFIG_CUSTOM_OQPSK = 6

}

Enumerations for PHY config type.

enum sl_wisun_lfn_profile_t {

SL_WISUN_LFN_PROFILE_TEST = 0
SL_WISUN_LFN_PROFILE_BALANCED = 1
SL_WISUN_LFN_PROFILE_ECO = 2

}

Enumeration for LFN configuration profile.

enum sl_wisun_crc_type_t {

SL_WISUN_NO_CRC = 0
SL_WISUN_2_BYTES_CRC = 1
SL_WISUN_4_BYTES_CRC = 2

}

Enumeration for CRC type.

enum sl_wisun_socket_protocol_t {

SL_WISUN_SOCKET_PROTOCOL_UDP = 0
SL_WISUN_SOCKET_PROTOCOL_TCP = 1
SL_WISUN_SOCKET_PROTOCOL_ICMP = 2

}

Enumerations for socket protocol Deprecated.

enum sl_wisun_socket_option_t {

SL_WISUN_SOCKET_OPTION_EVENT_MODE = 0
SL_WISUN_SOCKET_OPTION_MULTICAST_GROUP = 1
SL_WISUN_SOCKET_OPTION_SEND_BUFFER_LIMIT = 2
SL_WISUN_SOCKET_OPTION_EDFE_MODE = 3
SL_WISUN_SOCKET_OPTION_UNICAST_HOP_LIMIT = 4
SL_WISUN_SOCKET_OPTION_MULTICAST_HOP_LIMIT = 5

}

Enumerations for socket option Deprecated.

enum sl_wisun_neighbor_type_t {

SL_WISUN_NEIGHBOR_TYPE_PRIMARY_PARENT = 0
SL_WISUN_NEIGHBOR_TYPE_SECONDARY_PARENT = 1
SL_WISUN_NEIGHBOR_TYPE_CHILD = 2

}

Enumeration for RPL neighbor types.

enum sl_wisun_trace_group_t {

SL_WISUN_TRACE_GROUP_MAC = 0
SL_WISUN_TRACE_GROUP_NW = 1
SL_WISUN_TRACE_GROUP_LLC = 2
SL_WISUN_TRACE_GROUP_6LO = 3
SL_WISUN_TRACE_GROUP_IPV6 = 4
SL_WISUN_TRACE_GROUP_TCP = 5
SL_WISUN_TRACE_GROUP_UDP = 6
SL_WISUN_TRACE_GROUP_ICMP = 7
SL_WISUN_TRACE_GROUP_DHCP = 8
SL_WISUN_TRACE_GROUP_MPL = 9
SL_WISUN_TRACE_GROUP_DNS = 10
SL_WISUN_TRACE_GROUP_RPL = 11
SL_WISUN_TRACE_GROUP_TRIC = 12
SL_WISUN_TRACE_GROUP_WS = 15
SL_WISUN_TRACE_GROUP_BOOT = 16
SL_WISUN_TRACE_GROUP_WSR = 17
SL_WISUN_TRACE_GROUP_WSBR = 18
SL_WISUN_TRACE_GROUP_SEC = 19
SL_WISUN_TRACE_GROUP_TIME = 20
SL_WISUN_TRACE_GROUP_NEIGH = 21
SL_WISUN_TRACE_GROUP_STAT = 22
SL_WISUN_TRACE_GROUP_BUFF = 23
SL_WISUN_TRACE_GROUP_ADDR = 24
SL_WISUN_TRACE_GROUP_MON = 25
SL_WISUN_TRACE_GROUP_SOCK = 26
SL_WISUN_TRACE_GROUP_DENY = 27
SL_WISUN_TRACE_GROUP_ETX = 28
SL_WISUN_TRACE_GROUP_FHSS = 29
SL_WISUN_TRACE_GROUP_ROUT = 30
SL_WISUN_TRACE_GROUP_EVLP = 31
SL_WISUN_TRACE_GROUP_NVM = 32
SL_WISUN_TRACE_GROUP_CRYPTO = 33
SL_WISUN_TRACE_GROUP_RF = 34
SI_WISUN_TRACE_GROUP_WSIE = 35
SL_WISUN_TRACE_GROUP_CONFIG = 36
SL_WISUN_TRACE_GROUP_TIM_SRV = 37
SL_WISUN_TRACE_GROUP_LFN_TIM = 38
SL_WISUN_TRACE_GROUP_INT = 63
SL_WISUN_TRACE_GROUP_COUNT = 64

}

Enumeration for trace group.

enum sl_wisun_trace_level_t {

SL_WISUN_TRACE_LEVEL_NONE = 0
SL_WISUN_TRACE_LEVEL_ERROR = 1
SL_WISUN_TRACE_LEVEL_WARN = 2
SL_WISUN_TRACE_LEVEL_INFO = 3
SL_WISUN_TRACE_LEVEL_DEBUG = 4

}

Enumerations for trace level.

enum sl_wisun_regulation_t {

SL_WISUN_REGULATION_NONE = 0
SL_WISUN_REGULATION_ARIB = 1

}

Enumerations for regional regulation.

enum sl_wisun_mode_switch_mode_t {

SL_WISUN_MODE_SWITCH_DISABLED = 0
SL_WISUN_MODE_SWITCH_ENABLED = 1
SL_WISUN_MODE_SWITCH_DEFAULT = 2

}

Enumeration for Mode Switch mode.

enum sl_wisun_regulation_tx_level_t {

SL_WISUN_REGULATION_TX_LEVEL_LOW = 0
SL_WISUN_REGULATION_TX_LEVEL_WARNING = 1
SL_WISUN_REGULATION_TX_LEVEL_ALERT = 2

}

Enumeration for regional regulation transmission level.

enum sl_wisun_unicast_tx_mode_t {

SL_WISUN_UNICAST_TX_MODE_DEFAULT = 0
SL_WISUN_UNICAST_TX_MODE_SLOT = 1

}

Enumeration for unicast transmission mode.

enum sl_wisun_channel_exclusion_mode_t {

SL_WISUN_CHANNEL_EXCLUSION_MODE_BY_RANGE = 1
SL_WISUN_CHANNEL_EXCLUSION_MODE_BY_MASK = 2

}

Enumeration for channel exlusion modes.

enum sl_wisun_frame_type_t {

SL_WISUN_FRAME_TYPE_PAS = 0
SL_WISUN_FRAME_TYPE_PA = 1
SL_WISUN_FRAME_TYPE_PCS = 2
SL_WISUN_FRAME_TYPE_PC = 3
SL_WISUN_FRAME_TYPE_DIS = 4
SL_WISUN_FRAME_TYPE_DIO = 5

}

Enumeration for types of frame that can be triggered.

enum sl_wisun_channel_mask_type_t {

SL_WISUN_CHANNEL_MASK_TYPE_REGIONAL
SL_WISUN_CHANNEL_MASK_TYPE_ADVERTISED_UNICAST
SL_WISUN_CHANNEL_MASK_TYPE_ADVERTISED_BROADCAST
SL_WISUN_CHANNEL_MASK_TYPE_EFFECTIVE_ASYNC
SL_WISUN_CHANNEL_MASK_TYPE_EFFECTIVE_UNICAST
SL_WISUN_CHANNEL_MASK_TYPE_EFFECTIVE_BROADCAST

}

Enumeration for channel mask types.

Typedefs

typedef in6_addr_t sl_wisun_ip_address_t

Variables

const sl_wisun_mac_address_t sl_wisun_broadcast_mac

Broadcast MAC address.

Macros

#define SL_WISUN_NETWORK_NAME_SIZE 32
Maximum size of the Wi-SUN network name.

#define SL_WISUN_MAC_ADDRESS_SIZE 8
Size of a MAC address.

#define SL_WISUN_CHANNEL_MASK_SIZE 32
Size of a channel mask.

#define SL_WISUN_FILTER_BITFIELD_SIZE ((SL_WISUN_TRACE_GROUP_COUNT + 7) / 8)
Size of the filter bitfield.

#define SL_WISUN_ADVERT_FRAGMENT_DISABLE UINT32_MAX
Maximum fragment duration. Disables advert fragmentation.

#define SL_WISUN_MAX_PHY_MODE_ID_COUNT 15
Maximum number of PhyModeId allowed in POM-IE.

#define SL_WISUN_CHANNEL_SPACING_100HZ SL_WISUN_CHANNEL_SPACING_100KHZ
Channel spacing 100 kHz for backward compatibility.

#define SL_WISUN_CHANNEL_SPACING_200HZ SL_WISUN_CHANNEL_SPACING_200KHZ
Channel spacing 200 kHz for backward compatibility.

#define SL_WISUN_CHANNEL_SPACING_400HZ SL_WISUN_CHANNEL_SPACING_400KHZ
Channel spacing 400 kHz for backward compatibility.

#define SL_WISUN_CHANNEL_SPACING_600HZ SL_WISUN_CHANNEL_SPACING_600KHZ
Channel spacing 600 kHz for backward compatibility.

#define SL_WISUN_TRACE_THREAD_WISUN "WS"
Thread identifier "Wi-SUN Task".

#define SL_WISUN_TRACE_THREAD_EVENT_TASK "EVT"
Thread identifier "Wi-SUN Event Task".

#define SL_WISUN_TRACE_THREAD_EVENT_LOOP "EVL"
Thread identifier "Wi-SUN Event Loop Task".

#define SL_WISUN_TRACE_THREAD_MAC "MAC"
Thread identifier "Wi-SUN RF Task".

Enumeration Documentation

sl_wisun_device_type_t

> sl_wisun_device_type_t

Enumerations for device type.

Enumerator	
SL_WISUN_ROUTER	FFN Router.
SL_WISUN_LFN	LFN Router (experimental, for evaluation purposes only)
SL_WISUN_BORDER_ROUTER	Border Router.

Definition at line 58 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_network_size_t

> sl_wisun_network_size_t

Enumerations for network size.

Enumerator	
SL_WISUN_NETWORK_SIZE_AUTOMATIC	Determine the size from PAN advertisements.
SL_WISUN_NETWORK_SIZE_SMALL	Small size (less than 100 nodes)
SL_WISUN_NETWORK_SIZE_MEDIUM	Medium size (100 to 800 nodes)
SL_WISUN_NETWORK_SIZE_LARGE	Large size (800 to 1500 nodes)
SL_WISUN_NETWORK_SIZE_TEST	Test network (a few nodes)
SL_WISUN_NETWORK_SIZE_CERTIFICATION	Certification configuration.

Definition at line `68` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_ip_address_type_t

> sl_wisun_ip_address_type_t

Enumerations for IP address type.

Enumerator	
SL_WISUN_IP_ADDRESS_TYPE_LINK_LOCAL	Device link-local address.
SL_WISUN_IP_ADDRESS_TYPE_GLOBAL	Device global unicast address.
SL_WISUN_IP_ADDRESS_TYPE_BORDER_ROUTER	Border router global unicast address.
SL_WISUN_IP_ADDRESS_TYPE_PRIMARY_PARENT	Link-local address of the primary parent.
SL_WISUN_IP_ADDRESS_TYPE_SECONDARY_PARENT	Link-local address of the secondary parent.

Definition at line `84` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_certificate_option_t

> sl_wisun_certificate_option_t

Enumerations for certificate options.

Enumerator	
SL_WISUN_CERTIFICATE_OPTION_NONE	Empty option.
SL_WISUN_CERTIFICATE_OPTION_APPEND	Certificate is appended to a chain.
SL_WISUN_CERTIFICATE_OPTION_IS_REF	Certificate data will remain in scope.
SL_WISUN_CERTIFICATE_OPTION_HAS_KEY	Certificate has a private key.

Definition at line `98` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_private_key_option_t

> sl_wisun_private_key_option_t

Enumerations for private key options.

Enumerator	
SL_WISUN_PRIVATE_KEY_OPTION_NONE	Empty option.
SL_WISUN_PRIVATE_KEY_OPTION_IS_REF	Private key data will remain in scope.

Definition at line 110 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_type_t

> sl_wisun_statistics_type_t

Enumerations for statistics type.

Enumerator	
SL_WISUN_STATISTICS_TYPE_PHY	PHY/RF statistics.
SL_WISUN_STATISTICS_TYPE_MAC	MAC statistics.
SL_WISUN_STATISTICS_TYPE_FHSS	Frequency hopping statistics.
SL_WISUN_STATISTICS_TYPE_WISUN	Wi-SUN statistics.
SL_WISUN_STATISTICS_TYPE_NETWORK	6LoWPAN/IP stack statistics
SL_WISUN_STATISTICS_TYPE_REGULATION	Regional regulation.
SL_WISUN_STATISTICS_TYPE_HEAP	Heap usage.

Definition at line 118 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_regulatory_domain_t

> sl_wisun_regulatory_domain_t

Enumerations for regulatory domain.

Enumerator	
SL_WISUN_REGULATORY_DOMAIN_WW	World-wide (2.4 GHz)
SL_WISUN_REGULATORY_DOMAIN_NA	North America.
SL_WISUN_REGULATORY_DOMAIN_JP	Japan.
SL_WISUN_REGULATORY_DOMAIN_EU	Europe.
SL_WISUN_REGULATORY_DOMAIN_CN	China.
SL_WISUN_REGULATORY_DOMAIN_IN	India.
SL_WISUN_REGULATORY_DOMAIN_MX	Mexico.
SL_WISUN_REGULATORY_DOMAIN_BZ	Brazil.
SL_WISUN_REGULATORY_DOMAIN_AZ	Australia.
SL_WISUN_REGULATORY_DOMAIN_NZ	New Zealand.
SL_WISUN_REGULATORY_DOMAIN_KR	South Korea.
SL_WISUN_REGULATORY_DOMAIN_PH	Philippines.
SL_WISUN_REGULATORY_DOMAIN_MY	Malaysia.
SL_WISUN_REGULATORY_DOMAIN_HK	Hong Kong.
SL_WISUN_REGULATORY_DOMAIN_SG	Singapore.

SL_WISUN_REGULATORY_DOMAIN_TH	Thailand.
SL_WISUN_REGULATORY_DOMAIN_VN	Vietnam.
SL_WISUN_REGULATORY_DOMAIN_APP	Application-specific domain.

Definition at line 136 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_operating_class_t

> sl_wisun_operating_class_t

Enumerations for operating class.

Enumerator	
SL_WISUN_OPERATING_CLASS_1	Operating class# 1.
SL_WISUN_OPERATING_CLASS_2	Operating class# 2.
SL_WISUN_OPERATING_CLASS_3	Operating class# 3.
SL_WISUN_OPERATING_CLASS_4	Operating class# 4.
SL_WISUN_OPERATING_CLASS_5	Operating class# 5.
SL_WISUN_OPERATING_CLASS_APP	Application-specific class.

Definition at line 176 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_operating_mode_t

> sl_wisun_operating_mode_t

Enumerations for operating mode.

Enumerator	
SL_WISUN_OPERATING_MODE_1A	Operating mode# 1a.
SL_WISUN_OPERATING_MODE_1B	Operating mode# 1b.
SL_WISUN_OPERATING_MODE_2A	Operating mode# 2a.
SL_WISUN_OPERATING_MODE_2B	Operating mode# 2b.
SL_WISUN_OPERATING_MODE_3	Operating mode# 3.
SL_WISUN_OPERATING_MODE_4A	Operating mode# 4a.
SL_WISUN_OPERATING_MODE_4B	Operating mode# 4b.
SL_WISUN_OPERATING_MODE_5	Operating mode# 5.

Definition at line 192 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_multicast_group_action_t

> sl_wisun_multicast_group_action_t

Enumerations for multicast group action.

Enumerator	
SL_WISUN_MULTICAST_GROUP_ACTION_JOIN	Join a multicast group.
SL_WISUN_MULTICAST_GROUP_ACTION_LEAVE	Leave a multicast group.

Definition at line 212 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_channel_spacing_t

> sl_wisun_channel_spacing_t

Enumerations for channel spacing.

Enumerator	
SL_WISUN_CHANNEL_SPACING_100KHZ	100 kHz
SL_WISUN_CHANNEL_SPACING_200KHZ	200 kHz
SL_WISUN_CHANNEL_SPACING_400KHZ	400 kHz
SL_WISUN_CHANNEL_SPACING_600KHZ	600 kHz
SL_WISUN_CHANNEL_SPACING_250KHZ	250 kHz
SL_WISUN_CHANNEL_SPACING_800KHZ	800 kHz
SL_WISUN_CHANNEL_SPACING_1200KHZ	1200 kHz

Definition at line 220 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_join_state_t

> sl_wisun_join_state_t

Enumerations for join state.

Enumerator	
SL_WISUN_JOIN_STATE_DISCONNECTED	Join state 0: Disconnected.
SL_WISUN_JOIN_STATE_SELECT_PAN	Join state 1: Select PAN.
SL_WISUN_JOIN_STATE_AUTHENTICATE	Join state 2: Authenticate.
SL_WISUN_JOIN_STATE_ACQUIRE_PAN_CONFIG	Join state 3: Acquire PAN config.
SL_WISUN_JOIN_STATE_CONFIGURE_ROUTING	Join state 4: Configure routing.
SL_WISUN_JOIN_STATE_OPERATIONAL	Join state 5: Operational.
SL_WISUN_JOIN_STATE_PARENT_SELECT	Join state 4: Preferred parent selection.
SL_WISUN_JOIN_STATE_DHCP	Join state 4: DHCP address acquisition.
SL_WISUN_JOIN_STATE_EARO	Join state 4: Address registration.
SL_WISUN_JOIN_STATE_DAO	Join state 4: DAO registration.

Definition at line 250 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_network_update_flags_t

> sl_wisun_network_update_flags_t

Enumerations for network update flags.

Enumerator	
SL_WISUN_NETWORK_UPDATE_FLAGS_GLOBAL_IP	Global IP modification flag bit.
SL_WISUN_NETWORK_UPDATE_FLAGS_PRIMARY_PARENT	Primary Parent modification flag bit.
SL_WISUN_NETWORK_UPDATE_FLAGS_SECONDARY_PARENT	Secondary parent modification flag bit.

Definition at line 274 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_type_t

> sl_wisun_phy_config_type_t

Enumerations for PHY config type.

Enumerator	
SL_WISUN_PHY_CONFIG_FAN10	FAN1.0 configuration.
SL_WISUN_PHY_CONFIG_FAN11	FAN1.1 configuration.
SL_WISUN_PHY_CONFIG_EXPLICIT	Explicit configuration.
SL_WISUN_PHY_CONFIG_IDS	Specific RAIL channel configuration.
SL_WISUN_PHY_CONFIG_CUSTOM_FSK	FSK customization.
SL_WISUN_PHY_CONFIG_CUSTOM_OFDM	OFDM customization.
SL_WISUN_PHY_CONFIG_CUSTOM_OQPSK	OQPSK customization.

Definition at line 284 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_lfn_profile_t

> sl_wisun_lfn_profile_t

Enumeration for LFN configuration profile.

Enumerator	
SL_WISUN_LFN_PROFILE_TEST	Profile for test usage, best performance but highest power consumption.
SL_WISUN_LFN_PROFILE_BALANCED	Profile providing balance between power consumption and performance.
SL_WISUN_LFN_PROFILE_ECO	Profile optimized for low power consumption.

Definition at line 302 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_crc_type_t

> sl_wisun_crc_type_t

Enumeration for CRC type.

Enumerator	
SL_WISUN_NO_CRC	No CRC (OFDM and OQPSK only)
SL_WISUN_2_BYTES_CRC	2 bytes CRC
SL_WISUN_4_BYTES_CRC	4 bytes CRC

Definition at line 312 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_socket_protocol_t

> sl_wisun_socket_protocol_t

Enumerations for socket protocol Deprecated.

Enumerator	
SL_WISUN_SOCKET_PROTOCOL_UDP	User Datagram Protocol (UDP)
SL_WISUN_SOCKET_PROTOCOL_TCP	Transmission Control Protocol (TCP)
SL_WISUN_SOCKET_PROTOCOL_ICMP	Internet Control Message Protocol (ICMP)

Definition at line 652 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_socket_option_t

> sl_wisun_socket_option_t

Enumerations for socket option Deprecated.

Enumerator	
SL_WISUN_SOCKET_OPTION_EVENT_MODE	Option for socket event mode.
SL_WISUN_SOCKET_OPTION_MULTICAST_GROUP	Option for multicast group.
SL_WISUN_SOCKET_OPTION_SEND_BUFFER_LIMIT	Option for send buffer limit.
SL_WISUN_SOCKET_OPTION_EDFE_MODE	Option to enable/disable Extended Directed Frame Exchange mode.
SL_WISUN_SOCKET_OPTION_UNICAST_HOP_LIMIT	Option to set socket unicast hop limit.
SL_WISUN_SOCKET_OPTION_MULTICAST_HOP_LIMIT	Option to set socket multicast hop limit.

Definition at line 663 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_neighbor_type_t

> sl_wisun_neighbor_type_t

Enumeration for RPL neighbor types.

Enumerator	
SL_WISUN_NEIGHBOR_TYPE_PRIMARY_PARENT	Primary parent.
SL_WISUN_NEIGHBOR_TYPE_SECONDARY_PARENT	Secondary parent.
SL_WISUN_NEIGHBOR_TYPE_CHILD	Child.

Definition at line 757 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_trace_group_t

> sl_wisun_trace_group_t

Enumeration for trace group.

Enumerator	
SL_WISUN_TRACE_GROUP_MAC	Mac.
SL_WISUN_TRACE_GROUP_NW	Network.
SL_WISUN_TRACE_GROUP_LLC	LLC.
SL_WISUN_TRACE_GROUP_6LO	6lowpan

SL_WISUN_TRACE_GROUP_IPV6	IPV6.
SL_WISUN_TRACE_GROUP_TCP	TCP.
SL_WISUN_TRACE_GROUP_UDP	UDP.
SL_WISUN_TRACE_GROUP_ICMP	ICMP.
SL_WISUN_TRACE_GROUP_DHCP	DHCP.
SL_WISUN_TRACE_GROUP_MPL	MPL.
SL_WISUN_TRACE_GROUP_DNS	DNS.
SL_WISUN_TRACE_GROUP_RPL	RPL.
SL_WISUN_TRACE_GROUP_TRIC	Trickle.
SL_WISUN_TRACE_GROUP_WS	Wi-SUN Stack.
SL_WISUN_TRACE_GROUP_BOOT	Wi-SUN Bootstrap.
SL_WISUN_TRACE_GROUP_WSR	Wi-SUN Router.
SL_WISUN_TRACE_GROUP_WSBR	Border router.
SL_WISUN_TRACE_GROUP_SEC	Security.
SL_WISUN_TRACE_GROUP_TIME	Time and timers.
SL_WISUN_TRACE_GROUP_NEIGH	Neighbor.
SL_WISUN_TRACE_GROUP_STAT	Statistics.
SL_WISUN_TRACE_GROUP_BUFF	Dynamic Buffer.
SL_WISUN_TRACE_GROUP_ADDR	Address Manipulation.
SL_WISUN_TRACE_GROUP_MON	Monitoring.
SL_WISUN_TRACE_GROUP_SOCK	Socket.
SL_WISUN_TRACE_GROUP_DENY	Deny list.
SL_WISUN_TRACE_GROUP_ETX	ETX.
SL_WISUN_TRACE_GROUP_FHSS	FHSS.
SL_WISUN_TRACE_GROUP_ROUT	Routing table.
SL_WISUN_TRACE_GROUP_EVLP	Event loop.
SL_WISUN_TRACE_GROUP_NVM	NVM.
SL_WISUN_TRACE_GROUP_CRYPTO	Crypto.
SL_WISUN_TRACE_GROUP_RF	Wi-SUN RF Driver.
SI_WISUN_TRACE_GROUP_WSIE	Wi-SUN IE.
SL_WISUN_TRACE_GROUP_CONFIG	Configuration.
SL_WISUN_TRACE_GROUP_TIM_SRV	Timer service.
SL_WISUN_TRACE_GROUP_LFN_TIM	LFN timing measurement.
SL_WISUN_TRACE_GROUP_INT	Internal usage.
SL_WISUN_TRACE_GROUP_COUNT	Max number of trace group in this enum.

Definition at line `815` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_trace_level_t

```
sl_wisun_trace_level_t
```

Enumerations for trace level.

Enumerator	
SL_WISUN_TRACE_LEVEL_NONE	No trace.
SL_WISUN_TRACE_LEVEL_ERROR	Error only.
SL_WISUN_TRACE_LEVEL_WARN	Warning + error.
SL_WISUN_TRACE_LEVEL_INFO	Info + warning + error.
SL_WISUN_TRACE_LEVEL_DEBUG	Debug + info + warning + error.

Definition at line `868` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_regulation_t

> sl_wisun_regulation_t

Enumerations for regional regulation.

Enumerator	
SL_WISUN_REGULATION_NONE	No regulation.
SL_WISUN_REGULATION_ARIB	ARIB, can only be used with JP regulatory domain.

Definition at line `891` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_mode_switch_mode_t

> sl_wisun_mode_switch_mode_t

Enumeration for Mode Switch mode.

Enumerator	
SL_WISUN_MODE_SWITCH_DISABLED	Mode switch is not allowed.
SL_WISUN_MODE_SWITCH_ENABLED	Mode switch is allowed for all unicast data frames. Specified PhyModeId is used.
SL_WISUN_MODE_SWITCH_DEFAULT	Mode switch is allowed for all unicast data frames. Default PhyModeId is used.

Definition at line `899` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_regulation_tx_level_t

> sl_wisun_regulation_tx_level_t

Enumeration for regional regulation transmission level.

Thresholds are define with sl_wisun_set_regulation_tx_thresholds.

Enumerator	
SL_WISUN_REGULATION_TX_LEVEL_LOW	Transmission duration is compliant with regional regulation.
SL_WISUN_REGULATION_TX_LEVEL_WARNING	Transmission duration is above warning threshold.
SL_WISUN_REGULATION_TX_LEVEL_ALERT	Transmission duration is above alert threshold.

Definition at line `910` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_unicast_tx_mode_t

sl_wisun_unicast_tx_mode_t

Enumeration for unicast transmission mode.

Enumerator	
SL_WISUN_UNICAST_TX_MODE_DEFAULT	Default unicast transmission.
SL_WISUN_UNICAST_TX_MODE_SLOT	Allow unicast transmission only on a slot.

Definition at line 920 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_channel_exclusion_mode_t

sl_wisun_channel_exclusion_mode_t

Enumeration for channel exlusion modes.

Enumerator	
SL_WISUN_CHANNEL_EXCLUSION_MODE_BY_RANGE	Channels are excluded by range if possible (3 ranges maximum), otherwise channels will be excluded by mask.
SL_WISUN_CHANNEL_EXCLUSION_MODE_BY_MASK	Channels are excluded by mask.

Definition at line 931 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_frame_type_t

sl_wisun_frame_type_t

Enumeration for types of frame that can be triggered.

Enumerator	
SL_WISUN_FRAME_TYPE_PAS	
SL_WISUN_FRAME_TYPE_PA	
SL_WISUN_FRAME_TYPE_PCS	
SL_WISUN_FRAME_TYPE_PC	
SL_WISUN_FRAME_TYPE_DIS	
SL_WISUN_FRAME_TYPE_DIO	

Definition at line 940 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_channel_mask_type_t

sl_wisun_channel_mask_type_t

Enumeration for channel mask types.

Enumerator	
SL_WISUN_CHANNEL_MASK_TYPE_REGIONAL	Regional excluded channel mask (not advertised)
SL_WISUN_CHANNEL_MASK_TYPE_ADVERTISED_UNICAST	Excluded channel mask advertised in us-ie.
SL_WISUN_CHANNEL_MASK_TYPE_ADVERTISED_BROADCAST	Excluded channel mask advertised in bs-ie.

SL_WISUN_CHANNEL_MASK_TYPE_EFFECTIVE_ASYNC	Excluded channel mask applied to async frames.
SL_WISUN_CHANNEL_MASK_TYPE_EFFECTIVE_UNICAST	Excluded channel mask applied to unicast frequency hopping.
SL_WISUN_CHANNEL_MASK_TYPE_EFFECTIVE_BROADCAST	Excluded channel mask applied to broadcast frequency hopping.

Definition at line 1007 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

Typedef Documentation

sl_wisun_ip_address_t

```
typedef in6_addr_t sl_wisun_ip_address_t
```

Definition at line 685 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

Variable Documentation

sl_wisun_broadcast_mac

```
const sl_wisun_mac_address_t sl_wisun_broadcast_mac
```

Broadcast MAC address.

Definition at line 928 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

Macro Definition Documentation

SL_WISUN_NETWORK_NAME_SIZE

```
#define SL_WISUN_NETWORK_NAME_SIZE
```

Value:

```
32
```

Maximum size of the Wi-SUN network name.

Definition at line 45 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_MAC_ADDRESS_SIZE

```
#define SL_WISUN_MAC_ADDRESS_SIZE
```

Value:

```
8
```

Size of a MAC address.

Definition at line 47 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_CHANNEL_MASK_SIZE

```
#define SL_WISUN_CHANNEL_MASK_SIZE
```

Value:

```
32
```

Size of a channel mask.

Definition at line 49 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_FILTER_BITFIELD_SIZE

```
#define SL_WISUN_FILTER_BITFIELD_SIZE
```

Value:

```
((SL_WISUN_TRACE_GROUP_COUNT + 7) / 8)
```

Size of the filter bitfield.

Definition at line 51 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_ADVERT_FRAGMENT_DISABLE

```
#define SL_WISUN_ADVERT_FRAGMENT_DISABLE
```

Value:

```
UINT32_MAX
```

Maximum fragment duration. Disables advert fragmentation.

Definition at line 53 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_MAX_PHY_MODE_ID_COUNT

```
#define SL_WISUN_MAX_PHY_MODE_ID_COUNT
```

Value:

```
15
```

Maximum number of PhyModeId allowed in POM-IE.

Definition at line 55 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_CHANNEL_SPACING_100HZ

```
#define SL_WISUN_CHANNEL_SPACING_100HZ
```

Value:

```
SL_WISUN_CHANNEL_SPACING_100KHZ
```

Channel spacing 100 kHz for backward compatibility.

Definition at line 238 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_CHANNEL_SPACING_200HZ

```
#define SL_WISUN_CHANNEL_SPACING_200HZ
```

Value:

```
SL_WISUN_CHANNEL_SPACING_200KHZ
```

Channel spacing 200 kHz for backward compatibility.

Definition at line 241 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_CHANNEL_SPACING_400HZ

```
#define SL_WISUN_CHANNEL_SPACING_400HZ
```

Value:

```
SL_WISUN_CHANNEL_SPACING_400KHZ
```

Channel spacing 400 kHz for backward compatibility.

Definition at line 244 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_CHANNEL_SPACING_600HZ

```
#define SL_WISUN_CHANNEL_SPACING_600HZ
```

Value:

```
SL_WISUN_CHANNEL_SPACING_600KHZ
```

Channel spacing 600 kHz for backward compatibility.

Definition at line 247 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_TRACE_THREAD_WISUN

```
#define SL_WISUN_TRACE_THREAD_WISUN
```

Value:

```
"WS"
```

Thread identifier "Wi-SUN Task".

Definition at line 859 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_TRACE_THREAD_EVENT_TASK

```
#define SL_WISUN_TRACE_THREAD_EVENT_TASK
```

Value:

```
"EVT"
```

Thread identifier "Wi-SUN Event Task".

Definition at line 861 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_TRACE_THREAD_EVENT_LOOP

```
#define SL_WISUN_TRACE_THREAD_EVENT_LOOP
```

Value:

```
"EVL"
```

Thread identifier "Wi-SUN Event Loop Task".

Definition at line 863 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

SL_WISUN_TRACE_THREAD_MAC

```
#define SL_WISUN_TRACE_THREAD_MAC
```

Value:

```
"MAC"
```

Thread identifier "Wi-SUN RF Task".

Definition at line 865 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_msg_header_t

Wi-SUN Message API common header.

Public Attributes

uint16_t	length Total length of the message in bytes, this field included.
uint8_t	id ID (request, confirmation, indication) of the message.
uint8_t	info Processing metadata for the message.

Public Attribute Documentation

length

```
uint16_t sl_wisun_msg_header_t::length
```

Total length of the message in bytes, this field included.

Definition at line 325 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

id

```
uint8_t sl_wisun_msg_header_t::id
```

ID (request, confirmation, indication) of the message.

Definition at line 327 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

info

```
uint8_t sl_wisun_msg_header_t::info
```

Processing metadata for the message.

Definition at line 329 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_phy_t

PHY/RF statistics.

Public Attributes

uint32_t	crc_fails Number of CRC failures on reception.
uint32_t	tx_timeouts Number of transmission timeouts.
uint32_t	rx_timeouts Number of reception timeouts.

Public Attribute Documentation

crc_fails

```
uint32_t sl_wisun_statistics_phy_t::crc_fails
```

Number of CRC failures on reception.

Definition at line `336` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

tx_timeouts

```
uint32_t sl_wisun_statistics_phy_t::tx_timeouts
```

Number of transmission timeouts.

Definition at line `338` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

rx_timeouts

```
uint32_t sl_wisun_statistics_phy_t::rx_timeouts
```

Number of reception timeouts.

Definition at line `340` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_statistics_mac_t

MAC statistics.

Public Attributes

uint16_t	tx_queue_size Current number of frames in the MAC transmission queue.
uint16_t	tx_queue_peak Highest number of frames in the MAC transmission queue.
uint32_t	rx_count Number of successfully received MAC frames.
uint32_t	tx_count Number of transmitted MAC frames.
uint32_t	bc_rx_count Number of successfully received broadcast MAC frames.
uint32_t	bc_tx_count Number of transmitted broadcast MAC frames.
uint32_t	rx_drop_count Number of successfully received MAC frames discarded during processing.
uint32_t	tx_bytes Amount of transmitted MAC data in bytes. FCS is not included.
uint32_t	rx_bytes Amount of successfully received MAC data in bytes. FCS is not included.
uint32_t	tx_failed_count Number of failed MAC transmissions.
uint32_t	retry_count Number of retried MAC transmissions.
uint32_t	cca_attempts_count Number of MAC CCA attempts.
uint32_t	failed_cca_count Number of failed MAC transmissions due to CCA.
uint32_t	rx_ms_count Number of successfully received MAC frames using mode_switch.
uint32_t	tx_ms_count Number of transmitted MAC frames using mode switch.
uint32_t	rx_ms_failed_count Number of failed reception using mode switch (no data after PHR or MDR Command).
uint32_t	tx_ms_failed_count Number of failed MAC frames transmission using mode switch.

Public Attribute Documentation

tx_queue_size

```
uint16_t sl_wisun_statistics_mac_t::tx_queue_size
```

Current number of frames in the MAC transmission queue.

Definition at line 346 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

tx_queue_peak

```
uint16_t sl_wisun_statistics_mac_t::tx_queue_peak
```

Highest number of frames in the MAC transmission queue.

Definition at line 348 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rx_count

```
uint32_t sl_wisun_statistics_mac_t::rx_count
```

Number of successfully received MAC frames.

Definition at line 350 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

tx_count

```
uint32_t sl_wisun_statistics_mac_t::tx_count
```

Number of transmitted MAC frames.

Definition at line 352 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

bc_rx_count

```
uint32_t sl_wisun_statistics_mac_t::bc_rx_count
```

Number of successfully received broadcast MAC frames.

Definition at line 354 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

bc_tx_count

```
uint32_t sl_wisun_statistics_mac_t::bc_tx_count
```

Number of transmitted broadcast MAC frames.

Definition at line 356 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rx_drop_count

```
uint32_t sl_wisun_statistics_mac_t::rx_drop_count
```

Number of successfully received MAC frames discarded during processing.

Definition at line 358 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

tx_bytes

```
uint32_t sl_wisun_statistics_mac_t::tx_bytes
```

Amount of transmitted MAC data in bytes. FCS is not included.

Definition at line 360 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rx_bytes

```
uint32_t sl_wisun_statistics_mac_t::rx_bytes
```

Amount of successfully received MAC data in bytes. FCS is not included.

Definition at line 362 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

tx_failed_count

```
uint32_t sl_wisun_statistics_mac_t::tx_failed_count
```

Number of failed MAC transmissions.

Definition at line 364 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

retry_count

```
uint32_t sl_wisun_statistics_mac_t::retry_count
```

Number of retried MAC transmissions.

Definition at line 366 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

cca_attempts_count

```
uint32_t sl_wisun_statistics_mac_t::cca_attempts_count
```

Number of MAC CCA attempts.

Definition at line 368 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

failed_cca_count

```
uint32_t sl_wisun_statistics_mac_t::failed_cca_count
```

Number of failed MAC transmissions due to CCA.

Definition at line 370 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rx_ms_count

```
uint32_t sl_wisun_statistics_mac_t::rx_ms_count
```

Number of successfully received MAC frames using mode_switch.

Definition at line 372 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

tx_ms_count

```
uint32_t sl_wisun_statistics_mac_t::tx_ms_count
```

Number of transmitted MAC frames using mode switch.

Definition at line 374 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rx_ms_failed_count

```
uint32_t sl_wisun_statistics_mac_t::rx_ms_failed_count
```

Number of failed reception using mode switch (no data after PHR or MDR Command).

Definition at line 376 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

tx_ms_failed_count

```
uint32_t sl_wisun_statistics_mac_t::tx_ms_failed_count
```

Number of failed MAC frames transmission using mode switch.

Definition at line 378 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_fhss_t

Frequency hopping statistics.

Public Attributes

int16_t	drift_compensation Estimated clock drift to the parent in microseconds.
uint16_t	hop_count Estimated number of hops to the border router based on RPL rank.
uint16_t	synch_interval Number of seconds since last timing information from the parent.
int16_t	prev_avg_synch_fix Deprecated.
uint32_t	synch_lost Deprecated.
uint32_t	unknown_neighbor Number of times a transmission attempt has failed due to lack of timing information.

Public Attribute Documentation

drift_compensation

```
int16_t sl_wisun_statistics_fhss_t::drift_compensation
```

Estimated clock drift to the parent in microseconds.

Definition at line `384` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

hop_count

```
uint16_t sl_wisun_statistics_fhss_t::hop_count
```

Estimated number of hops to the border router based on RPL rank.

Definition at line `386` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

synch_interval

```
uint16_t sl_wisun_statistics_fhss_t::synch_interval
```

Number of seconds since last timing information from the parent.

Definition at line `388` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

prev_avg_synch_fix

> int16_t sl_wisun_statistics_fhss_t::prev_avg_synch_fix

Deprecated.

Definition at line 390 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

synch_lost

> uint32_t sl_wisun_statistics_fhss_t::synch_lost

Deprecated.

Definition at line 392 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

unknown_neighbor

> uint32_t sl_wisun_statistics_fhss_t::unknown_neighbor

Number of times a transmission attempt has failed due to lack of timing information.

Definition at line 394 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_wisun_t

Wi-SUN statistics.

Public Attributes

uint32_t	pan_control_rx_count Number of received PAN control frames.
uint32_t	pan_control_tx_count Number of completed PAN control transmission requests.

Public Attribute Documentation

pan_control_rx_count

```
uint32_t sl_wisun_statistics_wisun_t::pan_control_rx_count
```

Number of received PAN control frames.

Definition at line 400 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

pan_control_tx_count

```
uint32_t sl_wisun_statistics_wisun_t::pan_control_tx_count
```

Number of completed PAN control transmission requests.

Definition at line 402 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_network_t

6LoWPAN/IP stack statistics

Public Attributes

uint32_t	ip_rx_count Number of received IP6 packets.
uint32_t	ip_tx_count Number of transmitted IPv6 packets.
uint32_t	ip_rx_drop Number of discarded IPv6 packets during processing.
uint32_t	ip_cksum_error Number of discarded IPv6 packets due to a checksum error.
uint32_t	ip_tx_bytes Amount of transmitted IPv6 data in bytes.
uint32_t	ip_rx_bytes Amount received IPv6 data in bytes.
uint32_t	ip_routed_up Amount of forwarded IPv6 data in bytes.
uint32_t	ip_no_route Number of discarded IPv6 packets due to lack routing information.
uint32_t	frag_rx_errors Number of fragmentation errors in received IPv6 packets.
uint32_t	frag_tx_errors Number of fragmentation errors in transmitted IPv6 packets.
uint32_t	rpl_route_routecost_better_change Number of RPL parent changes due to better route cost.
uint32_t	ip_routeloop_detect Number of RPL packet forwarding errors due to inconsistent routing information.
uint32_t	rpl_memory_overflow Sum of RPL object sizes that have failed allocation in bytes.
uint32_t	rpl_parent_tx_fail Number of failed RPL transmissions to the parent.
uint32_t	rpl_unknown_instance Number of discarded RPL packets due to an unknown DODAG instance.
uint32_t	rpl_local_repair Number of times a local repair procedure has been triggered by the node.
uint32_t	rpl_global_repair Number of times a global repair has been triggered by the border router.

uint32_t	rpl_malformed_message	Number of discarded RPL packets due to malformed content.
uint32_t	rpl_time_no_next_hop	Number of seconds without an RPL parent.
uint32_t	rpl_total_memory	Amount of memory currently allocated for RPL objects in bytes.
uint32_t	buf_alloc	Number of data buffer allocation attempts.
uint32_t	buf_headroom_realloc	Number of times data buffers have been resized due to lack of header space.
uint32_t	buf_headroom_shuffle	Number of times data buffers have been reorganized due to lack of header space.
uint32_t	buf_headroom_fail	Number of times data buffer resizing has failed.
uint16_t	etx_1st_parent	ETX of the primary parent.
uint16_t	etx_2nd_parent	ETX of the secondary parent.
uint16_t	adapt_layer_tx_queue_size	Current number of frames in the adaptation layer transmission queue.
uint16_t	adapt_layer_tx_queue_peak	Highest number of frames in the adaptation layer transmission queue.

Public Attribute Documentation

ip_rx_count

```
uint32_t sl_wisun_statistics_network_t::ip_rx_count
```

Number of received IP6 packets.

Definition at line 408 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_tx_count

```
uint32_t sl_wisun_statistics_network_t::ip_tx_count
```

Number of transmitted IPv6 packets.

Definition at line 410 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_rx_drop

```
uint32_t sl_wisun_statistics_network_t::ip_rx_drop
```

Number of discarded IPv6 packets during processing.

Definition at line 412 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_cksum_error

```
uint32_t sl_wisun_statistics_network_t::ip_cksum_error
```

Number of discarded IPv6 packets due to a checksum error.

Definition at line 414 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_tx_bytes

```
uint32_t sl_wisun_statistics_network_t::ip_tx_bytes
```

Amount of transmitted IPv6 data in bytes.

Definition at line 416 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_rx_bytes

```
uint32_t sl_wisun_statistics_network_t::ip_rx_bytes
```

Amount received IPv6 data in bytes.

Definition at line 418 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_routed_up

```
uint32_t sl_wisun_statistics_network_t::ip_routed_up
```

Amount of forwarded IPv6 data in bytes.

Definition at line 420 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_no_route

```
uint32_t sl_wisun_statistics_network_t::ip_no_route
```

Number of discarded IPv6 packets due to lack routing information.

Definition at line 422 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

frag_rx_errors

```
uint32_t sl_wisun_statistics_network_t::frag_rx_errors
```

Number of fragmentation errors in received IPv6 packets.

Definition at line 424 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

frag_tx_errors

```
uint32_t sl_wisun_statistics_network_t::frag_tx_errors
```

Number of fragmentation errors in transmitted IPv6 packets.

Definition at line 426 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_route_routecost_better_change

```
uint32_t sl_wisun_statistics_network_t::rpl_route_routecost_better_change
```

Number of RPL parent changes due to better route cost.

Definition at line 428 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ip_routeloop_detect

```
uint32_t sl_wisun_statistics_network_t::ip_routeloop_detect
```

Number of RPL packet forwarding errors due to inconsistent routing information.

Definition at line 430 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_memory_overflow

```
uint32_t sl_wisun_statistics_network_t::rpl_memory_overflow
```

Sum of RPL object sizes that have failed allocation in bytes.

Definition at line 432 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_parent_tx_fail

```
uint32_t sl_wisun_statistics_network_t::rpl_parent_tx_fail
```

Number of failed RPL transmissions to the parent.

Definition at line 434 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_unknown_instance

```
uint32_t sl_wisun_statistics_network_t::rpl_unknown_instance
```

Number of discarded RPL packets due to an unknown DODAG instance.

Definition at line 436 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_local_repair

```
uint32_t sl_wisun_statistics_network_t::rpl_local_repair
```

Number of times a local repair procedure has been triggered by the node.

Definition at line 438 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_global_repair

```
uint32_t sl_wisun_statistics_network_t::rpl_global_repair
```

Number of times a global repair has been triggered by the border router.

Definition at line 440 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_malformed_message

```
uint32_t sl_wisun_statistics_network_t::rpl_malformed_message
```

Number of discarded RPL packets due to malformed content.

Definition at line 442 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_time_no_next_hop

```
uint32_t sl_wisun_statistics_network_t::rpl_time_no_next_hop
```

Number of seconds without an RPL parent.

Definition at line 444 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_total_memory

```
uint32_t sl_wisun_statistics_network_t::rpl_total_memory
```

Amount of memory currently allocated for RPL objects in bytes.

Definition at line 446 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

buf_alloc

```
uint32_t sl_wisun_statistics_network_t::buf_alloc
```

Number of data buffer allocation attempts.

Definition at line 448 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

buf_headroom_realloc

```
uint32_t sl_wisun_statistics_network_t::buf_headroom_realloc
```

Number of times data buffers have been resized due to lack of header space.

Definition at line 450 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

buf_headroom_shuffle

```
uint32_t sl_wisun_statistics_network_t::buf_headroom_shuffle
```

Number of times data buffers have been reorganized due to lack of header space.

Definition at line 452 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

buf_headroom_fail

```
uint32_t sl_wisun_statistics_network_t::buf_headroom_fail
```

Number of times data buffer resizing has failed.

Definition at line 454 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

etx_1st_parent

```
uint16_t sl_wisun_statistics_network_t::etx_1st_parent
```

ETX of the primary parent.

Definition at line 456 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

etx_2nd_parent

```
uint16_t sl_wisun_statistics_network_t::etx_2nd_parent
```

ETX of the secondary parent.

Definition at line 458 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

adapt_layer_tx_queue_size

```
uint16_t sl_wisun_statistics_network_t::adapt_layer_tx_queue_size
```

Current number of frames in the adaptation layer transmission queue.

Definition at line 460 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

adapt_layer_tx_queue_peak

> uint16_t sl_wisun_statistics_network_t::adapt_layer_tx_queue_peak

Highest number of frames in the adaptation layer transmission queue.

Definition at line 462 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_arib_regulation_t

ARIB regulation statistics.

Public Attributes

uint32_t	tx_duration_ms Sum of transmission durations during the last hour in milliseconds.

Public Attribute Documentation

tx_duration_ms

> uint32_t sl_wisun_statistics_arib_regulation_t::tx_duration_ms

Sum of transmission durations during the last hour in milliseconds.

Definition at line 467 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_regulation_t

Regional regulation statistics.

Public Attributes

sl_wisun_statistics_arib_regulation_t	arib ARIB statistics.

Public Attribute Documentation

arib

> sl_wisun_statistics_arib_regulation_t sl_wisun_statistics_regulation_t::arib

ARIB statistics.

Definition at line 473 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_heap_t

Heap usage statistics.

Public Attributes

uint32_t	arena Heap arena.
uint32_t	uordblks Current heap usage.

Public Attribute Documentation

arena

```
uint32_t sl_wisun_statistics_heap_t::arena
```

Heap arena.

Definition at line 479 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

uordblks

```
uint32_t sl_wisun_statistics_heap_t::uordblks
```

Current heap usage.

Definition at line 481 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_statistics_t

Statistics.

Public Attributes

sl_wisun_statistics_phy_t	phy PHY/RF statistics.
sl_wisun_statistics_mac_t	mac MAC statistics.
sl_wisun_statistics_fhss_t	fhss Frequency hopping statistics.
sl_wisun_statistics_wisun_t	wisun Wi-SUN statistics.
sl_wisun_statistics_network_t	network 6LoWPAN/IP stack statistics
sl_wisun_statistics_regulation_t	regulation Regional regulation statistics.
sl_wisun_statistics_heap_t	heap Heap usage statistics.

Public Attribute Documentation

phy

```
sl_wisun_statistics_phy_t sl_wisun_statistics_t::phy
```

PHY/RF statistics.

Definition at line 487 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

mac

```
sl_wisun_statistics_mac_t sl_wisun_statistics_t::mac
```

MAC statistics.

Definition at line 489 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

fhss

```
sl_wisun_statistics_fhss_t sl_wisun_statistics_t::fhss
```

Frequency hopping statistics.

Definition at line 491 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

wisun

```
sl_wisun_statistics_wisun_t sl_wisun_statistics_t::wisun
```

Wi-SUN statistics.

Definition at line 493 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

network

```
sl_wisun_statistics_network_t sl_wisun_statistics_t::network
```

6LoWPAN/IP stack statistics

Definition at line 495 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

regulation

```
sl_wisun_statistics_regulation_t sl_wisun_statistics_t::regulation
```

Regional regulation statistics.

Definition at line 497 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

heap

```
sl_wisun_statistics_heap_t sl_wisun_statistics_t::heap
```

Heap usage statistics.

Definition at line 499 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_fan10_t

FAN1.0 PHY configuration.

Public Attributes

uint8_t	reg_domain Regulatory domain (sl_wisun_regulatory_domain_t)
uint8_t	op_class Operating class (sl_wisun_operating_class_t)
uint8_t	op_mode Operating mode (sl_wisun_operating_mode_t)
uint8_t	fec 1 if FEC is enabled, 0 if not

Public Attribute Documentation

reg_domain

```
uint8_t sl_wisun_phy_config_fan10_t::reg_domain
```

Regulatory domain (sl_wisun_regulatory_domain_t)

Definition at line 505 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

op_class

```
uint8_t sl_wisun_phy_config_fan10_t::op_class
```

Operating class (sl_wisun_operating_class_t)

Definition at line 507 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

op_mode

```
uint8_t sl_wisun_phy_config_fan10_t::op_mode
```

Operating mode (sl_wisun_operating_mode_t)

Definition at line 509 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

fec

uint8_t sl_wisun_phy_config_fan10_t::fec

1 if FEC is enabled, 0 if not

Definition at line 511 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_fan11_t

FAN1.1 PHY configuration.

Public Attributes

uint8_t	reg_domain Regulatory domain (sl_wisun_regulatory_domain_t)
uint8_t	chan_plan_id Channel plan ID.
uint8_t	phy_mode_id PHY mode ID.

Public Attribute Documentation

reg_domain

```
uint8_t sl_wisun_phy_config_fan11_t::reg_domain
```

Regulatory domain (sl_wisun_regulatory_domain_t)

Definition at line 517 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

chan_plan_id

```
uint8_t sl_wisun_phy_config_fan11_t::chan_plan_id
```

Channel plan ID.

Definition at line 519 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

phy_mode_id

```
uint8_t sl_wisun_phy_config_fan11_t::phy_mode_id
```

PHY mode ID.

Definition at line 521 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_explicit_t

Explicit PHY configuration.

Public Attributes

uint32_t	ch0_frequency_khz Ch0 center frequency in kHz.
uint16_t	number_of_channels Number of channels.
uint8_t	channel_spacing Channel spacing (sl_wisun_channel_spacing_t)
uint8_t	phy_mode_id PHY mode ID.

Public Attribute Documentation

ch0_frequency_khz

```
uint32_t sl_wisun_phy_config_explicit_t::ch0_frequency_khz
```

Ch0 center frequency in kHz.

Definition at line 527 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

number_of_channels

```
uint16_t sl_wisun_phy_config_explicit_t::number_of_channels
```

Number of channels.

Definition at line 529 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

channel_spacing

```
uint8_t sl_wisun_phy_config_explicit_t::channel_spacing
```

Channel spacing (sl_wisun_channel_spacing_t)

Definition at line 531 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

phy_mode_id

uint8_t sl_wisun_phy_config_explicit_t::phy_mode_id

PHY mode ID.

Definition at line 533 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_ids_t

Explicit RAIL configuration.

Public Attributes

uint16_t	protocol_id Protocol ID.
uint16_t	channel_id Channel ID.
uint8_t	phy_mode_id PHY mode ID.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

protocol_id

```
uint16_t sl_wisun_phy_config_ids_t::protocol_id
```

Protocol ID.

Definition at line 539 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

channel_id

```
uint16_t sl_wisun_phy_config_ids_t::channel_id
```

Channel ID.

Definition at line 541 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

phy_mode_id

```
uint8_t sl_wisun_phy_config_ids_t::phy_mode_id
```

PHY mode ID.

Definition at line 543 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

reserved

uint8_t sl_wisun_phy_config_ids_t::reserved[3]

Reserved, set to zero.

Definition at line 545 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_custom_fsk_t

Custom FSK PHY configuration.

Public Attributes

uint32_t	ch0_frequency_khz Ch0 center frequency in kHz.
uint16_t	channel_spacing_khz Channel spacing in kHz.
uint16_t	number_of_channels Number of channels.
uint8_t	phy_mode_id PHY mode ID.
uint8_t	crc_type FSK CRC type (sl_wisun_crc_type_t)
uint8_t	preamble_length FSK preamble length in bits.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

ch0_frequency_khz

> uint32_t sl_wisun_phy_config_custom_fsk_t::ch0_frequency_khz

Ch0 center frequency in kHz.

Definition at line 551 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

channel_spacing_khz

> uint16_t sl_wisun_phy_config_custom_fsk_t::channel_spacing_khz

Channel spacing in kHz.

Definition at line 553 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

number_of_channels

> uint16_t sl_wisun_phy_config_custom_fsk_t::number_of_channels

Number of channels.

Definition at line 555 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

phy_mode_id

```
uint8_t sl_wisun_phy_config_custom_fsk_t::phy_mode_id
```

PHY mode ID.

Definition at line 557 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

crc_type

```
uint8_t sl_wisun_phy_config_custom_fsk_t::crc_type
```

FSK CRC type (sl_wisun_crc_type_t)

Definition at line 559 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

preamble_length

```
uint8_t sl_wisun_phy_config_custom_fsk_t::preamble_length
```

FSK preamble length in bits.

Definition at line 561 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

reserved

```
uint8_t sl_wisun_phy_config_custom_fsk_t::reserved[1]
```

Reserved, set to zero.

Definition at line 563 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_custom_ofdm_t

Custom OFDM PHY configuration.

Public Attributes

uint32_t	ch0_frequency_khz Ch0 center frequency in kHz.
uint16_t	channel_spacing_khz Channel spacing in kHz.
uint16_t	number_of_channels Number of channels.
uint8_t	phy_mode_id PHY mode ID.
uint8_t	crc_type OFDM CRC type (sl_wisun_crc_type_t)
uint8_t	stf_length STF length in number of symbols.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

ch0_frequency_khz

```
uint32_t sl_wisun_phy_config_custom_ofdm_t::ch0_frequency_khz
```

Ch0 center frequency in kHz.

Definition at line 569 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

channel_spacing_khz

```
uint16_t sl_wisun_phy_config_custom_ofdm_t::channel_spacing_khz
```

Channel spacing in kHz.

Definition at line 571 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

number_of_channels

```
uint16_t sl_wisun_phy_config_custom_ofdm_t::number_of_channels
```

Number of channels.

Definition at line 573 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

phy_mode_id

```
uint8_t sl_wisun_phy_config_custom_ofdm_t::phy_mode_id
```

PHY mode ID.

Definition at line 575 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

crc_type

```
uint8_t sl_wisun_phy_config_custom_ofdm_t::crc_type
```

OFDM CRC type (sl_wisun_crc_type_t)

Definition at line 577 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

stf_length

```
uint8_t sl_wisun_phy_config_custom_ofdm_t::stf_length
```

STF length in number of symbols.

Definition at line 579 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

reserved

```
uint8_t sl_wisun_phy_config_custom_ofdm_t::reserved[1]
```

Reserved, set to zero.

Definition at line 581 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_custom_oqpsk_t

Custom QPSK PHY configuration.

Public Attributes

uint32_t	ch0_frequency_khz Ch0 center frequency in kHz.
uint16_t	channel_spacing_khz Channel spacing in kHz.
uint16_t	number_of_channels Number of channels.
uint8_t	phy_mode_id PHY mode ID.
uint8_t	crc_type OFDM CRC type (sl_wisun_crc_type_t)
uint8_t	preamble_length OQPSK preamble length in bits.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

ch0_frequency_khz

```
uint32_t sl_wisun_phy_config_custom_oqpsk_t::ch0_frequency_khz
```

Ch0 center frequency in kHz.

Definition at line 587 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

channel_spacing_khz

```
uint16_t sl_wisun_phy_config_custom_oqpsk_t::channel_spacing_khz
```

Channel spacing in kHz.

Definition at line 589 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

number_of_channels

```
uint16_t sl_wisun_phy_config_custom_oqpsk_t::number_of_channels
```

Number of channels.

Definition at line 591 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

phy_mode_id

> uint8_t sl_wisun_phy_config_custom_oqpsk_t::phy_mode_id

PHY mode ID.

Definition at line 593 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

crc_type

> uint8_t sl_wisun_phy_config_custom_oqpsk_t::crc_type

OFDM CRC type (sl_wisun_crc_type_t)

Definition at line 595 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

preamble_length

> uint8_t sl_wisun_phy_config_custom_oqpsk_t::preamble_length

OQPSK preamble length in bits.

Definition at line 597 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

reserved

> uint8_t sl_wisun_phy_config_custom_oqpsk_t::reserved[1]

Reserved, set to zero.

Definition at line 599 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_phy_config_t

PHY configuration.

Public Attributes

uint32_t	type Configuration type (sl_wisun_phy_config_type_t)
sl_wisun_phy_config_fan10_t	fan10 Configuration for SL_WISUN_PHY_CONFIG_FAN10 type.
sl_wisun_phy_config_fan11_t	fan11 Configuration for SL_WISUN_PHY_CONFIG_FAN11 type.
sl_wisun_phy_config_explicit_t	explicit_plan Configuration for SL_WISUN_PHY_CONFIG_EXPLICIT type.
sl_wisun_phy_config_ids_t	ids Configuration for SL_WISUN_PHY_CONFIG_IDS type.
sl_wisun_phy_config_custom_fsk_t	custom_fsk Configuration for SL_WISUN_PHY_CONFIG_CUSTOM_FSK type.
sl_wisun_phy_config_custom_ofdm_t	custom_ofdm Configuration for SL_WISUN_PHY_CONFIG_CUSTOM_OFDM type.
sl_wisun_phy_config_custom_oqpsk_t	custom_oqpsk Configuration for SL_WISUN_PHY_CONFIG_CUSTOM_OQPSK type.
union sl_wisun_phy_config_t::@1	config Configuration.

Public Attribute Documentation

type

```
uint32_t sl_wisun_phy_config_t::type
```

Configuration type (sl_wisun_phy_config_type_t)

Definition at line 606 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

fan10

```
sl_wisun_phy_config_fan10_t sl_wisun_phy_config_t::fan10
```

Configuration for SL_WISUN_PHY_CONFIG_FAN10 type.

Definition at line 610 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

fan11

> sl_wisun_phy_config_fan11_t sl_wisun_phy_config_t::fan11

Configuration for SL_WISUN_PHY_CONFIG_FAN11 type.

Definition at line 612 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

explicit_plan

> sl_wisun_phy_config_explicit_t sl_wisun_phy_config_t::explicit_plan

Configuration for SL_WISUN_PHY_CONFIG_EXPLICIT type.

Definition at line 614 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ids

> sl_wisun_phy_config_ids_t sl_wisun_phy_config_t::ids

Configuration for SL_WISUN_PHY_CONFIG_IDS type.

Definition at line 616 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

custom_fsk

> sl_wisun_phy_config_custom_fsk_t sl_wisun_phy_config_t::custom_fsk

Configuration for SL_WISUN_PHY_CONFIG_CUSTOM_FSK type.

Definition at line 618 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

custom_ofdm

> sl_wisun_phy_config_custom_ofdm_t sl_wisun_phy_config_t::custom_ofdm

Configuration for SL_WISUN_PHY_CONFIG_CUSTOM_OFDM type.

Definition at line 620 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

custom_oqpsk

> sl_wisun_phy_config_custom_oqpsk_t sl_wisun_phy_config_t::custom_oqpsk

Configuration for SL_WISUN_PHY_CONFIG_CUSTOM_OQPSK type.

Definition at line 622 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

config

```
union sl_wisun_phy_config_t::@1 sl_wisun_phy_config_t::config
```

Configuration.

Definition at line 623 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_mac_address_t

MAC address.

Public Attributes

uint8_t	address MAC address.

Public Attribute Documentation

address

```
uint8_t sl_wisun_mac_address_t::address[SL_WISUN_MAC_ADDRESS_SIZE]
```

MAC address.

This field contains a MAC address (EUI-64) stored in canonical format where the first byte of the array is the most-significant byte of the MAC address.

Definition at line `634` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_channel_mask_t

Channel mask.

Public Attributes

uint8_t	mask Bit mask of channels.

Public Attribute Documentation

mask

```
uint8_t sl_wisun_channel_mask_t::mask[SL_WISUN_CHANNEL_MASK_SIZE]
```

Bit mask of channels.

This field specifies a bit mask of channels, one bit per channel. First byte of the array represents channel numbers 0 - 7, with bit 0 being channel 0. Second byte represents channel numbers 8 - 15 and so forth.

Definition at line `646` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_socket_option_event_mode_t

Socket option for event mode Deprecated.

Public Attributes

uint32_t	mode Socket event mode.

Public Attribute Documentation

mode

```
uint32_t sl_wisun_socket_option_event_mode_t::mode
```

Socket event mode.

Definition at line 682 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_socket_option_multicast_group_t

Socket option for multicast group Deprecated.

Public Attributes

uint32_t	action Multicast group action.
sl_wisun_ip_address_t	address Multicast group address.

Public Attribute Documentation

action

```
uint32_t sl_wisun_socket_option_multicast_group_t::action
```

Multicast group action.

Definition at line 691 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

address

```
sl_wisun_ip_address_t sl_wisun_socket_option_multicast_group_t::address
```

Multicast group address.

Definition at line 693 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_socket_option_send_buffer_limit_t

Socket option for send buffer limit Deprecated.

Public Attributes

uint32_t	limit Send buffer limit.

Public Attribute Documentation

limit

```
uint32_t sl_wisun_socket_option_send_buffer_limit_t::limit
```

Send buffer limit.

Definition at line 700 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_socket_option_edfe_mode_t

Socket option for EDFE mode Deprecated.

Public Attributes

uint32_t	mode Socket EDFE mode (1 to enable, 0 to disable)

Public Attribute Documentation

mode

```
uint32_t sl_wisun_socket_option_edfe_mode_t::mode
```

Socket EDFE mode (1 to enable, 0 to disable)

Definition at line 707 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_socket_option_unicast_hop_limit

Socket option for socket unicast hop limit Deprecated.

Public Attributes

int16_t	hop_limit Socket unicast hop limit (0 to 255 hops, -1 to use default)
uint16_t	reserved Reserved, set to 0.

Public Attribute Documentation

hop_limit

```
int16_t sl_wisun_socket_option_unicast_hop_limit::hop_limit
```

Socket unicast hop limit (0 to 255 hops, -1 to use default)

Definition at line 714 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

reserved

```
uint16_t sl_wisun_socket_option_unicast_hop_limit::reserved
```

Reserved, set to 0.

Definition at line 716 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_socket_option_multicast_hop_limit

Socket option for socket multicast hop limit Deprecated.

Public Attributes

int16_t	hop_limit Socket multicast hop limit (0 to 255 hops, -1 to use default)
uint16_t	reserved Reserved, set to 0.

Public Attribute Documentation

hop_limit

```
int16_t sl_wisun_socket_option_multicast_hop_limit::hop_limit
```

Socket multicast hop limit (0 to 255 hops, -1 to use default)

Definition at line `723` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

reserved

```
uint16_t sl_wisun_socket_option_multicast_hop_limit::reserved
```

Reserved, set to 0.

Definition at line `725` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_socket_option_data_t

socket options

Public Attributes

sl_wisun_socket_option_event_mode_t	event_mode Socket event mode Deprecated.
sl_wisun_socket_option_multicast_group_t	multicast_group Socket multicast group Deprecated.
sl_wisun_socket_option_send_buffer_limit_t	send_buffer_limit Socket send buffer limit Deprecated.
sl_wisun_socket_option_edfe_mode_t	edfe_mode Socket EDFE mode Deprecated.
sl_wisun_socket_option_unicast_hop_limit	unicast_hop_limit Socket unicast hop limit Deprecated.
sl_wisun_socket_option_multicast_hop_limit	multicast_hop_limit Socket multicast hop limit Deprecated.
int32_t	value Option-specific value.
in6_addr_t	ipv6_address IPv6 address.

Public Attribute Documentation

event_mode

```
sl_wisun_socket_option_event_mode_t sl_wisun_socket_option_data_t::event_mode
```

Socket event mode Deprecated.

Definition at line 733 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

multicast_group

```
sl_wisun_socket_option_multicast_group_t sl_wisun_socket_option_data_t::multicast_group
```

Socket multicast group Deprecated.

Definition at line 736 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

send_buffer_limit

> sl_wisun_socket_option_send_buffer_limit_t sl_wisun_socket_option_data_t::send_buffer_limit

Socket send buffer limit Deprecated.

Definition at line 739 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

edfe_mode

> sl_wisun_socket_option_edfe_mode_t sl_wisun_socket_option_data_t::edfe_mode

Socket EDFE mode Deprecated.

Definition at line 742 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

unicast_hop_limit

> sl_wisun_socket_option_unicast_hop_limit sl_wisun_socket_option_data_t::unicast_hop_limit

Socket unicast hop limit Deprecated.

Definition at line 745 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

multicast_hop_limit

> sl_wisun_socket_option_multicast_hop_limit sl_wisun_socket_option_data_t::multicast_hop_limit

Socket multicast hop limit Deprecated.

Definition at line 748 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

value

> int32_t sl_wisun_socket_option_data_t::value

Option-specific value.

Definition at line 750 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

ipv6_address

> in6_addr_t sl_wisun_socket_option_data_t::ipv6_address

IPv6 address.

Definition at line 752 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_neighbor_info_t

RPL neighbor information.

Public Attributes

in6_addr_t	link_local_address Link-local address.
in6_addr_t	global_address ULA/GUA address (unspecified address :: if unknown)
uint32_t	type Neighbor type (sl_wisun_neighbor_type_t)
uint32_t	lifetime Remaining lifetime (Link lifetime for parents, EARO lifetime for children) in seconds.
uint32_t	mac_tx_count MAC TX packet count.
uint32_t	mac_tx_failed_count MAC TX failed count.
uint32_t	mac_tx_ms_count MAC TX packet count using mode switch.
uint32_t	mac_tx_ms_failed_count MAC TX failed count using mode switch.
uint32_t	mac_rx_count MAC RX packet count.
uint16_t	rpl_rank RPL Rank value for parents (0xffff if unknown or child)
uint16_t	etx Measured ETX value if known (0xffff if unknown)
uint16_t	routing_cost ETX to Border Router.
uint16_t	pan_size Number devices connected to Border Router.
uint8_t	rsl_out Parent RSSI Out measured RSSI value (0xff if unknown) Calculated using EWMA specified by Wi-SUN from range of -174 (0) to +80 (254) dBm.
uint8_t	rsl_in Parent RSSI In measured RSSI value (0xff if unknown) Calculated using EWMA specified by Wi-SUN from range of -174 (0) to +80 (254) dBm.
int8_t	rssi RSSI of the last received packet in integer dBm. */.

uint8_t is_lfn
Indicate if the device is an LFN. 1 = LFN, 0 = FFN.

uint8_t phy_mode_id_count
Number of PhyModeId supported.

uint8_t phy_mode_ids
List of phy_mode_id_count PhyModeId.

uint8_t is_mdr_command_capable
Indicate if the neighbor supports MAC mode switch.

Public Attribute Documentation

link_local_address

```
in6_addr_t sl_wisun_neighbor_info_t::link_local_address
```

Link-local address.

Definition at line 770 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

global_address

```
in6_addr_t sl_wisun_neighbor_info_t::global_address
```

ULA/GUA address (unspecified address :: if unknown)

Definition at line 772 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

type

```
uint32_t sl_wisun_neighbor_info_t::type
```

Neighbor type (sl_wisun_neighbor_type_t)

Definition at line 774 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

lifetime

```
uint32_t sl_wisun_neighbor_info_t::lifetime
```

Remaining lifetime (Link lifetime for parents, EARO lifetime for children) in seconds.

Definition at line 776 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

mac_tx_count

```
uint32_t sl_wisun_neighbor_info_t::mac_tx_count
```

MAC TX packet count.

Definition at line 778 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

mac_tx_failed_count

> uint32_t sl_wisun_neighbor_info_t::mac_tx_failed_count

MAC TX failed count.

Definition at line 780 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

mac_tx_ms_count

> uint32_t sl_wisun_neighbor_info_t::mac_tx_ms_count

MAC TX packet count using mode switch.

Definition at line 782 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

mac_tx_ms_failed_count

> uint32_t sl_wisun_neighbor_info_t::mac_tx_ms_failed_count

MAC TX failed count using mode switch.

Definition at line 784 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

mac_rx_count

> uint32_t sl_wisun_neighbor_info_t::mac_rx_count

MAC RX packet count.

Definition at line 786 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rpl_rank

> uint16_t sl_wisun_neighbor_info_t::rpl_rank

RPL Rank value for parents (0xffff if unknown or child)

Definition at line 788 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

etx

> uint16_t sl_wisun_neighbor_info_t::etx

Measured ETX value if known (0xffff if unknown)

Definition at line 790 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

routing_cost

```
uint16_t sl_wisun_neighbor_info_t::routing_cost
```

ETX to Border Router.

Definition at line 792 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

pan_size

```
uint16_t sl_wisun_neighbor_info_t::pan_size
```

Number devices connected to Border Router.

Definition at line 794 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rsl_out

```
uint8_t sl_wisun_neighbor_info_t::rsl_out
```

Parent RSSI Out measured RSSI value (0xff if unknown) Calculated using EWMA specified by Wi-SUN from range of -174 (0) to +80 (254) dBm.

Definition at line 797 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rsl_in

```
uint8_t sl_wisun_neighbor_info_t::rsl_in
```

Parent RSSI In measured RSSI value (0xff if unknown) Calculated using EWMA specified by Wi-SUN from range of -174 (0) to +80 (254) dBm.

Definition at line 800 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

rssi

```
int8_t sl_wisun_neighbor_info_t::rssi
```

RSSI of the last received packet in integer dBm. */.

Definition at line 802 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

is_lfn

> uint8_t sl_wisun_neighbor_info_t::is_lfn

Indicate if the device is an LFN. 1 = LFN, 0 = FFN.

Definition at line `804` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

phy_mode_id_count

> uint8_t sl_wisun_neighbor_info_t::phy_mode_id_count

Number of PhyModeId supported.

Definition at line `806` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

phy_mode_ids

> uint8_t sl_wisun_neighbor_info_t::phy_mode_ids[SL_WISUN_MAX_PHY_MODE_ID_COUNT]

List of phy_mode_id_count PhyModeId.

Definition at line `808` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

is_mdr_command_capable

> uint8_t sl_wisun_neighbor_info_t::is_mdr_command_capable

Indicate if the neighbor supports MAC mode switch.

Definition at line `810` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_trace_group_config_t

Configure the trace level of 1 group.

Public Attributes

uint8_t	group_id Trace Group ID. Coded with enum sl_wisun_trace_group_t.
uint8_t	trace_level Maximum trace level to display for this group.

Public Attribute Documentation

group_id

```
uint8_t sl_wisun_trace_group_config_t::group_id
```

Trace Group ID. Coded with enum sl_wisun_trace_group_t.

Definition at line `884` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

trace_level

```
uint8_t sl_wisun_trace_group_config_t::trace_level
```

Maximum trace level to display for this group.

It is coded using enum sl_wisun_trace_level_t.

Definition at line `887` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_network_info_t

Wi-SUN network information.

Public Attributes

uint16_t	pan_id PAN ID.

Public Attribute Documentation

pan_id

```
uint16_t sl_wisun_network_info_t::pan_id
```

PAN ID.

Definition at line 953 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

sl_wisun_rpl_info_t

RPL information.

Public Attributes

uint16_t	dodag_rank DODAG rank or the node.
uint16_t	dag_max_rank_increase DAG max rank increase, the allowable increase in Rank in support of local repair (0 to disable the mechanism)
uint16_t	min_hop_rank_increase Min hop rank increase, minimum increase in Rank between a node and any of its DODAG parents.
uint16_t	lifetime_unit Lifetime unit, unit in seconds that is used to express route lifetimes in RPL.
uint8_t	instance_id Instance ID, set by the DODAG root, it indicates of which RPL Instance the DODAG is a part.
uint8_t	dodag_version_number DODAG version number, set by the DODAG root.
uint8_t	grounded Grounded, indicates whether the DODAG advertised can satisfy the application-defined goal.
uint8_t	mode_of_operation Mode of Operation (MOP), must be 1 for Non-Storing Mode of Operation.
uint8_t	dodag_preference DODAG Preference, defines how preferable the root of this DODAG is compared to other DODAG roots within the instance.
uint8_t	dodag_dtsn Destination Advertisement Trigger Sequence Number (DTSN)
uint8_t	dio_interval_min DIO minimum interval, used to configure Imin of the DIO Trickle timer.
uint8_t	dio_interval_doublings DIO interval doublings, used to configure Imax of the DIO Trickle timer.
uint8_t	dio_redundancy_constant DIO redundancy constant, used to configure k of the DIO Trickle timer.
uint8_t	default_lifetime Default lifetime, lifetime that is used as default for all RPL routes.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

dodag_rank

uint16_t sl_wisun_rpl_info_t::dodag_rank

DODAG rank or the node.

Definition at line 961 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

dag_max_rank_increase

uint16_t sl_wisun_rpl_info_t::dag_max_rank_increase

DAG max rank increase, the allowable increase in Rank in support of local repair (0 to disable the mechanism)

Definition at line 964 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

min_hop_rank_increase

uint16_t sl_wisun_rpl_info_t::min_hop_rank_increase

Min hop rank increase, minimum increase in Rank between a node and any of its DODAG parents.

Definition at line 967 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

lifetime_unit

uint16_t sl_wisun_rpl_info_t::lifetime_unit

Lifetime unit, unit in seconds that is used to express route lifetimes in RPL.

Definition at line 970 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

instance_id

uint8_t sl_wisun_rpl_info_t::instance_id

Instance ID, set by the DODAG root, it indicates of which RPL Instance the DODAG is a part.

Definition at line 973 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

dodag_version_number

uint8_t sl_wisun_rpl_info_t::dodag_version_number

DODAG version number, set by the DODAG root.

Definition at line 975 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

grounded

uint8_t sl_wisun_rpl_info_t::grounded

Grounded, indicates whether the DODAG advertised can satisfy the application-defined goal.

If set, the DODAG is grounded. If cleared, the DODAG is floating.

Definition at line 979 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

mode_of_operation

uint8_t sl_wisun_rpl_info_t::mode_of_operation

Mode of Operation (MOP), must be 1 for Non-Storing Mode of Operation.

Definition at line 981 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

dodag_preference

uint8_t sl_wisun_rpl_info_t::dodag_preference

DODAG Preference, defines how preferable the root of this DODAG is compared to other DODAG roots within the instance.

DAGPreference ranges from 0x00 (least preferred) to 0x07 (most preferred).

Definition at line 986 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

dodag_dtsn

uint8_t sl_wisun_rpl_info_t::dodag_dtsn

Destination Advertisement Trigger Sequence Number (DTSN)

Definition at line 988 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

dio_interval_min

uint8_t sl_wisun_rpl_info_t::dio_interval_min

DIO minimum interval, used to configure Imin of the DIO Trickle timer.

Definition at line 991 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

dio_interval_doublings

uint8_t sl_wisun_rpl_info_t::dio_interval_doublings

DIO interval doublings, used to configure Imax of the DIO Trickle timer.

Definition at line 994 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h

dio_redundancy_constant

```
uint8_t sl_wisun_rpl_info_t::dio_redundancy_constant
```

DIO redundancy constant, used to configure k of the DIO Trickle timer.

Definition at line `997` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

default_lifetime

```
uint8_t sl_wisun_rpl_info_t::default_lifetime
```

Default lifetime, lifetime that is used as default for all RPL routes.

Expressed in units of Lifetime Units.

Definition at line `1000` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

reserved

```
uint8_t sl_wisun_rpl_info_t::reserved[2]
```

Reserved, set to zero.

Definition at line `1002` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_types.h`

sl_wisun_trickle_params_t

Trickle parameter set.

Public Attributes

uint16_t	imin_s Minimum interval size (seconds)
uint16_t	imax_s Maximum interval size (seconds)
uint8_t	k Redundancy constant (0 for infinity)
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

imin_s

```
uint16_t sl_wisun_trickle_params_t::imin_s
```

Minimum interval size (seconds)

Definition at line 51 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

imax_s

```
uint16_t sl_wisun_trickle_params_t::imax_s
```

Maximum interval size (seconds)

Definition at line 53 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

k

```
uint8_t sl_wisun_trickle_params_t::k
```

Redundancy constant (0 for infinity)

Definition at line 55 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

reserved

uint8_t sl_wisun_trickle_params_t::reserved[3]

Reserved, set to zero.

Definition at line 57 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_params_discovery

PAN discovery parameter set.

Public Attributes

sl_wisun_trickle_params_t	trickle_pa PAN Advertisement trickle timer.
sl_wisun_trickle_params_t	trickle_pas PAN Advertisement Solicit trickle timer.
uint8_t	eapol_target_min_sens Minimum signal level for a node to be selected as the EAPOL target for authentication immediately after a PAN Advertisement reception.
uint8_t	allow_skip If true, allow join state 1 to be skipped using cached information from the previous connection.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

trickle_pa

```
sl_wisun_trickle_params_t sl_wisun_params_discovery::trickle_pa
```

PAN Advertisement trickle timer.

Definition at line `65` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h`

trickle_pas

```
sl_wisun_trickle_params_t sl_wisun_params_discovery::trickle_pas
```

PAN Advertisement Solicit trickle timer.

Definition at line `67` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h`

eapol_target_min_sens

```
uint8_t sl_wisun_params_discovery::eapol_target_min_sens
```

Minimum signal level for a node to be selected as the EAPOL target for authentication immediately after a PAN Advertisement reception.

Range from -174 (0) to +80 (254) dBm, 255 to disable feature.

Definition at line 71 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

allow_skip

> uint8_t sl_wisun_params_discovery::allow_skip

If true, allow join state 1 to be skipped using cached information from the previous connection.

Definition at line 74 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

reserved

> uint8_t sl_wisun_params_discovery::reserved[2]

Reserved, set to zero.

Definition at line 76 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_params_eapol

Authentication parameter set.

Public Attributes

sl_wisun_trickle_params_t	sec_prot_trickle Security protocol trickle timer.
uint32_t	pmk_lifetime_m PMK lifetime (minutes)
uint32_t	ptk_lifetime_m PTK lifetime (minutes)
uint16_t	sec_prot_retry_timeout_s Security protocol retry timeout (seconds)
uint16_t	initial_key_min_s Initial EAPOL-Key first Tx min delay (seconds)
uint16_t	initial_key_max_s Initial EAPOL-Key first Tx max delay (seconds)
uint16_t	initial_key_retry_min_s Initial EAPOL-Key retry exponential backoff min (seconds)
uint16_t	initial_key_retry_max_s Initial EAPOL-Key retry exponential backoff max (seconds)
uint16_t	initial_key_retry_max_limit_s Initial EAPOL-Key retry exponential backoff max limit (seconds)
uint16_t	temp_min_timeout_s Temporary neighbor link minimum timeout (seconds)
uint16_t	gtk_request_imin_m GTK_REQUEST_IMIN (minutes)
uint16_t	gtk_request_imax_m GTK_REQUEST_IMAX (minutes)
uint16_t	gtk_max_mismatch_m GTK_MAX_MISMATCH (minutes)
uint16_t	lgtk_max_mismatch_m LGTK_MAX_MISMATCH (minutes)
uint8_t	sec_prot_trickle_expirations Security protocol trickle timer expirations.
uint8_t	initial_key_retry_limit Initial EAPOL-Key retry limit.
uint8_t	allow_skip If true, allow join state 2 to be skipped using cached credentials from the previous connection.

uint8_t reserved
Reserved, set to zero.

Public Attribute Documentation

sec_prot_trickle

> sl_wisun_trickle_params_t sl_wisun_params_eapol::sec_prot_trickle

Security protocol trickle timer.

Definition at line 84 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

pmk_lifetime_m

> uint32_t sl_wisun_params_eapol::pmk_lifetime_m

PMK lifetime (minutes)

Definition at line 86 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

ptk_lifetime_m

> uint32_t sl_wisun_params_eapol::ptk_lifetime_m

PTK lifetime (minutes)

Definition at line 88 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sec_prot_retry_timeout_s

> uint16_t sl_wisun_params_eapol::sec_prot_retry_timeout_s

Security protocol retry timeout (seconds)

Definition at line 90 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

initial_key_min_s

> uint16_t sl_wisun_params_eapol::initial_key_min_s

Initial EAPOL-Key first Tx min delay (seconds)

Definition at line 92 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

initial_key_max_s

uint16_t sl_wisun_params_eapol::initial_key_max_s

Initial EAPOL-Key first Tx max delay (seconds)

Definition at line 94 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

initial_key_retry_min_s

uint16_t sl_wisun_params_eapol::initial_key_retry_min_s

Initial EAPOL-Key retry exponential backoff min (seconds)

Definition at line 96 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

initial_key_retry_max_s

uint16_t sl_wisun_params_eapol::initial_key_retry_max_s

Initial EAPOL-Key retry exponential backoff max (seconds)

Definition at line 98 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

initial_key_retry_max_limit_s

uint16_t sl_wisun_params_eapol::initial_key_retry_max_limit_s

Initial EAPOL-Key retry exponential backoff max limit (seconds)

Definition at line 100 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

temp_min_timeout_s

uint16_t sl_wisun_params_eapol::temp_min_timeout_s

Temporary neighbor link minimum timeout (seconds)

Definition at line 102 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

gtk_request_imin_m

uint16_t sl_wisun_params_eapol::gtk_request_imin_m

GTK_REQUEST_IMIN (minutes)

Definition at line 104 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

gtk_request_imax_m

uint16_t sl_wisun_params_eapol::gtk_request_imax_m

GTK_REQUEST_IMAX (minutes)

Definition at line 106 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

gtk_max_mismatch_m

uint16_t sl_wisun_params_eapol::gtk_max_mismatch_m

GTK_MAX_MISMATCH (minutes)

Definition at line 108 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

lgtk_max_mismatch_m

uint16_t sl_wisun_params_eapol::lgtk_max_mismatch_m

LGTK_MAX_MISMATCH (minutes)

Definition at line 110 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sec_prot_trickle_expirations

uint8_t sl_wisun_params_eapol::sec_prot_trickle_expirations

Security protocol trickle timer expirations.

Definition at line 112 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

initial_key_retry_limit

uint8_t sl_wisun_params_eapol::initial_key_retry_limit

Initial EAPOL-Key retry limit.

Definition at line 114 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

allow_skip

uint8_t sl_wisun_params_eapol::allow_skip

If true, allow join state 2 to be skipped using cached credentials from the previous connection.

Definition at line 117 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

reserved

uint8_t sl_wisun_params_eapol::reserved[3]

Reserved, set to zero.

Definition at line 119 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_params_configuration

PAN configuration parameter set.

Public Attributes

sl_wisun_trickle_params_t	trickle_pc PAN Configuration trickle timer.
sl_wisun_trickle_params_t	trickle_pcs PAN Configuration Solicit trickle timer.

Public Attribute Documentation

trickle_pc

```
sl_wisun_trickle_params_t sl_wisun_params_configuration::trickle_pc
```

PAN Configuration trickle timer.

Definition at line 127 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

trickle_pcs

```
sl_wisun_trickle_params_t sl_wisun_params_configuration::trickle_pcs
```

PAN Configuration Solicit trickle timer.

Definition at line 129 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_params_rpl

RPL parameter set.

Public Attributes

uint16_t	dis_max_delay_first_s RPL first DIS maximum delay (seconds)
uint16_t	dis_max_delay_s RPL DIS maximum delay (seconds)
uint16_t	init_parent_selection_s Delay for preferred parent selection after first DIO reception (seconds)
uint16_t	etx_probe_period_max_s Maximum period of NS probes used to get samples for ETX calculation (seconds)
uint8_t	etx_samples_init Number of samples used to calculate ETX during join state 4.
uint8_t	etx_samples_refresh Number of samples used to refresh ETX.
uint8_t	candidate_parents_max RPL max candidate parents.
uint8_t	parents_max RPL max parents.

Public Attribute Documentation

dis_max_delay_first_s

> uint16_t sl_wisun_params_rpl::dis_max_delay_first_s

RPL first DIS maximum delay (seconds)

Definition at line 137 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

dis_max_delay_s

> uint16_t sl_wisun_params_rpl::dis_max_delay_s

RPL DIS maximum delay (seconds)

Definition at line 139 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

init_parent_selection_s

uint16_t sl_wisun_params_rpl::init_parent_selection_s

Delay for preferred parent selection after first DIO reception (seconds)

Definition at line 141 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

etx_probe_period_max_s

uint16_t sl_wisun_params_rpl::etx_probe_period_max_s

Maximum period of NS probes used to get samples for ETX calculation (seconds)

Definition at line 143 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

etx_samples_init

uint8_t sl_wisun_params_rpl::etx_samples_init

Number of samples used to calculate ETX during join state 4.

Definition at line 145 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

etx_samples_refresh

uint8_t sl_wisun_params_rpl::etx_samples_refresh

Number of samples used to refresh ETX.

Definition at line 147 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

candidate_parents_max

uint8_t sl_wisun_params_rpl::candidate_parents_max

RPL max candidate parents.

Definition at line 149 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

parents_max

uint8_t sl_wisun_params_rpl::parents_max

RPL max parents.

Definition at line 151 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_params_mpl

MPL parameter set.

Public Attributes

sl_wisun_trickle_params_t	trickle MPL trickle timer.
uint16_t	seed_set_entry_lifetime_s MPL seed set entry lifetime (seconds)
uint8_t	trickle_expirations MPL trickle timer expirations.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

trickle

```
sl_wisun_trickle_params_t sl_wisun_params_mpl::trickle
```

MPL trickle timer.

Definition at line 159 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

seed_set_entry_lifetime_s

```
uint16_t sl_wisun_params_mpl::seed_set_entry_lifetime_s
```

MPL seed set entry lifetime (seconds)

Definition at line 161 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

trickle_expirations

```
uint8_t sl_wisun_params_mpl::trickle_expirations
```

MPL trickle timer expirations.

Definition at line 163 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

reserved

uint8_t sl_wisun_params_mpl::reserved

Reserved, set to zero.

Definition at line 165 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_params_misc

Misc parameter set.

Public Attributes

uint16_t	temp_link_min_timeout_s Temporary neighbor link minimum timeout.
uint8_t	pan_timeout_m Border router communication timeout PAN_TIMEOUT.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

temp_link_min_timeout_s

```
uint16_t sl_wisun_params_misc::temp_link_min_timeout_s
```

Temporary neighbor link minimum timeout.

Definition at line 173 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

pan_timeout_m

```
uint8_t sl_wisun_params_misc::pan_timeout_m
```

Border router communication timeout PAN_TIMEOUT.

Definition at line 175 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

reserved

```
uint8_t sl_wisun_params_misc::reserved
```

Reserved, set to zero.

Definition at line 177 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_connection_params_t

FFN parameter set.

Public Attributes

uint32_t	version Version of this API.
sl_wisun_params_discovery	discovery PAN discovery parameter set.
sl_wisun_params_configuration	configuration PAN configuration parameter set.
sl_wisun_params_eapol	eapol Authentication parameter set.
sl_wisun_params_rpl	rpl RPL parameter set.
sl_wisun_params_mpl	mpl MPL parameter set.
sl_wisun_params_misc	misc Misc parameter set.

Public Attribute Documentation

version

```
uint32_t sl_wisun_connection_params_t::version
```

Version of this API.

This field allows to store the parameters in an NVM and check on reload that they are compatible with the stack if there was an update.

Definition at line 190 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

discovery

```
sl_wisun_params_discovery sl_wisun_connection_params_t::discovery
```

PAN discovery parameter set.

Definition at line 192 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

configuration

sl_wisun_params_configuration sl_wisun_connection_params_t::configuration

PAN configuration parameter set.

Definition at line 194 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

eapol

sl_wisun_params_eapol sl_wisun_connection_params_t::eapol

Authentication parameter set.

Definition at line 196 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

rpl

sl_wisun_params_rpl sl_wisun_connection_params_t::rpl

RPL parameter set.

Definition at line 198 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

mpl

sl_wisun_params_mpl sl_wisun_connection_params_t::mpl

MPL parameter set.

Definition at line 200 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

misc

sl_wisun_params_misc sl_wisun_connection_params_t::misc

Misc parameter set.

Definition at line 202 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

sl_wisun_lfn_params_connection_t

LFN connection parameters.

Public Attributes

uint8_t	discovery_slot_time_ms Duration of LFN PAN Advertisement (LPA) listening slot (millisecond) Specification range [15, 255].
uint8_t	discovery_slots Number of LPA slots for which an LFN shall listen for LPA frames Specification range [1, 255].
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

discovery_slot_time_ms

```
uint8_t sl_wisun_lfn_params_connection_t::discovery_slot_time_ms
```

Duration of LFN PAN Advertisement (LPA) listening slot (millisecond) Specification range [15, 255].

Definition at line 52 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

discovery_slots

```
uint8_t sl_wisun_lfn_params_connection_t::discovery_slots
```

Number of LPA slots for which an LFN shall listen for LPA frames Specification range [1, 255].

Definition at line 55 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

reserved

```
uint8_t sl_wisun_lfn_params_connection_t::reserved[2]
```

Reserved, set to zero.

Definition at line 57 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

sl_wisun_lfn_params_data_layer_t

LFN data layer parameters.

Public Attributes

uint32_t	unicast_interval_ms Initial LFN Unicast interval proposed by the LFN (milliseconds).
uint32_t	unicast_interval_min_ms Minimum acceptable LFN unicast interval (milliseconds)
uint32_t	unicast_interval_max_ms Maximum acceptable LFN unicast interval (milliseconds)
uint8_t	lfn_maintain_parent_time The LFN assumes its parent is lost after [lfn_maintain_parent_time] number of Broadcast sync periods with no message received from its parent.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

unicast_interval_ms

```
uint32_t sl_wisun_lfn_params_data_layer_t::unicast_interval_ms
```

Initial LFN Unicast interval proposed by the LFN (milliseconds).

The real unicast interval duration is negotiated with the LFN parent, between unicast_interval_min_ms and unicast_interval_max_ms.

Definition at line `67` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h`

unicast_interval_min_ms

```
uint32_t sl_wisun_lfn_params_data_layer_t::unicast_interval_min_ms
```

Minimum acceptable LFN unicast interval (milliseconds)

Definition at line `69` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h`

unicast_interval_max_ms

```
uint32_t sl_wisun_lfn_params_data_layer_t::unicast_interval_max_ms
```

Maximum acceptable LFN unicast interval (milliseconds)

Definition at line 71 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

lfn_maintain_parent_time

> uint8_t sl_wisun_lfn_params_data_layer_t::lfn_maintain_parent_time

The LFN assumes its parent is lost after [lfn_maintain_parent_time] number of Broadcast sync periods with no message received from its parent.

Specification range [1, 60]

Definition at line 75 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

reserved

> uint8_t sl_wisun_lfn_params_data_layer_t::reserved[3]

Reserved, set to zero.

Definition at line 77 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

sl_wisun_lfn_params_network_t

LFN network parameters.

Public Attributes

uint16_t	lfn_registration_lifetime_m Address registration lifetime (IPv6 lease duration) the LFN requires to the Border Router (minutes).
uint8_t	lfn_na_wait_duration_m Duration for which an LFN waits for a registration confirmation (minutes).
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

lfn_registration_lifetime_m

> uint16_t sl_wisun_lfn_params_network_t::lfn_registration_lifetime_m

Address registration lifetime (IPv6 lease duration) the LFN requires to the Border Router (minutes).

Specification range [1440, 5040]

Definition at line 87 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

lfn_na_wait_duration_m

> uint8_t sl_wisun_lfn_params_network_t::lfn_na_wait_duration_m

Duration for which an LFN waits for a registration confirmation (minutes).

Specification range [30, 120]

Definition at line 90 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

reserved

> uint8_t sl_wisun_lfn_params_network_t::reserved

Reserved, set to zero.

Definition at line 92 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

sl_wisun_lfn_params_power_t

LFN power parameters.

Public Attributes

uint16_t	listening_window_min_us Minimum duration of the listening window.
uint16_t	window_margin_min_us Minimum margin added to the listening window (before and after).
uint8_t	broadcast_lts_only If true, the LFN wakes up only for broadcast slots containing synchronization information.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

listening_window_min_us

> uint16_t sl_wisun_lfn_params_power_t::listening_window_min_us

Minimum duration of the listening window.

Applies to both Unicast and Broadcast slots.

Definition at line 101 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

window_margin_min_us

> uint16_t sl_wisun_lfn_params_power_t::window_margin_min_us

Minimum margin added to the listening window (before and after).

The real margin increases with aging synchronization info.

Definition at line 104 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

broadcast_lts_only

> uint8_t sl_wisun_lfn_params_power_t::broadcast_lts_only

If true, the LFN wakes up only for broadcast slots containing synchronization information.

If false, the node wakes up on every LFN broadcast slot.

Definition at line 108 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

reserved

```
uint8_t sl_wisun_lfn_params_power_t::reserved[3]
```

Reserved, set to zero.

Definition at line 110 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

sl_wisun_lfn_params_t

LFN parameter set.

Public Attributes

uint32_t	version Version of this API.
sl_wisun_lfn_params_connection_t	connection LFN connection parameters.
sl_wisun_lfn_params_data_layer_t	data_layer LFN data layer parameters.
sl_wisun_lfn_params_network_t	network LFN network parameters.
sl_wisun_lfn_params_power_t	power LFN power parameters.

Public Attribute Documentation

version

```
uint32_t sl_wisun_lfn_params_t::version
```

Version of this API.

This field allows to store the parameters in an NVM and check on reload that they are compatible with the stack if there was an update.

Definition at line 123 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

connection

```
sl_wisun_lfn_params_connection_t sl_wisun_lfn_params_t::connection
```

LFN connection parameters.

Definition at line 125 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

data_layer

```
sl_wisun_lfn_params_data_layer_t sl_wisun_lfn_params_t::data_layer
```

LFN data layer parameters.

Definition at line 127 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

network

```
sl_wisun_lfn_params_network_t sl_wisun_lfn_params_t::network
```

LFN network parameters.

Definition at line 129 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

power

```
sl_wisun_lfn_params_power_t sl_wisun_lfn_params_t::power
```

LFN power parameters.

Definition at line 131 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

Predefined FFN parameter sets

Predefined FFN parameter sets

Predefined FFN parameter sets for sl_wisun_set_connection_parameters().

These parameter sets can be used either as-is or used as an initialization value for an application-specific parameter set.

Variables

const sl_wisun_connection_params_t	SL_WISUN_PARAMS_PROFILE_TEST Profile for development (shorter connection time)
const sl_wisun_connection_params_t	SL_WISUN_PARAMS_PROFILE_CERTIF Profile for certification testing.
const sl_wisun_connection_params_t	SL_WISUN_PARAMS_PROFILE_SMALL Profile for a small network.
const sl_wisun_connection_params_t	SL_WISUN_PARAMS_PROFILE_MEDIUM Profile for a medium network.
const sl_wisun_connection_params_t	SL_WISUN_PARAMS_PROFILE_LARGE Profile for a large network.

Variable Documentation

SL_WISUN_PARAMS_PROFILE_TEST

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_TEST
```

Profile for development (shorter connection time)

Definition at line 218 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

SL_WISUN_PARAMS_PROFILE_CERTIF

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_CERTIF
```

Profile for certification testing.

Definition at line 295 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

SL_WISUN_PARAMS_PROFILE_SMALL

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_SMALL
```

Profile for a small network.

Definition at line 372 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

SL_WISUN_PARAMS_PROFILE_MEDIUM

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_MEDIUM
```

Profile for a medium network.

Definition at line 449 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

SL_WISUN_PARAMS_PROFILE_LARGE

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_LARGE
```

Profile for a large network.

Definition at line 526 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_connection_params_api.h

Predefined LFN parameter sets

Predefined LFN parameter sets

Predefined LFN parameter sets for sl_wisun_set_lfn_parameters().

These parameter sets can be used either as-is or used as an initialization value for an application-specific parameter set.

Variables

const sl_wisun_lfn_params_t	SL_WISUN_PARAMS_LFN_TEST Profile for test usage, best performance but highest power consumption.
const sl_wisun_lfn_params_t	SL_WISUN_PARAMS_LFN_BALANCED Profile providing balance between power consumption and performance.
const sl_wisun_lfn_params_t	SL_WISUN_PARAMS_LFN_ECO Profile optimized for low power consumption.

Variable Documentation

SL_WISUN_PARAMS_LFN_TEST

```
const sl_wisun_lfn_params_t SL_WISUN_PARAMS_LFN_TEST
```

Profile for test usage, best performance but highest power consumption.

Definition at line 147 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

SL_WISUN_PARAMS_LFN_BALANCED

```
const sl_wisun_lfn_params_t SL_WISUN_PARAMS_LFN_BALANCED
```

Profile providing balance between power consumption and performance.

Definition at line 175 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

SL_WISUN_PARAMS_LFN_ECO

```
const sl_wisun_lfn_params_t SL_WISUN_PARAMS_LFN_ECO
```

Profile optimized for low power consumption.

Definition at line 203 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/sl_wisun_lfn_params_api.h

Socket API

Socket API

Modules

sockaddr

in6_addr

sockaddr_in6

Protocol levels used for socket_setsockopt.

#define SOL_SOCKET 0
Socket option level.

#define SOL_APPLICATION 1
Application socket option level.

#define IPPROTO_IPV6 41
IPV6 socket option level.

application level socket options

application level socket options summaryopt_name data type set/get sendmsg recvmsg SO_EVENT_MODE int16_t Set only No Yes SO_NONBLOCK int16_t Set only Yes Yes

#define SO_EVENT_MODE 10

#define SOCKET_EVENT_MODE SO_EVENT_MODE

#define SO_NONBLOCK 11
Enable/disable nonblocking mode.

socket level options

socket level options summaryopt_name data type set/get sendmsg recvmsg SO_RCVBUF int32_t Set/Get No Yes SO_SNDBUF int32_t Set/Get Yes No SO_SNDLOWAT int32_t Set/Get Yes No

#define SO_RCVBUF 1

#define SO_SNDBUF 2
Specify send buffer size in payload bytes.

#define SO_SNDLOWAT 4
Specify send low water mark in payload bytes.

IPv6 socket options

IPv6 socket options summaryopt_name Data type set/getsockopt sendmsg recvmsg IPV6_UNICAST_HOPS int16_t Set/Get Yes No IPV6_MULTICAST_HOPS int16_t Set/Get Yes No IPV6_JOIN_GROUP ns_ipv6_mreq_t Set only Yes Yes IPV6_LEAVE_GROUP ns_ipv6_mreq_t Set only Yes Yes SO_EDFE_MODE uint32_t Set only Yes No

#define IPV6_UNICAST_HOPS 2

#define IPV6_MULTICAST_HOPS 3
Set the multicast hop limit for the socket.

#define IPV6_JOIN_GROUP 15
Join a multicast group.

#define IPV6_LEAVE_GROUP 16
Leave a multicast group.

#define SO_EDFE_MODE 0xfb
Experimental: Enable Extended Directed Frame Exchange mode.

#define SOCKET_EDFE_MODE SO_EDFE_MODE

Enumerations

enum sl_wisun_socket_event_mode_t {
  SL_WISUN_SOCKET_EVENT_MODE_INDICATION = 0
  SL_WISUN_SOCKET_EVENT_MODE_POLLING = 1
}
Enumerations for socket event mode.

enum socket_domain {
  AF_INET6 = 0
}
Supported address families.

enum socket_type {
  SOCK_STREAM = 1
  SOCK_DGRAM = 2
  SOCK_RAW = 3
}
Socket types.

enum socket_protocol {
  IPPROTO_IP = 0
  IPPROTO_ICMP = 1
  IPPROTO_TCP = 2
  IPPROTO_UDP = 3
}
IP protocols.

Typedefs

typedef uint32_t socklen_t
Socket address length type definition.

typedef int32_t sl_wisun_socket_id_t
Socket id.

typedef enum socket_domain — sl_socket_domain_t
Supported address families.

typedef enum socket_type — sl_socket_type_t
Socket types.

typedef enum socket_protocol — sl_socket_protocol_t
IP protocols.

typedef struct in6_addr — in6_addr_t
IPv6 Internet address.

typedef struct sockaddr_in6 — sockaddr_in6_t
IPv6 address format.

Variables

const in6_addr_t — in6addr_any
IPv6 wildcard address.

Functions

int32_t — socket(int32_t domain, int32_t type, int32_t protocol)
Creates an endpoint for communication and returns an Id that refers to that endpoint.

int32_t — close(int32_t sockid)
Close a socket.

int32_t — bind(int32_t sockid, const struct sockaddr *addr, socklen_t addrlen)
Bind a name to a socket.

int32_t — send(int32_t sockid, const void *buff, uint32_t len, int32_t flags)
Send a message on a socket.

int32_t — sendto(int32_t sockid, const void *buff, uint32_t len, int32_t flags, const struct sockaddr *dest_addr, socklen_t addr_len)
Send a message to a given address.

int32_t — recvfrom(int32_t sockid, void *buf, uint32_t len, int32_t flags, struct sockaddr *src_addr, socklen_t *addrlen)
Receive messages from a socket.

int32_t — recv(int32_t sockid, void *buf, uint32_t len, int32_t flags)
Receive a message from a socket.

int32_t — accept(int32_t sockid, struct sockaddr *addr, socklen_t *addrlen)
Accept a connection on a socket.

int32_t — connect(int32_t sockid, const struct sockaddr *addr, socklen_t addrlen)
Initiate a connection on a socket.

int32_t — listen(int32_t sockid, int32_t backlog)
Listen for connections on a socket.

int32_t — setsockopt(int32_t sockid, int32_t level, int32_t optname, const void *optval, socklen_t optlen)
Set socket option designated by optname at a given protocol level to the value pointed by optval.

int32_t — getsockopt(int32_t sockid, int32_t level, int32_t optname, void *optval, socklen_t *optlen)
Get socket option.

__STATIC_INLINE uint32_t	htonl(uint32_t hostlong) Convert the long host byte order to network order.
__STATIC_INLINE uint16_t	htons(uint16_t hostshort) Convert the short host byte order to network order.
__STATIC_INLINE uint32_t	ntohl(uint32_t netlong) Convert the long network byte order to host byte order.
__STATIC_INLINE uint16_t	ntohs(uint16_t netshort) Convert the short network byte order to host byte order.
int32_t	inet_pton(int32_t af, const char *src, void *dst) Convert the IPv4 and IPv6 addresses from text to binary form.
const char *	inet_ntop(int32_t af, const void *src, char *dst, socklen_t size) Convert IPv6 addresses from binary to text form.

Macros

#define	APP_LEVEL_SOCKET SOL_APPLICATION
#define	IPV6_ADDR_SIZE 16 Size of an IPv6 address.
#define	SOCK_NONBLOCK 0X00010000 When bitwise ored with socket's type, it sets the O_NONBLOCK status flag on the opened socket file description.

Protocol levels used for socket_setsockopt. Documentation

SOL_SOCKET

```
#define SOL_SOCKET
```

Value:

```
0
```

Socket option level.

Definition at line 61 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SOL_APPLICATION

```
#define SOL_APPLICATION
```

Value:

```
1
```

Application socket option level.

Definition at line 62 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

IPPROTO_IPV6

#define IPPROTO_IPV6

Value:

```
41
```

IPV6 socket option level.

Definition at line 63 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

application level socket options Documentation

SO_EVENT_MODE

#define SO_EVENT_MODE

Value:

```
10
```

Specify event mode of a socket. When set, optval must point to an uint32_t.

Possible values are:

- SL_WISUN_SOCKET_EVENT_MODE_INDICATION: received data is included in SL_WISUN_MSG_SOCKET_DATA_IND_ID indication
- SL_WISUN_SOCKET_EVENT_MODE_POLLING: only the amount of received data is included SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID indication. recv() or recvfrom() shouldbe invoked after indication reception to retrieve data

Definition at line 93 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SOCKET_EVENT_MODE

#define SOCKET_EVENT_MODE

Value:

```
SO_EVENT_MODE
```

Specify event mode of a socket. When set, optval must point to an uint32_t.

Possible values are:

- SL_WISUN_SOCKET_EVENT_MODE_INDICATION: received data is included in SL_WISUN_MSG_SOCKET_DATA_IND_ID indication
- SL_WISUN_SOCKET_EVENT_MODE_POLLING: only the amount of received data is included SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID indication. recv() or recvfrom() shouldbe invoked after indication reception to retrieve data

Definition at line 94 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SO_NONBLOCK

```
#define SO_NONBLOCK
```

Value:

```
11
```

Enable/disable nonblocking mode.

This option takes an uint32_t value.

Possible values are:

- (1) enables nonblocking mode
- (0) disables nonblocking mode

Definition at line 102 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

socket level options Documentation

SO_RCVBUF

```
#define SO_RCVBUF
```

Value:

```
1
```

Specify receive buffer size in payload bytes. When set, optval must point to an int32_t.

0 means unread data are dropped, unless read in data callback.

Definition at line 123 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SO_SNDBUF

```
#define SO_SNDBUF
```

Value:

```
2
```

Specify send buffer size in payload bytes.

When set, optval must point to an int32_t.

Used only for stream sockets.

Definition at line 129 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SO_SNDLOWAT

```
#define SO_SNDLOWAT
```

Value:

```
4
```

Specify send low water mark in payload bytes.

When set, optval must point to an int32_t.

Definition at line 133 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

IPv6 socket options Documentation

IPV6_UNICAST_HOPS

```
#define IPV6_UNICAST_HOPS
```

Value:

```
2
```

Set the unicast hop limit for the socket. When set, optval must point to an int16_t.

-1 in the value means use the route default, otherwise it should be between 0 and 255.

Definition at line 157 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

IPV6_MULTICAST_HOPS

```
#define IPV6_MULTICAST_HOPS
```

Value:

```
3
```

Set the multicast hop limit for the socket.

When set, optval must point to an int16_t.

-1 in the value means use the route default, otherwise it should be between 0 and 255.

Definition at line 162 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

IPV6_JOIN_GROUP

```
#define IPV6_JOIN_GROUP
```

Value:

```
15
```

Join a multicast group.

When set, optval must point to an in6_addr_t.

Definition at line 165 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

IPV6_LEAVE_GROUP

```
#define IPV6_LEAVE_GROUP
```

Value:

```
16
```

Leave a multicast group.

When set, optval must point to an in6_addr_t.

Definition at line 168 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SO_EDFE_MODE

```
#define SO_EDFE_MODE
```

Value:

```
0xfb
```

Experimental: Enable Extended Directed Frame Exchange mode.

When set, optval must point to an uint32_t.

Definition at line 171 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SOCKET_EDFE_MODE

```
#define SOCKET_EDFE_MODE
```

Value:

```
SO_EDFE_MODE
```

Set the unicast hop limit for the socket. When set, optval must point to an int16_t.

-1 in the value means use the route default, otherwise it should be between 0 and 255.

Definition at line 172 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

Enumeration Documentation

sl_wisun_socket_event_mode_t

```
sl_wisun_socket_event_mode_t
```

Enumerations for socket event mode.

Enumerator	
SL_WISUN_SOCKET_EVENT_MODE_INDICATION	SL_WISUN_MSG_SOCKET_DATA_IND_ID is sent to the app with the packet contained in the indication.

SL_WISUN_SOCKET_EVENT_MODE_POLLING	SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID is sent to the app indicating the amount of data received recv() or recvfrom() should be invoked after indication reception to retrieve data This is the default socket event mode option.

Definition at line 189 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

socket_domain

> socket_domain

Supported address families.

Enumerator	
AF_INET6	IP version 6.

Definition at line 199 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

socket_type

> socket_type

Socket types.

Enumerator	
SOCK_STREAM	stream (connection) socket (TCP)
SOCK_DGRAM	datagram (connectionless) socket (UDP)
SOCK_RAW	raw socket

Definition at line 204 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

socket_protocol

> socket_protocol

IP protocols.

Enumerator	
IPPROTO_IP	Dummy protocol.
IPPROTO_ICMP	Internet Control Message Protocol.
IPPROTO_TCP	Transmission Control Protocol.
IPPROTO_UDP	User Datagram Protocol.

Definition at line 211 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

Typedef Documentation

socklen_t

> typedef uint32_t socklen_t

Socket address length type definition.

Definition at line 183 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sl_wisun_socket_id_t

```
typedef int32_t sl_wisun_socket_id_t
```

Socket id.

Definition at line 186 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sl_socket_domain_t

```
typedef enum socket_domain sl_socket_domain_t
```

Supported address families.

Definition at line 201 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sl_socket_type_t

```
typedef enum socket_type sl_socket_type_t
```

Socket types.

Definition at line 208 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sl_socket_protocol_t

```
typedef enum socket_protocol sl_socket_protocol_t
```

IP protocols.

Definition at line 216 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

in6_addr_t

```
typedef struct in6_addr in6_addr_t
```

IPv6 Internet address.

Definition at line 227 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sockaddr_in6_t

```
typedef struct sockaddr_in6 sockaddr_in6_t
```

IPv6 address format.

Definition at line 239 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

Variable Documentation

in6addr_any

```
const in6_addr_t in6addr_any
```

IPv6 wildcard address.

Definition at line 230 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

Function Documentation

socket

```
int32_t socket (int32_t domain, int32_t type, int32_t protocol)
```

Creates an endpoint for communication and returns an Id that refers to that endpoint.

Parameters

[in]	domain	Specifies a communication domain. It selects the protocol family which will be used for communication. Must be set to AF_INET6 (IPv6 network socket)
[in]	type	The communication semantics It can be: • SOCK_STREAM - TCP stream socket type • SOCK_DGRAM - UDP datagram socket type • SOCK_RAW - Raw Socket type (ICMP)
[in]	protocol	Specifies the particular protocol to be used. It can be: • IPPROTO_ICMP - Ping • IPPROTO_IP and IPPROTO_TCP - TCP stream sockets • IPPROTO_IP and IPPROTO_UDP - UDP datagram sockets

Returns

- The socket's id on success, (-1) on failure.

Definition at line 261 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

close

```
int32_t close (int32_t sockid)
```

Close a socket.

Parameters

[in]	sockid	socket id

Returns

- 0 on success, -1 on failure.

Close a socket and remove from the socket handler storage.

Definition at line 277 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

bind

```
int32_t bind (int32_t sockid, const struct sockaddr *addr, socklen_t addrlen)
```

Bind a name to a socket.

Parameters

[in]	sockid	socket id
[in]	addr	address structure ptr
[in]	addrlen	address structure size

Returns

- 0 on success, -1 on failure.

Assigns the address to the socket, referred to by the socket ID, as specified by addr. It is normally necessary to assign a local address using bind() before a SOCK_STREAM socket may receive connections.

Definition at line 291 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

send

```
int32_t send (int32_t sockid, const void *buff, uint32_t len, int32_t flags)
```

Send a message on a socket.

Parameters

[in]	sockid	socket descriptor.
[in]	buff	pointer to data buffer to send.
[in]	len	length of data buffer to send.
[in]	flags	flags to select send options. Ignored in our implementation.

Returns

- The number of bytes sent on success, -1 if an error occurred.

Preferred with connection-oriented sockets (TCP).

Definition at line 304 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sendto

```
int32_t sendto (int32_t sockid, const void *buff, uint32_t len, int32_t flags, const struct sockaddr *dest_addr, socklen_t addr_len)
```

Send a message to a given address.

Parameters

[in]	sockid	socket descriptor.
[in]	buff	pointer to data buffer to send.
[in]	len	length of data buffer to send.
[in]	flags	flags to select send options. Ignored in our implementation.
[in]	dest_addr	pointer to destination address buffer; Required for datagram sockets.
[in]	addr_len	length of destination address buffer.

Returns

- The number of bytes sent on success, -1 if an error occurred.

Preferred in datagram mode (UDP).

Definition at line `319` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h`

recvfrom

```
int32_t recvfrom (int32_t sockid, void *buf, uint32_t len, int32_t flags, struct sockaddr *src_addr, socklen_t *addrlen)
```

Receive messages from a socket.

Parameters

[in]	sockid	descriptor of socket to receive the message from.
[out]	buf	pointer to destination data buffer.
[in]	len	length of destination data buffer.
[in]	flags	flags to select type of message reception. Ignored in our implementation.
[in]	src_addr	pointer to a sockaddr structure in which the sending address is to be stored.
[in]	addrlen	length of the supplied sockaddr structure.

Returns

- The number of bytes received on success, -1 if an error occurred.

Receives data on a socket whether or not it is connection-oriented.

Definition at line `337` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h`

recv

```
int32_t recv (int32_t sockid, void *buf, uint32_t len, int32_t flags)
```

Receive a message from a socket.

Parameters

[in]	sockid	descriptor of socket to receive the message from.
[out]	buf	pointer to destination data buffer.
[in]	len	length of destination data buffer.
[in]	flags	flags to select type of message reception. Ignored in our implementation.

Returns

- The number of bytes received on success, -1 if an error occurred.

Should be used for connection-oriented protocol (TCP)

Definition at line 352 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

accept

```
int32_t accept (int32_t sockid, struct sockaddr *addr, socklen_t *addrlen)
```

Accept a connection on a socket.

Parameters

[in]	sockid	socket descriptor
[inout]	addr	A pointer to sockaddr structure filled in with the address of the peer (remote) socket. When addr is NULL, nothing is filled in; in this case, addrlen is not used, and should also be NULL.
[inout]	addrlen	The caller must initialize it to contain the size (in bytes) of the structure pointed to by addr. On return it will contain the actual size of the peer address.

Returns

- The socket id of the accepted socket on success, -1 if an error occurred.

Used with connection-based socket types (TCP). It extracts the first connection request on the queue of pending connections for the listening socket.

Definition at line 372 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

connect

```
int32_t connect (int32_t sockid, const struct sockaddr *addr, socklen_t addrlen)
```

Initiate a connection on a socket.

Parameters

[in]	sockid	socket descriptor.
[in]	addr	If the socket sockid is of type SOCK_DGRAM, addr is the address to which datagrams are sent by default and the only address from which datagrams are received. If the socket is of type SOCK_STREAM, this call attempts to make a connection to the socket that is bound to the address specified by addr.
[in]	addrlen	length of the supplied sockaddr structure.

Returns

- 0 on connection or binding success, -1 if an error occurred.

Connects the socket referred to by the sockid to the address specified by address.

Definition at line 388 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

listen

```
int32_t listen (int32_t sockid, int32_t backlog)
```

Listen for connections on a socket.

Parameters

[in]	sockid	socket id
[in]	backlog	Argument defines the maximum length to which the queue of pending connections for sockid may grow. Not implemented for Wi-SUN, the connection queue size is always 1

Returns

- 0 on success, -1 if an error occurred.

Marks the socket referred to by sockid as a passive socket, that is, as a socket that will be used to accept incoming connection requests using accept.

Definition at line `402` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h`

setsockopt

> int32_t setsockopt (int32_t sockid, int32_t level, int32_t optname, const void *optval, socklen_t optlen)

Set socket option designated by optname at a given protocol level to the value pointed by optval.

Parameters

[in]	sockid	socket ID
[in]	level	protocol level at which the option resides. it could be: • SOL_SOCKET • IPPROTO_IPV6 • SOL_APPLICATION

[in]	optname	option name. it could be:
		• for SOL_SOCKET level:
		  ◦ SO_RCVBUF
		  ◦ SO_SNDBUF
		  ◦ SO_SNDLOWAT
		• for IPPROTO_IPV6 level:
		  ◦ IPV6_UNICAST_HOPS
		  ◦ IPV6_MULTICAST_HOPS
		  ◦ IPV6_JOIN_GROUP
		  ◦ IPV6_LEAVE_GROUP
		  ◦ SO_EDFE_MODE
		• for SOL_APPLICATION:
		  ◦ SO_EVENT_MODE
		  ◦ SO_NONBLOCK

[in]	optval	Pointer to the socket option new value. The type of variable pointed by optval depends level and optname values.
[in]	optlen	Must be the size of the symbol pointed by optval.

Returns

- 0 on success, -1 if an error occurred.

This function can set socket properties.

Definition at line `436` of file `/mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h`

getsockopt

> int32_t getsockopt (int32_t sockid, int32_t level, int32_t optname, void *optval, socklen_t *optlen)

Get socket option.

Parameters

[in]	sockid	socket descriptor.
[in]	level	socket protocol level.

[in]	optname	Option name. Supported options:
		• for SOL_SOCKET level:
		  ◦ SO_RCVBUF
		  ◦ SO_SNDBUF
		  ◦ SO_SNDLOWAT
		• for IPPROTO_IPV6 level:
		  ◦ IPV6_UNICAST_HOPS
		  ◦ IPV6_MULTICAST_HOPS

[out]	optval	option value structure pointer.
[in]	optlen	size of the option value structure.

Returns

- 0 on success, -1 if an error occurred.

The function gets socket option by optname, and copies option data to optval ptr.

Definition at line 459 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

htonl

> __STATIC_INLINE uint32_t htonl (uint32_t hostlong)

Convert the long host byte order to network order.

Parameters

[in]	hostlong	Long host integer

Returns

- Long network integer

This function converts the unsigned integer hostlong from host byte order to network byte order. For Wi-SUN, the conversion is not needed. Dummy implementation

Definition at line 471 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

htons

> __STATIC_INLINE uint16_t htons (uint16_t hostshort)

Convert the short host byte order to network order.

Parameters

[in]	hostshort	Short host integer

Returns

- Short network integer

This function converts the unsigned short integer hostshort from host byte order to network byte order. For Wi-SUN, the conversion is not needed. Dummy implementation

Definition at line 485 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

ntohl

```
__STATIC_INLINE uint32_t ntohl (uint32_t netlong)
```

Convert the long network byte order to host byte order.

Parameters

[in]	netlong	Long network integer

Returns

- Long host integer

This function converts the unsigned integer netlong from network byte order to host byte order. For Wi-SUN, the conversion is not needed. Dummy implementation

Definition at line 499 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

ntohs

```
__STATIC_INLINE uint16_t ntohs (uint16_t netshort)
```

Convert the short network byte order to host byte order.

Parameters

[in]	netshort	

Returns

- Short host integer

This function converts the unsigned short integer netshort from the network byte order to host byte order.For Wi-SUN, the conversion is not needed. Dummy implementation.

Definition at line 514 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

inet_pton

```
int32_t inet_pton (int32_t af, const char *src, void *dst)
```

Convert the IPv4 and IPv6 addresses from text to binary form.

Parameters

[in]	af	The address family. Only AF_INET6 is supported by our implementation.
[in]	src	Source string

[out]	dst	Destination address pointer

Returns

- 1 on succes, -1 if an error occurred. (POSIX described the 0 value too)

This function converts the character string src into a network address structure in the af address family, then copies the network address structure to dst. AF_INET (IPv4) case not supported.

Definition at line 531 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

inet_ntop

```
const char * inet_ntop (int32_t af, const void *src, char *dst, socklen_t size)
```

Convert IPv6 addresses from binary to text form.

Parameters

[in]	af	- Address family. • Only AF_INET6 is supported
[in]	src	Source address in byte form
[out]	dst	Destination buffer ptr
[in]	size	Size of the destination buffer.

Returns

- It returns a non-null pointer to dst. NULL if an error occurred.

Converts the network address structure src in the af address family into a character string. The resulting string is copied to the buffer pointed to by dst, which must be a non-null pointer. The caller specifies the number of bytes available in this buffer in the argument size. AF_INET (IPv4) case not supported.

Definition at line 549 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

Macro Definition Documentation

APP_LEVEL_SOCKET

```
#define APP_LEVEL_SOCKET
```

Value:

```
SOL_APPLICATION
```

Definition at line 67 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

IPV6_ADDR_SIZE

```
#define IPV6_ADDR_SIZE
```

Value:

```
16
```

Size of an IPv6 address.

Definition at line 176 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

SOCK_NONBLOCK

```
#define SOCK_NONBLOCK
```

Value:

```
0X00010000
```

When bitwise ored with socket's type, it sets the O_NONBLOCK status flag on the opened socket file description.

Definition at line 180 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sockaddr

Socket address.

Public Attributes

uint16_t	sa_family address family, AF_XXXX
uint8_t	sa_data 26 bytes of protocol address (IPv6)

Public Attribute Documentation

sa_family

```
uint16_t sockaddr::sa_family
```

address family, AF_XXXX

Definition at line 220 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sa_data

```
uint8_t sockaddr::sa_data[26]
```

26 bytes of protocol address (IPv6)

Definition at line 221 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

in6_addr

IPv6 Internet address.

Public Attributes

uint8_t	address IPv6 address.

Public Attribute Documentation

address

```
uint8_t in6_addr::address[IPV6_ADDR_SIZE]
```

IPv6 address.

Definition at line 226 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sockaddr_in6

IPv6 address format.

Public Attributes

uint16_t	sin6_family AF_INET6.
uint16_t	sin6_port Transport layer port.
uint32_t	sin6_flowinfo IPv6 flow information.
in6_addr_t	sin6_addr IPv6 address.
uint32_t	sin6_scope_id Scope ID.

Public Attribute Documentation

sin6_family

```
uint16_t sockaddr_in6::sin6_family
```

AF_INET6.

Definition at line 234 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sin6_port

```
uint16_t sockaddr_in6::sin6_port
```

Transport layer port.

Definition at line 235 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sin6_flowinfo

```
uint32_t sockaddr_in6::sin6_flowinfo
```

IPv6 flow information.

Definition at line 236 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sin6_addr

> in6_addr_t sockaddr_in6::sin6_addr

IPv6 address.

Definition at line 237 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

sin6_scope_id

> uint32_t sockaddr_in6::sin6_scope_id

Scope ID.

Definition at line 238 of file /mnt/raid/workspaces/ws.Mae4JfP9d/overlay/gsdk/protocol/wisun/stack/inc/socket/socket.h

Wi-SUN Stack Release Note

Wi-SUN Stack Release Note

Release 1.8

(release date 2023-12-13)

New Features and Improvements

- added a new API sl_wisun_get_stack_version() that returns the stack version.
- updated sl_wisun_join() to support the customization of PHY configurations. Extended sl_wisun_phy_config_type_t and sl_wisun_phy_config_t to allow the customization of OFDM, FSK and O-QPSK entries.
- added support for LFN Timing Offset (LTO). Avoid LFN broadcast and unicast overlaps.
- added support for LFN multicast reception.
- added support for SUN DSSS-OQPSK.
- added support for blocking sockets.
- added support for the new Indian PHY configurations.

Bug Fixes

- fixed an error causing an assert when a device connects to network and then later re-join with a different device type.
- fixed an error causing an LFN parent to hard fault on an LFN disconnection.
- fixed an invalid variable initialization that could cause a device frame counter to be set to 0 when using IAR.
- fixed an invalid configuration preventing LFN to connect when using the Balanced or Eco modes.
- fixed an invalid initialization in MPL that was causing a multicast packet to be considered as old after a router re-connection.
- fixed an invalid time synchronization between an LFN and his FFN parent. It was causing significant drifts making downstream communications instable after a while.
- the Join Metric IE was not forwarded when the join state 1 was skipped.
- fixed an error causing a LFN to be out-of-sync when no packet is exchanged for more than 1h10 (uint32 max us)
- fixed an error causing MAC retries to be skipped on rare occasions.
- fixed an error causing routers to stay on the wrong channel after performing asynchronous transmissions.
- LFN are now send a Neighbor Solicitation with an EARO with a zero lifetime when disconnecting.
- Wi-SUN OUI was coded with the wrong byte ordering.
- maintained and restored the DHCP Identity Association ID (IAID) across reboots.
- that stack performed a CSMA/CA on asynchronous frame transmissions. This behavior is specifically forbidden in the FAN TPS.
- the stack occasionally tried to send a Neighbor Advertisement (NA) as a reply to a received Neighbor Solicitation (NS) used for Neighbor Unreachability Detection (NUD). This behavior is specifically forbidden in the FAN TPS.

Release 1.7.1

(release date 2023-10-10)

Bug Fixes

- fixed an issue causing LFN LGTK and frame counter recovery from NVM after a reboot to be skipped.
- fixed an invalid memory access when receiving a multicast packet with a full neighbor table.

- fixed LFN parent timeout. It was still partially relying on the FFN timeout mechanism.
- fixed LFN address renewal.
- fixed an invalid memory access in the RCP. It could either trigger an assert “ref_counter <= 0" or call free() on an invalid memory section.
- fixed an issue causing a mis-calculation of the ETX.
- fixed an issue causing an invalid memory access when starting the SoC border router with an invalid PHY configuration
- fixed an interoperability issue when using JP regulatory domain.
- fixed a race between FHSS and ND causing an invalid memory access when disconnecting itself or an LFN child.
- fixed FFN LGTK acquisition. FFN were performing a full 4-way handshake instead of a shorter 2-way handshake.
- fixed an issue causing DHCP lease renewal to happen too often.
- fixed an issue causing an invalid memory access when starting the stack without any certificate.
- added missing Node Role KDE. Routers not supporting LFN parenting were missing the information element and were considered as FAN1.0 devices by Silicon Labs border routers.

Release 1.7

(release date 2023-06-07)

New Features and Improvements

- LFN devices are now able to enter in Energy Mode 2 (EM2) if the application allows it.
- added support for PAN-Wide IE.
- adapted EDFE support for FAN 1.1.
- adapted MPL support for FAN 1.1.
- major refactoring of the stack internal timekeeping.

Bug Fixes

- frame counters were not increased on retries.
- fixed an issue causing UDP packets to be silently dropped. This was caused by an inappropriate management of fragmented packets.
- fixed an issue causing a suitable neighbor to be refused as a potential parent. Data packets missing a US-IE were refused while they should have been accepted. The stack was not keeping track of the US-IE received during the authentication process. This was causing interoperability issues with Nissin System routers.
- fixed an improper stack initialization when used outside of a project generated with SLC.
- fixed an issue causing a memory corruption when disconnecting an LFN.

Release 1.6

(release date 2023-06-07)

New Features and Improvements

- added a new API sl_wisun_set_lfn_parameters() that configures all the LFN-specific settings.
- added a new API sl_wisun_set_lfn_support() that sets the maximum number of LFNs that can be connected to a single FFN.
- added a new API sl_wisun_set_mode_switch() that supersedes sl_wisun_set_mode_switch().The old API is still availa-ble but it is recommended to move to the more recent one.
- added a new API sl_wisun_set_pti_state() that enable the Packet Trace Interface (PTI). For more information about the PTI in the context of Wi-SUN, refer to Wi-SUN’s Getting Started section on docs.silabs.com.
- implemented LFN LGTK rotation
- added support for a non-standard OFDM 64QAM PHY.
- added support for EFR32FG28
- optimized the reconnection of routers to an existing network: if configured to do so, the routers will now try to skip the scanning and authenticated step of the joining procedure.

Bug Fixes

- fixed an issue that could cause acknowlges to be sent to the wrong channel.
- fixed several issues that could cause an assert after a call to sl_wisun_disconnect().
- fixed an issue that could cause a mutex to be kept for an undefined period of time. It was causing devices to be indefinitely stalled.

Release 1.5.2

(release date 2023-04-19)

- fixed an RNG error that could happen on EFR32xG12.
- fixed an error causing LPA to be missed on PHY configurations using a lot of channels.
- fixed an error causing a segmentation fault on congested networks.

Release 1.5.1

(release date 2023-03-08)

- reduced the NS(EARO) retry period during the first connection. It reduces the time to connect in case of retransmission and improves the process's reliability.
- unencrypted LPC are not accepted by LFN anymore.
- fixed FEC support in Linux Border Router RCP.

Release 1.5.0

(release date 2023-02-01)

- added a new SL_WISUN_SOCKET_OPTION_SEND_BUFFER_LIMIT socket option to configure the transmission buffer length.
- added a new SL_WISUN_PHY_CONFIG_IDS option in sl_wisun_join(). It allows the selection of a specific entry in the radio configuration.
- added LFN parent synchronization and time-out detection.
- added LFN EAPOL accelerated listening schedule.

Bug Fixes

- fixed a regression in the connection time.

Release 1.4.0

(release date 2022-12-14)

New Features and Improvements

- added minimal support for FAN1.1 LFN (Limited Functional Node). LFN devices are able to connect and communicate but are not using any of the EFR32 energy management mode. As such, they are not optimized for battery powered devices and should only be used for evaluation and experimentation.
  - added a new API sl_wisun_set_device_type()that configures the role of device. It can be either a router (FFN – Full Functional Node) or an end node (LFN)
  - added a new set of libraries supporting both LFN and FFN device types. Those libraries are used when the new "Stack LFN Support" plugin is installed.
- added support for FAN 1.1 PHY mode switch.
  - added a new API sl_wisun_set_mode_switch() that indicates if the device can mode switch with a given neighbor.

    - added new APIs sl_wisun_set_pom_ie()and sl_wisun_get_pom_ie() that write or read the content of the POM-IE (PHY Operating Mode Information Element). It contains the list of the PHY operating mode a node is willing to use for communication.
- added a new API sl_wisun_join() that triggers a new connection. It can be used either with the old (1V08 – regulatory domain, operating class, operating mode) or the new (1VA8 – regulatory domain, channel plan id, PHY mode if) nomenclature. That new API is meant to replace sl_wisun_connect().
- added a new API sl_wisun_set_connection_parameters() that extends the set of the configuration parameters. Used in pair with sl_wisun_join(), it replaces sl_wisun_connect() network size parameter.
- added support for FSK FEC. FEC can be enable either by setting fec field of sl_wisun_phy_config_t structure or by using a PHY mode id that explicity enables FEC.
- added support of EFR32FG25. It supports all FAN1.1 OFDM modulation schemes and all FAN1.0 FSK configurations.
- added support of EFF01

Deprecated Items

- sl_wisun_connect() is replaced by sl_wisun_join().
- sl_wisun_set_channel_plan() is replaced by sl_wisun_join().
- sl_wisun_set_network_size() is replaced by sl_wisun_set_connection_parameters()

Release 1.3.2

(release date 2022-28-09)

Bug Fixes

- fixed an invalid Path Control field configuration in RPL DAO packet.

Release 1.3.1

(release date 2022-06-08)

Net Features and Improvements

- when ARIB radio regulation is selected, the stack refuses all EDFE initialization requests sending an EDFE final frame.

Bug Fixes

- fixed PAN Advert and PAN Config Trickle timer configuration. Inconsistent transmissions were not correctly managed. It could lead to suboptimal behavior in dense areas of a network.
- fixed a performance issue that was causing Linux Border Router RCP to become unreachable when running throughput tests with high-speed data rates. That issue was fixed by using DMA to collect UART data.
- fixed an initialization issue that was causing PAN Advert and PAN Config asynchronous transmission requests to be dropped. That issue was most likely to happen with TEST and SMALL network size configuration and could cause connection durations to be significantly longer.

Release 1.3.0

(release date 2022-06-08)

New Features and Improvements

- most of the stack crypto operations are now made through ARM PSA Crypo API.

- added a new API sl_wisun_set_device_private_key_id() that indicates which PSA Crypto key handler contains the device private key and has to be used by the stack. It is the application responsibility to create the key.
- added a new API sl_wisun_set_regulation() that configures the regional regulation the stack must comply to. Refer to UG495 for more information about regional regulation in Wi-SUN Stack.
- added a new event SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID that is fired when the transmission budget is cross-ing one of the transmission budget threshold. The transmission budget is defined by the regional regulation. Refer to UG495 for more information about regional regulation in Wi-SUN Stack.
- added a new API sl_wisun_set_regulation_tx_thresholds() that configures the threshold used to fire SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID event.
- added a new API sl_wisun_set_advert_fragment_duration() that configures the asynchronous transmission frag-ment duration. It can be used to reduce the impact on the latency of long advertisement periods.
- added a new API sl_wisun_set_unicast_tx_mode() that enables a high-reliability transmission mechanism for unicast communication. It trades off unicast communication reliability for latency.
- sl_wisun_set_channel_mask() channel filter is now applied to asynchronous transmissions and unicast listening schedule. The function was renamed sl_wisun_set_allowed_channel_mask() to make it more self-explanatory.

Bug Fixes

- Fixed an error that was causing radio calibrations to be skipped during Wi-SUN Stack initialization.

Release 1.2.3

(release date 2022-03-09)

Bug Fixes

- Applied the PA configuration set in the application. It was previously ignored and the same configuration was always used.

Release 1.2.2

(release date 2022-02-21)

Bug Fixes

- Fixed a recurrence of the error that could cause the stack to assert on a RAIL_StartCcaCsmaTx when trying to connect to a network that cannot be reached.

Release 1.2.1

(release date 2022-01-26)

Bug Fixes

- Fixed an error that could cause the stack to assert on a RAIL_StartCcaCsmaTx when trying to connect to a network that cannot be reached.

Release 1.2

(release date 2021-10-13)

New Features and Improvements

- added release quality libraries. They provide the same Wi-SUN features but are not logging anything.
- added a new API sl_wisun_reset_statistics that resets all the counters read by calling sl_wisun_get_statistics.
- added new APIs sl_wisun_get_neighbor_count() and sl_wisun_get_neighbors() that indicate the neighbor count (parents and children) and their MAC address.
- added a new API sl_wisun_get_neighbor_info() that returns information about a neighbor.
- added a new API sl_wisun_set_unicast_settings() that configures the frequency hopping unicast dwell interval.
- added a new API sl_wisun_set_trace_level() and sl_wisun_set_trace_filter() that configure the stack traces.
- added support for mbedtls v3.0.
- stack flash footprint reduction.
- reworked Wi-SUN stack tasks priorities.
- fixed the unicast channel filtering.
- fixed an error causing the authentication waiting list to be broken.

Release 1.1.2

(release date 2021-10-13)

New Features and Improvements

- fixed an issue causing a hard-fault during a parent information update.

Release 1.1.1

(release date 2021-09-08)

New Features and Improvements

- fixed a drift in the frequency hopping mechanism that could lead to disconnections in quiet networks.

Release 1.1.0

(release date 2021-07-21)

New Features and Improvements

- fixed an issue causing the event SL_WISUN_MSG_CONNECTED_IND_ID to be fired although no new connection was established. It was fired after each network update.
- fixed an issue causing connections to fail after an operating class update.
- fixed an issue causing US-IE configuration to be invalid when excluding channels.

Release 1.0.1

(release date 2021-06-16)

New Features and Improvements

- fixed an issue causing a parent to lose track of its child frequency hopping sequence. The child router was sending an incorrect IFSU misleading the parent router and forcing it to be one frequency hop interval late.

Release 1.0.0

(release date 2021-04-21)

New Features and Improvements

- fixed memory leaks during the disconnection and reset processes.
- added a new disconnected join state to sl_wisun_get_join_state API.
- miscellaneous API minor updates

Release 0.2.2

(release date 2021-04-21)

New Features and Improvements

- fixed a bug causing connection with network_size 'Test' to be slower than it used to be.
- added border router support for custom PHYs
- added a new API for clearing credential cache
- added a new API for MAC address management

Release 0.2.1

(release date 2021-04-07)

New Features and Improvements

- fixed a bug causing unicasts to be sent on the wrong channel
- updated a configuration that prevented multicast to be forwarded

Release 0.2.0

(release date 2021-01-25)

New Features and Improvements

- added support for CMSIS RTOS API2
- fixed UDP sockets corner cases.

Release 0.1.0

(release date 2020-12-10)

New Features and Improvements

- added support for Gecko SDK v3.1
- added support for operating mode 2a (100kbps) and 3 (150kbps)
- added new APIs sl_wisun_allow_mac_address() and sl_wisun_deny_mac_address() that enable MAC filtering.

Wi-SUN Sockets

Wi-SUN Sockets

The Wi-SUN socket API is loosely modeled after the well-known BSD socket API, also known as POSIX socket API. Each socket represents a handle for the local endpoint of a communication circuit.

> v1.8 deprecates Silicon Labs socket API. For a seamless compatibility, refer to the Silicon Labs socket API (deprecated) software component.

API Overview

BSD Function	Former Wi-SUN API Function	Description
socket()	sl_wisun_open_socket()	Creates an endpoint for communication and returns an Id that refers to that endpoint.
bind()	sl_wisun_bind_socket()	Bind a name to a socket.
listen()	sl_wisun_listen_on_socket()	Listen for connections on a socket.
connect()	sl_wisun_connect_socket()	Initiate a connection on a socket.
accept()	sl_wisun_accept_on_socket()	Accept a connection on a socket.
send()	sl_wisun_send_on_socket()	Send a message on a socket.
sendto()	sl_wisun_sendto_on_socket()	Send a message to a given address.
recv()	sl_wisun_receive_on_socket()	Receive a message from a socket.
recvfrom()	Not supported	Receive messages from a socket.
close()	sl_wisun_close_socket()	
gethostbyname()	Not supported	
gethostbyaddr()	Not supported	
select()	Not supported	
poll()	Not supported	
getsockopt()	sl_wisun_get_socket_option()	Get socket option.
setsockopt()	sl_wisun_set_socket_option()	Set socket option designated by optname at a given protocol level to the value pointed by optval.

Socket Life Cycle

In general, a socket is created by explicitly calling socket() and must be explicitly closed by calling `close()`. This both closes the communication circuit and frees any resources allocated to the socket.

In certain protocols, the communication circuit may be closed by the remote peer. This is indicated with a SL_WISUN_MSG_SOCKET_CLOSING_IND_ID event. Although the socket connection is closed, the system resources will not be freed until the application calls `close()`.

Exchanging Data

Certain protocols require a socket in a connected state, meaning the underlying protocol must first establish a session with the remote peer before data can be exchanged. It is also optionally possible on certain connectionless protocols. In either case, connecting a socket creates a direct communication link with a remote peer, allowing the socket to be used only with that particular peer.

To switch a socket into connected state, use `connect()`. See the table below for protocol details.

Protocol	Unconnected	Connected
IPPROTO_UDP	MAY	MAY
IPPROTO_TCP	MUST NOT	MUST
IPPROTO_ICMP	MUST	MUST NOT

Transmit

When a socket is in unconnected state, `sendto()` is used to transmit data. The function requires the application to specify the remote peer address and port number on each call. When a socket is connected, `send()` is used instead. Because the socket is "locked" into a single remote peer, address and port number are not needed.

Both functions are asynchronous. A successful return code indicates the socket has buffered the provided data and the application may free the data resources. The application receives a SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID event when a part or all buffered data has been sent.

Receive

To manage received socket data, the application has indication mode and polling mode. All sockets default to polling mode. In this mode, the application will receive a SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID event on data reception. The received data is stored in the socket buffer until the application reads it using `recv()` or `recvfrom()`; Both `recv()` or `recvfrom()` could be used only on the latter mode (Polling). A socket that generated a SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID will never block. In indication mode, the received data is sent as a SL_WISUN_MSG_SOCKET_DATA_IND_ID event as soon as it is received. The application must either handle the data immediately or store it for later processing. Either way, the socket will not keep the data once the event handler returns.

Blocking operations

By default, socket operations block the RTOS task until its completion. Sockets can be configured as non-blocking or'ing SOCK_NONBLOCK flag to the type parameter of `socket()`, or using the socket option #O_NONBLOCK. Our implementation only supports one operation at a time.

Client

TCP Example

1. Open a TCP socket using `socket(AFINET_6, SOCK_STREAM, IPPROTO_TCP)`.
2. Initiate connection to the remote server using `connect()`.
3. Wait for SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID event.
4. ... Exchange data with the remote server ...
5. Close the socket using `close()`.

UDP Example

Open an UDP socket using `socket(AFINET_6, SOCK_DGRAM, IPPROTO_UDP)`.
2. ... Exchange data with the remote server ...
3. Close the socket using `close()`.

Server

TCP Example

1. Open a TCP socket using `socket(AFINET_6, SOCK_STREAM, IPPROTO_TCP)`.
2. Set the server socket port number using `bind()`.
3. Set the server socket into listening state using `listen()`.
4. Wait for SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID event.
5. Accept the client connection using `accept()` on the server socket.
6. ... Exchange data with the remote client using the newly created client socket ...
7. Close the client socket using `close()`.
8. Go to 4.

UDP Example

1. Open an UDP socket using `socket(AFINET_6, SOCK_DGRAM, IPPROTO_UDP)`.
2. Set the socket port number using `bind()`.
3. ... Exchange data with the remote client ...

Deprecated List

Deprecated List

- Global sl_wisun_accept_on_socket (sl_wisun_socket_id_t socket_id, sl_wisun_socket_id_t *remote_socket_id, sl_wisun_ip_address_t *remote_address, uint16_t *remote_port) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See accept() for a replacement.
- Global sl_wisun_bind_socket (sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *local_address, uint16_t local_port) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See bind() for a replacement.
- Global sl_wisun_close_socket (sl_wisun_socket_id_t socket_id) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See close() for a replacement.
- Global sl_wisun_connect_socket (sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *remote_address, uint16_t remote_port) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See connect() for a replacement.
- Global sl_wisun_get_socket_option (sl_wisun_socket_id_t socket_id, sl_wisun_socket_option_t option, sl_wisun_socket_option_data_t *option_data) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See getsockopt() for a replacement.
- Global sl_wisun_listen_on_socket (sl_wisun_socket_id_t socket_id) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See listen() for a replacement.
- Global sl_wisun_open_socket (sl_wisun_socket_protocol_t protocol, sl_wisun_socket_id_t *socket_id) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See socket() for a replacement.
- Global sl_wisun_receive_on_socket (sl_wisun_socket_id_t socket_id, sl_wisun_ip_address_t *remote_address, uint16_t *remote_port, uint16_t *data_length, uint8_t *data) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See recvfrom() for a replacement.
- Global sl_wisun_send_on_socket (sl_wisun_socket_id_t socket_id, uint16_t data_length, const uint8_t *data) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See send() for a replacement.
- Global sl_wisun_sendto_on_socket (sl_wisun_socket_id_t socket_id, const sl_wisun_ip_address_t *remote_address, uint16_t remote_port, uint16_t data_length, const uint8_t *data) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See sendto() for a replacement.
- Global sl_wisun_set_socket_option (sl_wisun_socket_id_t socket_id, sl_wisun_socket_option_t option, const sl_wisun_socket_option_data_t *option_data) SL_DEPRECATED_API_SDK_4_4
  This function will be removed in the future versions of the Wi-SUN stack. See setsockopt() for a replacement.
- Global sl_wisun_util_get_rf_settings (uint8_t *reg_domain, uint8_t *op_class, uint16_t *op_mode) SL_DEPRECATED_API_SDK_4_2
  This function will be removed in the future versions of the Wi-SUN stack. See sl_wisun_util_get_phy_config() for a replacement.

Overview

Wi-SUN PHY

The content in this section is related to Wi-SUN PHY and system aspects provided by the Radio Configurator and RAIL. This is particularly of interest for customers not using the Silicon Labs stack. It explains how to configure the Wi-SUN PHY using the Radio Configurator provided with Simplicity Studio and how to use this using RAILtest commands.

- **Wi-SUN on EFR32FG25: Getting Started with RAILtest**: Describes how to program EFR32FG25 devices to enable Wi-SUN OFDM modulation and how to enable Wi-SUN FSK modulation.
- **Wi-SUN Mode Switch on EFR32FG25 with RAILtest**: Explains how to program EFR32FG25 devices to enable the Wi-SUN Mode Switch feature.
- **Wi-SUN Concurrent Detection on EFR32FG25 with RAILtest**: Explains how to program EFR32FG25 devices to enable the Wi-SUN concurrent detection feature.

Getting Started with Wi-SUN PHY

Getting Started with Wi-SUN PHY Configuration

This section is a quick start guide to configuring Wi-SUN PHYs and features using the Radio Configurator and RAILtest. It describes:

- How to configure Wi-SUN PHYs using the Radio Configurator
- How to use Wi-SUN FSK or OFDM modulation through RAILtest
- Known issues

Proprietary is supported on all EFR32FG devices, although OFDM modulation is available only on EFR32FG25 devices. In this document, only the RAILtest example application is used.

How To Use Wi-SUN FAN 1.0 PHY

How to Use Wi-SUN FAN 1.0 PHY

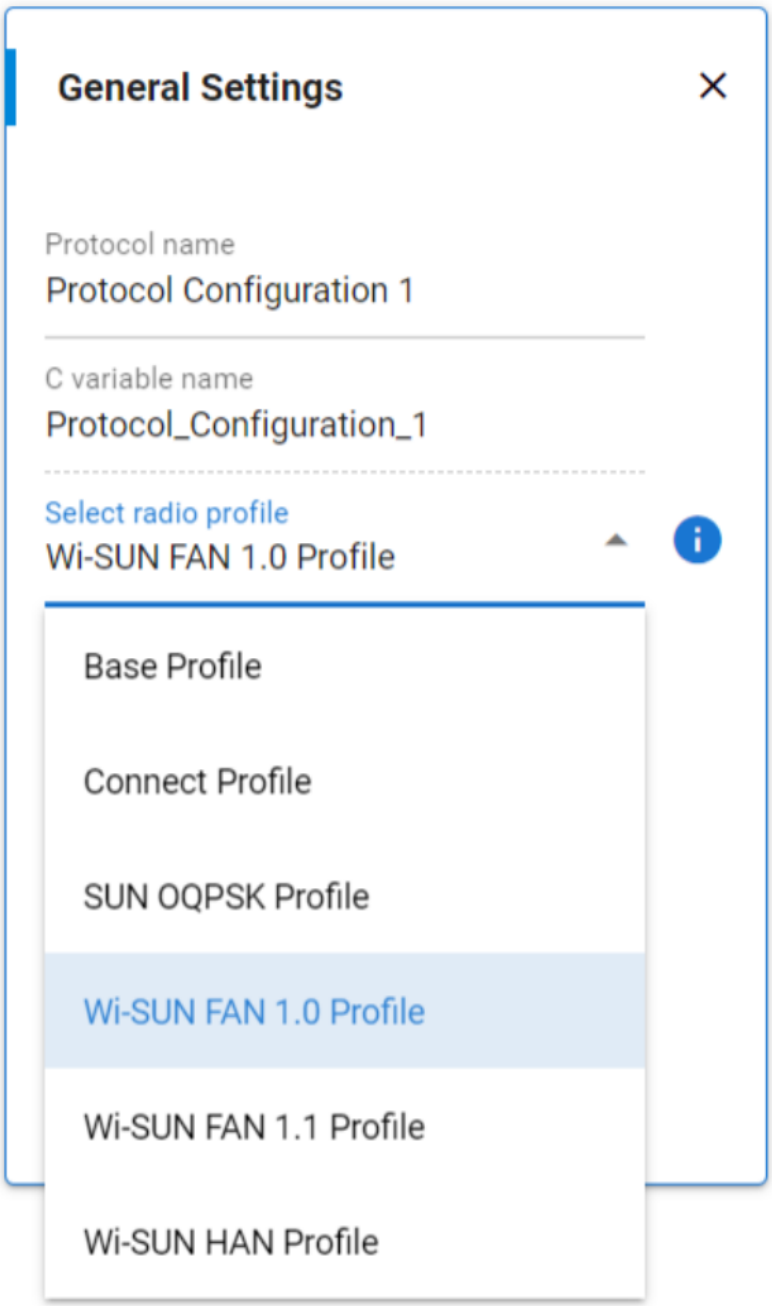
This section applies both to the EFR32FG25 and the EFR32MG12. It assumes that you have downloaded Simplicity Studio 5 and the Gecko SDK (GSDK) and are familiar with creating a project. If not, see the Getting Started section in the Simplicity Studio v5 User's Guide.

Set Up the Wi-SUN FAN 1.0 Configuration

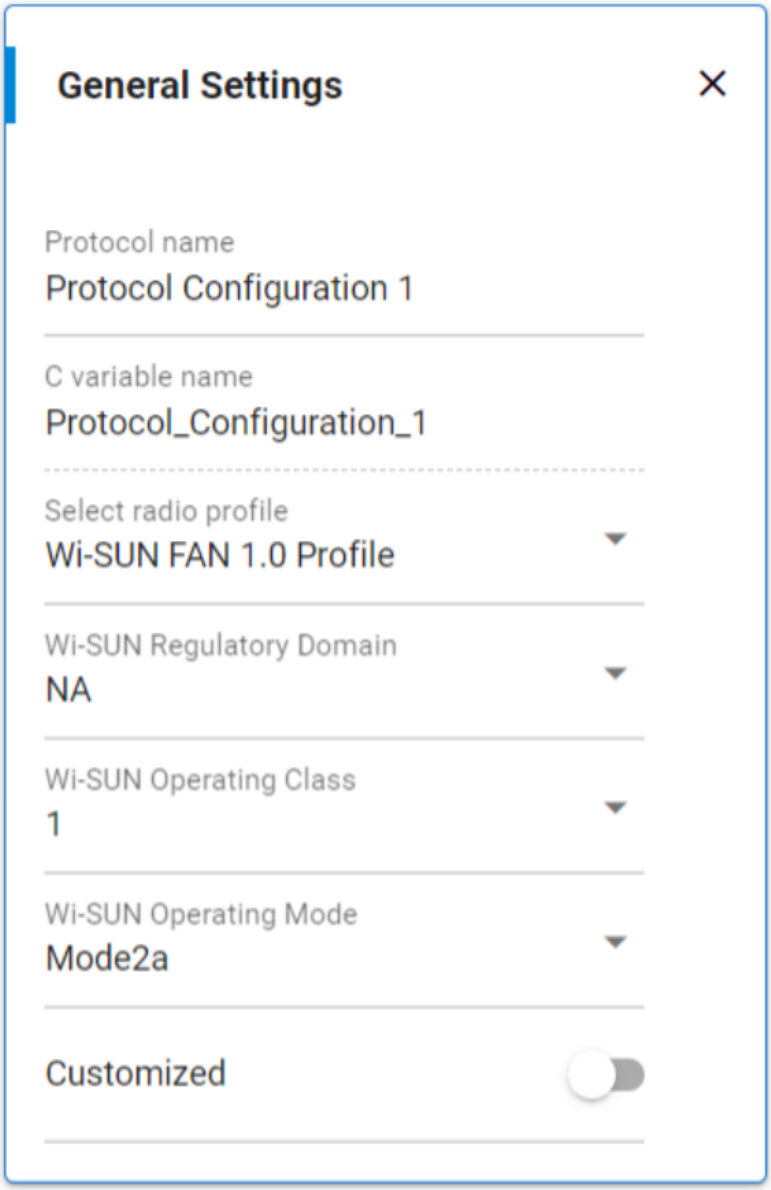
Configure the Radio by a Radio Configurator-Generated Config

Create a project based on the RAILTest example. Once a new RAILtest project is created, the Radio Configurator opens automatically.

1. Select the **Wi-SUN FAN 1.0 Profile.**



2. Radio Configurator provides all Wi-SUN FSK PHYs (defined in Wi-SUN PHY specification 2V00) as part of the Wi-SUN FAN 1.0 profile. Select the PHY based on the applicable **Wi-SUN Regulatory Domain** (EU, NA, JP, CN, ...) and **Wi-SUN Operating class and Modes** according to Wi-SUN FAN 1.0. The channel parameters are set automatically.



RF Frequency Planning in Multi-PHY Radio Configuration for Different Regions

If a Multi-PHY radio includes PHYs for different bands, you may need to select an RF Frequency planning band (see RF Frequency Planning).

How To Use Wi-SUN FAN 1.1 On EFR32FG25

How to Use Wi-SUN FAN 1.1 on EFR32FG25

This section assumes that you have downloaded Simplicity Studio 5 and the Gecko SDK (GSDK) and are familiar with creating a project. If not, see the Getting Started section in the Simplicity Studio v5 User's Guide.

Set Up the Wi-SUN FAN 1.1 Configuration

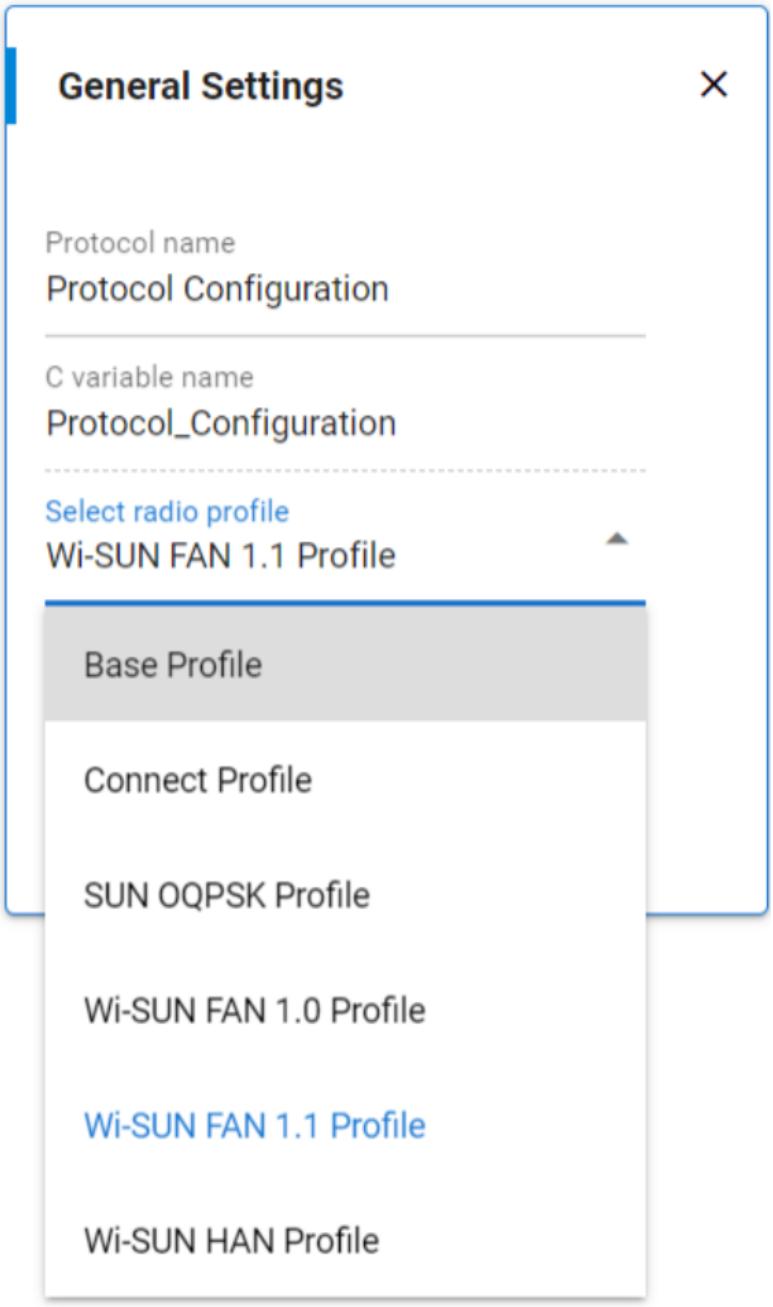
For the best performance with EFR32FG25, use the following radio board:

- BRD4270B revA04 for North America or Japan regions
- BRD4271A revA04 for Europe region

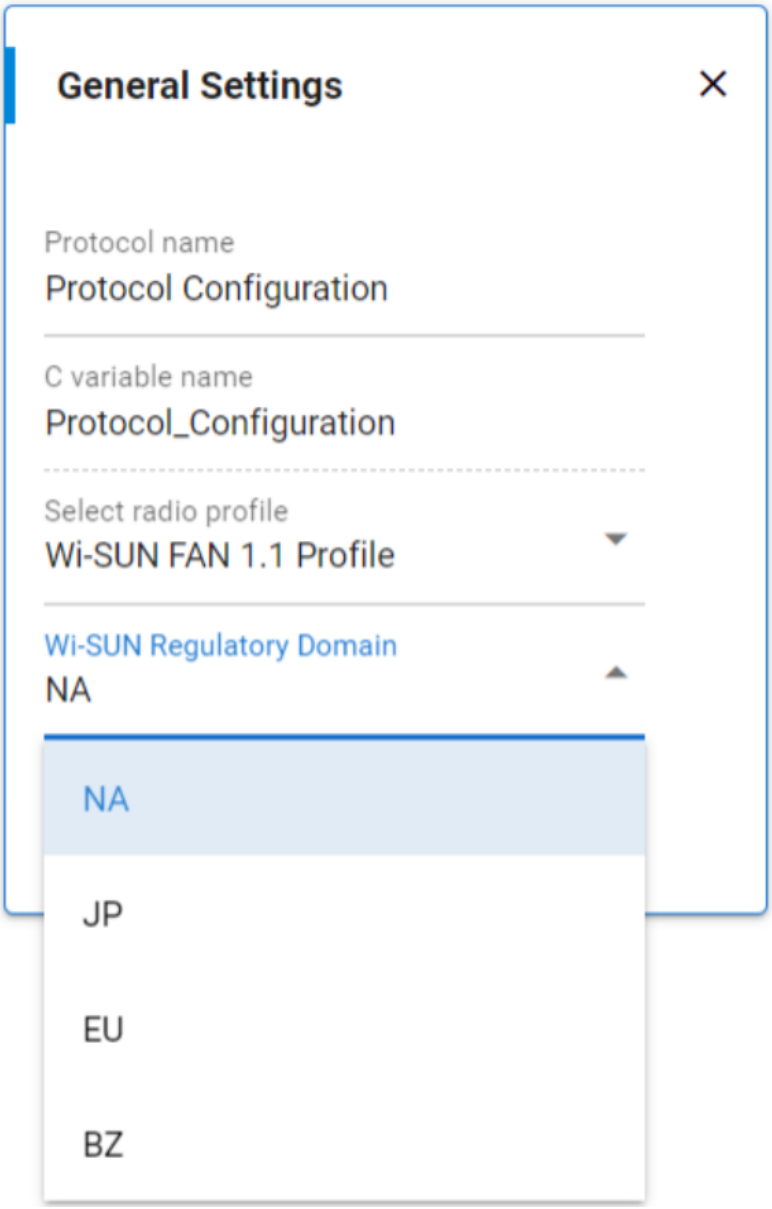
Configure the Radio by a Radio Configurator-Generated Config

Create a project based on the RAILTest example. Once a new RAILtest project is created, the Radio Configurator opens automatically. Here you can select the Wi-SUN FAN 1.1 Profile and browse among the available PHY definitions, including both Frequency Shift Keying (FSK) and Orthogonal frequency-division multiplexing (OFDM) PHYs, as defined in Wi-SUN PHY Technical specification amendment 2V00.

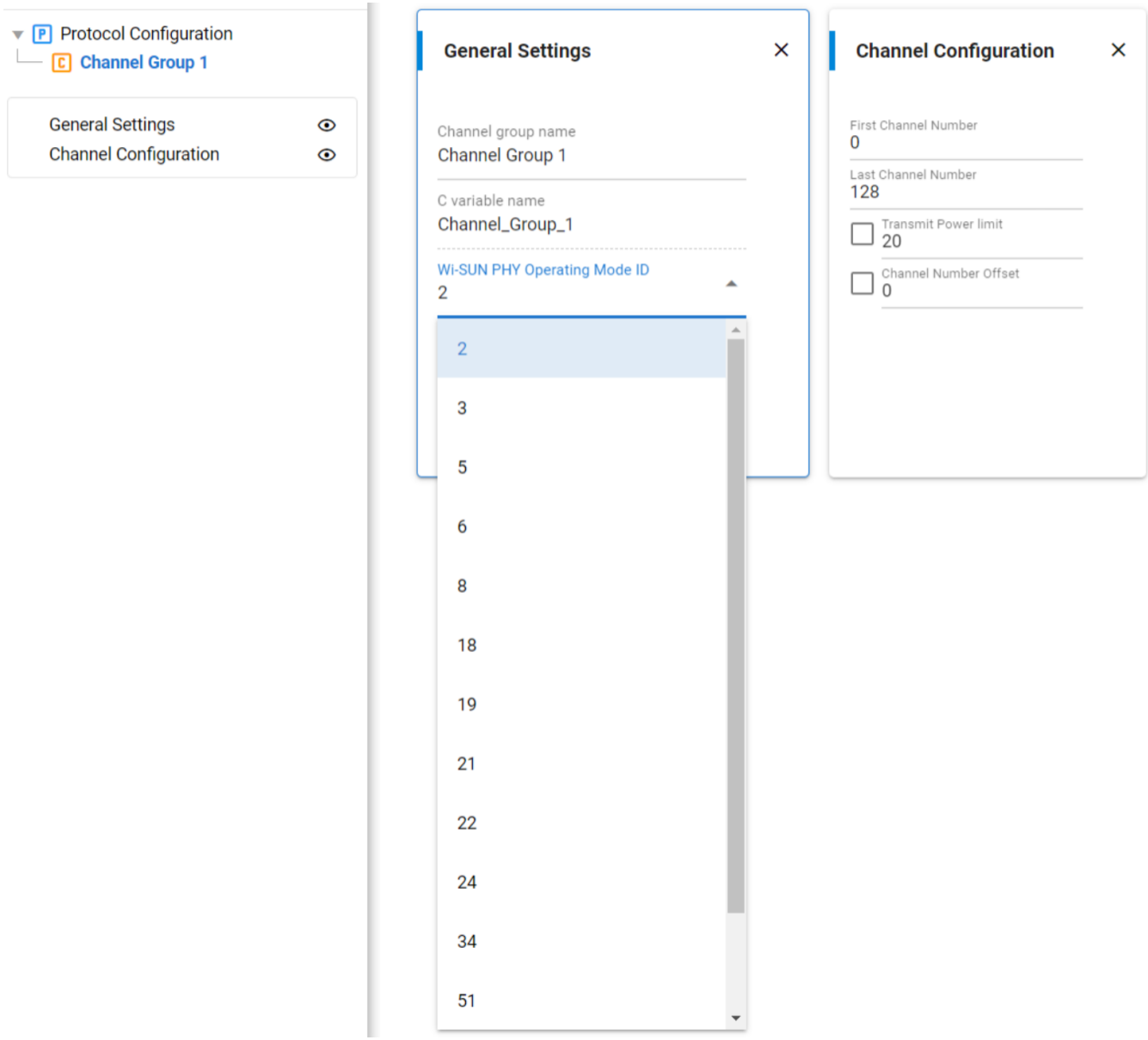
1. For the Protocol Configuration, the Radio Configurator provides all Wi-SUN FAN 1.1 PHYs as part of the Wi-SUN FAN 1.1 profile. Select the Wi-SUN FAN 1.1 profile. For more information about the Radio Configurator see the section in the Simplicity Studio v5 User's Guide .



2. Select the Wi-SUN Regulatory Domain (EU, NA, JP, BZ).

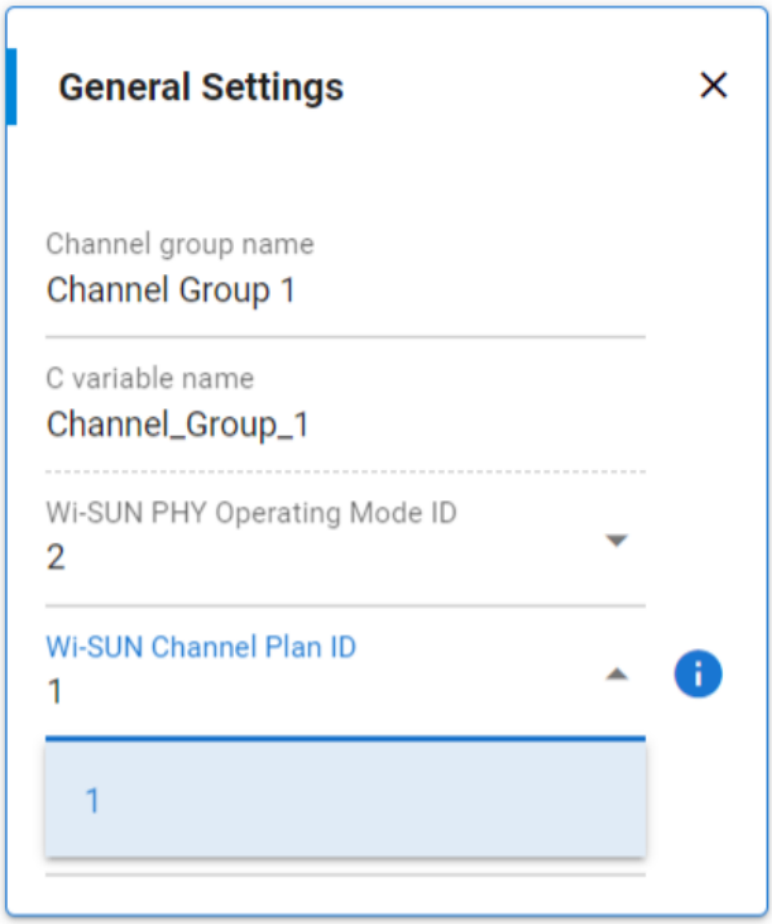


3. In **Channel Group**, select the **Wi-SUN PHY Operating Mode ID** according to Wi-SUN FAN 1.1 PhyModeID and the Wi-SUN Channel Plan ID.



Note: For OFDM PHYs, only one PhyModeID is listed using the lowest Modulation and Coding Scheme (MCS) allowed. This allows you to select the OFDM option (1, 2, 3 or 4). The MCS is selected through the RAILtest command set802154PHR as described in Set Up PHR and TX FIFO.

4. Select the **Wi-SUN Channel Plan ID,** which is available with the selected Wi-SUN PHY Operating Mode ID:



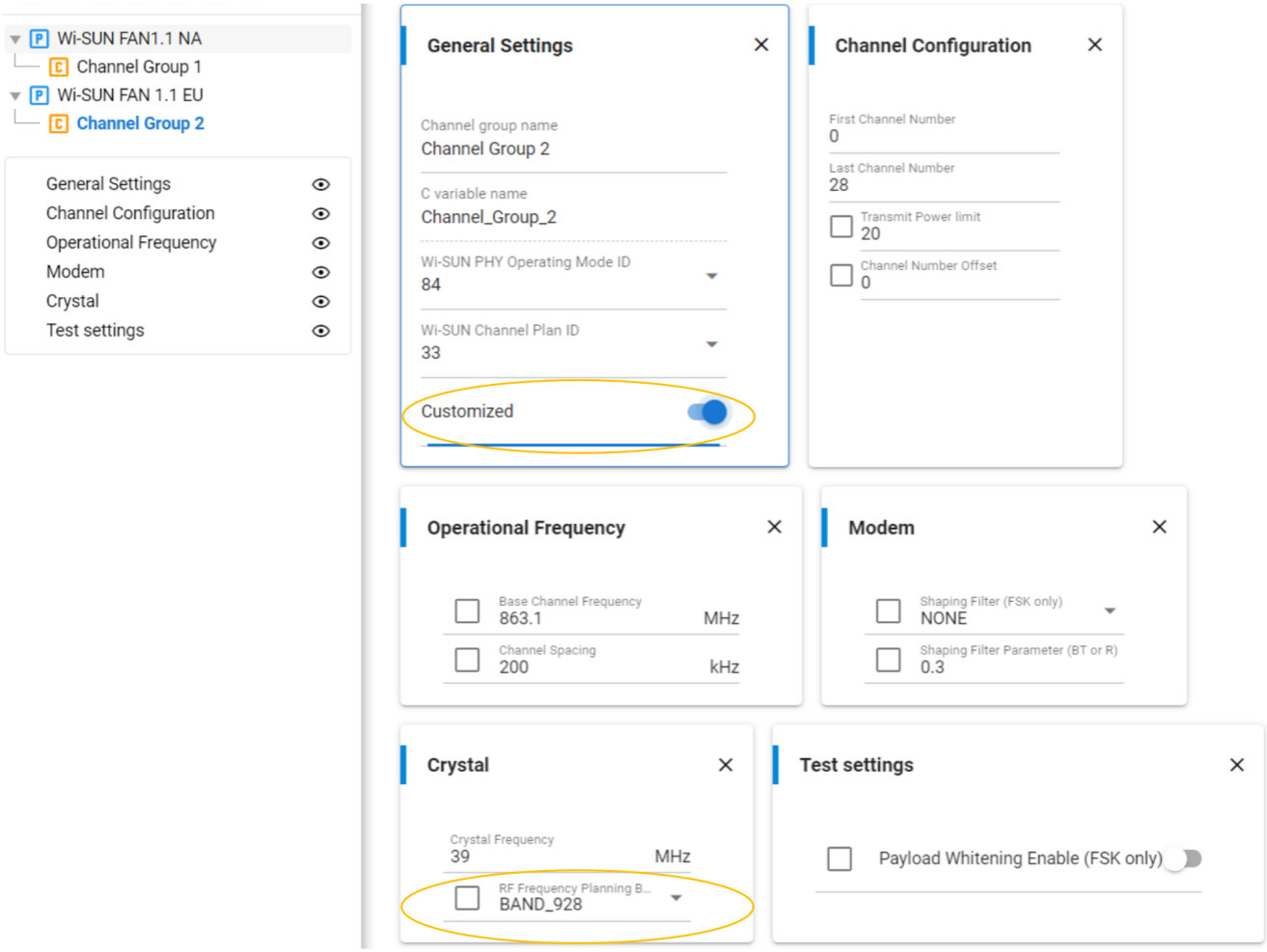
5. Save the *radio_settings.radioconf* file and build the RAILtest project.

RF Frequency Planning in Multi-PHY Radio Configuration for Different Regions

RF Frequency Planning (RFFPLL) settings are optimized to provide the best performance for a given band. By default, the Radio Configurator automatically selects these optimized settings when generating the PHYs. These settings are the same for the 9xx MHz band and the 868 MHz band with a 39 MHz crystal oscillator used in BRD4270B and BRD4271A, but they are different for the 470 MHz Band (not supported yet).

The RFFPLL is configured at boot time with the best settings depending on the board reference and its targeted band. Only the PHYs generated with the same RFFPLL configuration can then be loaded. Otherwise, RAIL returns the following error: RAIL_ASSERT_INVALID_RFFPLL_CONFIGURATION = 73. For instance, loading a 470 MHz PHY on a BRD4270B board would fail and return that error.

To select the desired RF Frequency planning setting, turn on **Customized** and select the band, for example BAND_928 as shown.



Note: If all PHYs are in the same band, Silicon Labs does not recommend customizing the RF Frequency Planning settings.

The recommended usage for the RF Frequency Planning enumerate is as described below:

- 'BAND_928': for channels from 928MHz up to 960MHz
- 'BAND_9xx': for channels from 902MHz up to 928MHz
- 'BAND_896': for channels from 896MHz up to 901MHz
- 'BAND_863': for channels from 863MHz up to 870MHz
- 'BAND_780': for channels from 779MHz up to 787MHz
- 'BAND_470': for channels from 470MHz up to 510MHz
- 'BAND_450': for channels below 470MHz

Wi-SUN Configuration With RAILtest For EFR32FG25

Wi-SUN Configuration with RAILtest for EFR32FG25

Wi-SUN FSK Configuration

Set Up PHR and TX FIFO

Wi-SUN FSK PHYs are configured to variable length packet encoding. Also, other configurable parameters are stored in the PHR field according to the standard. It might be difficult to set it up correctly. Therefore, a helper function – the set802154PHR CLI command – has been implemented in the RAILtest project, which simplifies configuration of a valid PHR field and the payload.

Using the RAILtest example, the transmitted packet length should be defined first with the `setTxLength` CLI command, including the PHR's size (2 bytes). This command sets up the Tx FIFO's length and allocates memory for it.

The `set802154PHR` CLI command for the FSK case sets the FCS type, Whitening bit, and Length fields of the PHR:

1. The first argument selects the PHR format (1, 2, 4 bytes). For FSK, this should be always set to 1, which selects 2 bytes.
2. The second argument sets the FCS type (it should be 0 for Wi-SUN FAN for 4 bytes FCS and 1 for ECHONET/HAN).
3. The third argument specifies the payload whitening (0 for payload whitening disabled and 1 for payload whitening enabled). The payload whitening is mandatory for Wi-SUN, but some Wi-SUN conformance tests could require disabling it. The whitening mode must also be set in the *radio_setting.radioconf*.

See an example use case in A Complete FSK Configuration Example.

The Frame Length field of the PHR is configured according to the length that previously was configured by the setTxLength command. The payload will be padded by autogenerated data.

Warning: `setTxLength` must always precede the `set802154PHR` command when the Tx packet's length is changed.

Note: To comply with IEEE 802.15.4, the `set802154PHR` command should be used according to the configured PHY in the Radio Configurator. The command only changes the PHR field in the Tx frames. No FSK modem configurations are made through this command.

Configure PA for FSK

The FSK PA (RAIL_TX_POWER_MODE_SUBGIG_POWERSETTING_TABLE) is configured by the `RAIL_ConfigTxPower()` RAIL API. This function is wrapped by the `setPowerConfig` CLI command.

A Complete FSK Configuration Example

In this section, a complete example is presented of how to transmit FSK-modulated packets with the EFR32FG25 radio using the RAILtest application. It assumes the EFR32FG25 is flashed with an image with a Wi-SUN FSK configuration with whitening enabled set with the Radio Configurator.

The following CLI commands are used:

- Set up the radio to use the FSK PA (only needed when also using OFDM PA).
- Set the desired PA level (check the maximum level depending on the radio board).
- Set the transmitted packet's length to 250 (= framelength field in the PHR of 252 – 4-byte FCS + 2 bytes of PHR).
- Configure the PHR.
- Transmit 10 packets.

```
> rx 0
{{(rx)}{Rx:Disabled}{Idle:Enabled}{Time:3295762881}}
> setpowerconfig RAIL_TX_POWER_MODE_SUBGIG_POWERSETTING_TABLE 3600 10
{{(setpowerconfig)}{success:true}{mode:RAIL_TX_POWER_MODE_SUBGIG_POWERSETTING_TABLE}{modeIndex:0}{voltage:3600}{rampTime:7}}
> setpower 140
{{(setpower)}{powerLevel:72}{power:140}}
> setchannel 0
{{(setchannel)}{channel:0}}
> rx 1
{{(rx)}{Rx:Enabled}{Idle:Disabled}{Time: 329573470724420}}
> setTxLength 250
{{(setTxLength)}{TxLength:250}{TxLength Written:250}}
> Set802154phr 1 0 1
{{(Set802154phr)}{PhrSize:2}{PHR: 0x3f10}}
{{(Set802154phr)}{len:250}{payload: 0x10 0x3f 0x11 0x22 0x33 0x44 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee
 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24 0x25 0x26
 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d
 0x3e 0x3f 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f 0x50 0x51 0x52 0x53 0x54
 0x55 0x56 0x57 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b
 0x6c 0x6d 0x6e 0x6f 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f 0x80 0x81 0x82
 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99
 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f 0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf 0xb0
 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7
 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde
 0xdf 0xe0 0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef 0xf0 0xf1 0xf2 0xf3 0xf4 0xf5
 0xf6 0xf7 0xf8 0xf9}}
> tx 10
{{(tx)}{PacketTx:Enabled}{None:Disabled}{Time:3294056593}}
{{(appMode)}{None:Enabled}{PacketTx:Disabled}{Time:3296735316}}
{{(txEnd)}{txStatus:Complete}{transmitted:10}{lastTxTime:3296735238}{timePos:6}{lastTxStart:3296692963}{ccaSuccess:0}{failed:0}
{lastTxStatus:0x000000000}{txRemain:0}{isAck:False}}
```

Notes:

- The maximum framelength value in the PHR is 2047. Therefore, in Wi-SUN FSK, this provides a setTxLength RAILtest command of 2045 (= 2047 – 4 bytes FCS + 2 bytes of PHR).
- Use `setConfigIndex` to select the appropriate Wi-SUN FSK mode available in the *radio_settings.radioconf*.
- Use the `help` command for a full list of CLI command options.

Diagnostic Signals

Both stream mode (PN9) and CW signal generation are supported.

PN9 stream mode also requires setting up PHR (with the set802154PHR command).

As an example, the following CLI commands configure and enable the PN9 stream transmission mode on the radio.

```
> set802154Phr 1 0 1
> setTxStream 1
```

Example:

```
> set802154Phr 1 0 1
 {{(set802154Phr)}{PhrSize:2}{PHR:0x6010}}
 {{(set802154Phr)}{len:4}{payload: 0x10 0x60 0x00 0x00}}
> setTxStream 1
{{(setTxStream)}{Stream:Enabled}{StreamMode:PN9}{Time:2158238801}}
```

Use the following command to send a tone at the carrier frequency.

```
> setTxTone 1
```

Examples:

```
# To Start the tone
> setTxTone 1
 {{{setTxTone}}{Stream:Enabled}{None:Disabled}{StreamMode:Tone}{Time:1293513244}}
# To Stop the tone
> setTxTone 0
{{{setTxTone}}{None:Enabled}{Stream:Disabled}{Time:1298384979}}
```

Wi-SUN OFDM Configuration

Set Up PHR and TX FIFO

Wi-SUN OFDM PHYs are configured to variable packet length encoding. Also, other configurable parameters are stored in the PHR field according to the standard 802.15.4-2020 (detailed in section 20.2.4). It might be difficult to set it up correctly. Therefore, a helper function – the set802154PHR CLI command – has been implemented in the RAILtest project, which simplifies configuration of a valid PHR field and the payload.

Using the RAILtest example, first the transmitted packet's length should be defined with the `setTxLength` CLI command, including the PHR's size (4 bytes). This command sets up the TX FIFO's length and allocates memory for it.

The `set802154PHR` CLI command sets the Rate, Scrambler, and Length fields of the PHR:

1. The first argument selects the PHR format (1, 2, 4 bytes). For OFDM, this should be always set to 2, selecting 4 bytes.
2. The second argument sets the data Rate ("the numerical value of the MCS" as described in the standard in section 20.4).
3. The third argument specifies the scrambling seed (both MSB and LSB bits), according to table 20-20 in the standard.

See an example use case in A Complete OFDM Configuration Example.

The Frame Length field of the PHR is configured according to the length that previously was configured by the setTxLength command. The payload will be padded by autogenerated data.

Note:

- `setTxLength` must always precede the `set802154PHR` command when the TX packet's length is changed.
- Use the `help` command for a full listing of the CLI command options.
- To comply with IEEE 802.15.4, the `set802154PHR` command should be used according to the configured OFDM option set in the Radio Configurator. The command changes the PHR field in the TX packet and for OFDM PHY selects the used MCS in TX.

Configure OFDM PA

OFDM modulation uses a separate PA (Power Amplifier), which should be selected manually before the transmission.

The OFDM PA (RAIL_TX_POWER_MODE_SUBGIG_POWERSETTING_TABLE) is configured by the `RAIL_ConfigTxPower()` RAIL API. This function is wrapped by the `setPowerConfig` CLI command.

This configuration is useful only when changing from FSK modulation to OFDM modulation. It only needs to be done once at initialization.

For an example Multi-PHY configuration with FSK (channel 512) and OFDM (channel 21504), the recommended PA configuration with IR calibration (see IR Calibration Initialization for FSK or OFDM) is:

```
> rx 0
> setchannel 512
> rx 1
> rx 0
> setpowerconfig RAIL_TX_POWER_MODE_SUBGIG_POWERSETTING_TABLE 3600 10
> setpower 100
> setchannel 512
> rx 1
> rx 0
> setchannel 21504
> rx 1
> rx 0
> setpowerconfig RAIL_TX_POWER_MODE_OFDM_PA_POWERSETTING_TABLE 3600 10
> setpower 100
> setchannel 21504
> rx 1
```

A Complete OFDM Configuration Example

This section contains a complete example of how to transmit OFDM modulated packets with the EFR32FG25 radio using the RAILtest application. It assumes the EFR32FG25 is flashed with an image with a Wi-SUN OFDM configuration set with the Radio Configurator.

The following CLI commands are used:

- Set up the radio to use the OFDM PA (only done once when starting to use OFDM modulation).
- Set the recommended PA level (check the maximum level depending on the radio board, 16 dBm for EFR32FG25).
- Set the transmitted packet's length to 250 (=246-byte payload including the 4-byte FCS field plus 4-byte PHR field).
- Configure the PHR with MCS 6 with OFDM Option depending on the *radio_settings.radioconf* and using Wi-SUN 0b00010111 scrambling seed (value 0 in the PHR field).
- Transmit 10 packets.

```
> rx 0
{{(rx)}{Rx:Disabled}{Idle:Enabled}{Time:3470713331}}
> setpowerconfig RAIL_TX_POWER_MODE_OFDM_PA_POWERSETTING_TABLE 3600 10
{{(setpowerconfig)}{success:true}{mode:RAIL_TX_POWER_MODE_OFDM_PA_POWERSETTING_TABLE}{modeIndex:2}{voltage:3600}{rampTime:7}}
> setpower 140
{{(setPower)}{powerLevel:151}{power:140}}
> setchannel 0
{{(setchannel)}{channel:0}}
> rx 1
{{(rx)}{Rx:Enabled}{Idle:Disabled}{Time: 3470724420}}
> setTxLength 250
{{(setTxLength)}{TxLength:250}{TxLength Written:250}}
> Set802154phr 2 6 0
{{(Set802154phr)}{PhrSize:4}{PHR:0xde0c00}}
{{(Set802154phr)}{len:250}{payload: 0x00 0x0c 0xde 0x00 0x33 0x44 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee
 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24 0x25 0x26
 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d
 0x3e 0x3f 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f 0x50 0x51 0x52 0x53 0x54
 0x55 0x56 0x57 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b
 0x6c 0x6d 0x6e 0x6f 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f 0x80 0x81 0x82
 0x83 0x84 0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99
 0x9a 0x9b 0x9c 0x9d 0x9e 0x9f 0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf 0xb0
 0xb1 0xb2 0xb3 0xb4 0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7
 0xc8 0xc9 0xca 0xcb 0xcc 0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde
 0xdf 0xe0 0xe1 0xe2 0xe3 0xe4 0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef 0xf0 0xf1 0xf2 0xf3 0xf4 0xf5
 0xf6 0xf7 0xf8 0xf9}}
> tx 10
{{(tx)}{PacketTx:Enabled}{None:Disabled}{Time:3116912126}}
{{(appMode)}{None:Enabled}{PacketTx:Disabled}{Time:3117028916}}
{{(txEnd)}{txStatus:Complete}{transmitted:10}{lastTxTime:3117028845}{timePos:6}{lastTxStart:3117026790}{ccaSuccess:0}{failed:0}
{lastTxStatus:0x000000000}{txRemain:0}{isAck:False}}
```

Notes:

- `setConfigIndex` can be used to select the appropriate OFDM option available in the *radio_settings.radioconf*.
- For Wi-SUN, the maximum framelength value in the PHR is 2047. So, in Wi-SUN OFDM, this provides a setTxLength RAILtest command of 2051 (= 2047 bytes including the 4 bytes FCS field + 4 bytes of PHR).
- After `set802154phr` , the 4-byte FCS is not added in the packet payload replacing the 4 trailing bytes, but it will be added in the transmitted packet. At the receiving side, the Rx frame has the 4-byte FCS removed.

```
> {{(rxPacket)}{len:246}{timeUs:48492691}{timePos:5}{crc:Pass}{filterMask:0x0}{rssi:-97}{lqi:197}{phy:6}{isAck:False}
{syncWordId:0}{antenna:0}{channelHopIdx:254}{payload: 0x03 0x0c 0xde 0x00 0x33 0x44 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb
0xcc 0xdd 0xee 0x10 0x11 0x12 0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24
0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c 0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c
0x3d 0x3e 0x3f 0x40 0x41 0x42 0x43 0x44 0x45 0x46 0x47 0x48 0x49 0x4a 0x4b 0x4c 0x4d 0x4e 0x4f 0x50 0x51 0x52 0x53 0x54
0x55 0x56 0x57 0x58 0x59 0x5a 0x5b 0x5c 0x5d 0x5e 0x5f 0x60 0x61 0x62 0x63 0x64 0x65 0x66 0x67 0x68 0x69 0x6a 0x6b 0x6c
0x6d 0x6e 0x6f 0x70 0x71 0x72 0x73 0x74 0x75 0x76 0x77 0x78 0x79 0x7a 0x7b 0x7c 0x7d 0x7e 0x7f 0x80 0x81 0x82 0x83 0x84
0x85 0x86 0x87 0x88 0x89 0x8a 0x8b 0x8c 0x8d 0x8e 0x8f 0x90 0x91 0x92 0x93 0x94 0x95 0x96 0x97 0x98 0x99 0x9a 0x9b 0x9c
0x9d 0x9e 0x9f 0xa0 0xa1 0xa2 0xa3 0xa4 0xa5 0xa6 0xa7 0xa8 0xa9 0xaa 0xab 0xac 0xad 0xae 0xaf 0xb0 0xb1 0xb2 0xb3 0xb4
0xb5 0xb6 0xb7 0xb8 0xb9 0xba 0xbb 0xbc 0xbd 0xbe 0xbf 0xc0 0xc1 0xc2 0xc3 0xc4 0xc5 0xc6 0xc7 0xc8 0xc9 0xca 0xcb 0xcc
0xcd 0xce 0xcf 0xd0 0xd1 0xd2 0xd3 0xd4 0xd5 0xd6 0xd7 0xd8 0xd9 0xda 0xdb 0xdc 0xdd 0xde 0xdf 0xe0 0xe1 0xe2 0xe3 0xe4
0xe5 0xe6 0xe7 0xe8 0xe9 0xea 0xeb 0xec 0xed 0xee 0xef 0xf0 0xf1 0xf2 0xf3 0xf4 0xf5}}
```

Diagnostic Signals

With OFDM modulation configured, both stream mode (PN9) and CW signal generation are supported.

PN9 stream mode also requires setting up PHR (with the `set802154PHR` command) to select the MCS Level. It is also mandatory to configure the frame length field to 4 before the transmission, to set up the infinite stream needed for PN9.

As an example, the following CLI commands configure and enable the PN9 stream transmission mode on the radio in OFDM option 1 MCS4 at configindex 1.

```
> setconfigindex 1
> rx 0
> setpowerconfig RAIL_TX_POWER_MODE_OFDM_PA_POWERSETTING_TABLE 3600 10
> setpower 140
> rx 1
> setTxLength 4
> set802154Phr 2 4 0
> setTxStream 1
```

Example:

```
> setconfigindex 1
 {{(setconfigindex)}{configIndex:1}{firstAvailableChannel:0}}
> setpowerconfig RAIL_TX_POWER_MODE_OFDM_PA_POWERSETTING_TABLE 3600 10
 {{(setpowerconfig)}{success:true}{mode:RAIL_TX_POWER_MODE_OFDM_PA_POWERSETTING_TABLE}{modeIndex:2}{voltage:3600}{rampTime:7}}
> setpower 140
 {{(setpower)}{powerLevel:151}{power:140}}
> setTxLength 4
 {{(setTxLength)}{TxLength:4}{TxLength Written:4}}
> set802154Phr 2 4 0
 {{(set802154Phr)}{PhrSize:4}{PHR:0x400}}
 {{(set802154Phr)}{len:4}{payload: 0x00 0x04 0x00 0x00}}
> setTxStream 1
{{(setTxStream)}{Stream:Enabled}{None:Disabled}{StreamMode:PN9}{Time:1737513509}}
```

Use the following command to send a tone at the carrier frequency.

```
> setTxTone 1
```

Examples:

```
# To Start the tone
> setTxTone 1
  {{{setTxTone}}{Stream:Enabled}{None:Disabled}{StreamMode:Tone}{Time:1293513244}}
# To Stop the tone
> setTxTone 0
{{{setTxTone}}{None:Enabled}{Stream:Disabled}{Time:1298384979}}
```

It is equivalent to:

```
> setTxStream 1 0
```

IR Calibration Initialization for FSK or OFDM

For FSK or OFDM PHY (during initialization), after a setchannel on the corresponding PHY, a `rx 1` or `tx 1` is required to perform the calibration.

When using multiple PHYs including an OFDM PhyModeID, an IR calibration for OFDM must be performed. Silicon Labs strongly recommends loading one OFDM PHY (during initialization) using the setchannel on the corresponding OFDM channel and a `rx 1` RAILtest CLI command to perform the calibration. Once image rejection calibration has been performed with a Wi-SUN OFDM PHY, you do not need to do it with a Wi-SUN FSK PHY.

Known Issues With EFRFG25

Known Issues with EFRFG25

Board-Specific Limitations

Board Support Limitations

When using the mainboard's LCD, Silicon Labs recommends resetting the mainboard using either the reset button or the CLI reset command.

Missing CTune Calibration

Officially released Silicon Labs EFR32 radio boards equipped with HFXO (High-frequency Crystal Oscillator) are manufacturer-calibrated, and the calibrated CTune value is stored in the radio board's EEPROM memory. This value is then accessible (can be read and written) by Simplicity Commander.

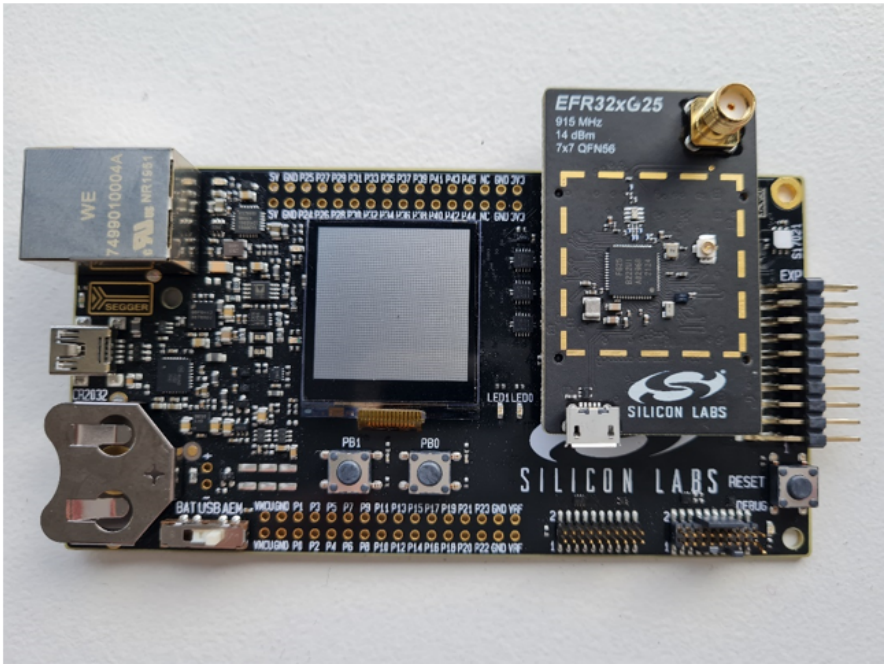
However, the EFR32FG25 radio boards have no calibrated CTune value stored in the EEPROM memory.

The CTune value can be calibrated and retrieved at runtime by `setCTune` and `getCTune` RAILtest CLI commands respectively, or by using the `RAIL_SetTune()` and `RAIL_GetTune()` RAIL APIs. The calibration value can be stored using Simplicity Commander or through the Simplicity Studio Device Configuration menu.

Missing Notch on the Radio Board

Silicon Labs radio boards have little notches at the board connectors that prevent the user from connecting the radio board in a reversed orientation onto a WSTK.

For the correct orientation, see the following figure.



Warning: Incorrect placement of the radio board might cause serious damage on the mainboard. Make sure that the radio board is connected with the correct orientation before supplying power to the mainboard!

OFDM PA Limitations

OFDM modulation is designed to use a separate PA from that being used with different modulation types (2/4(G)FSK, MSK, OOK).

For all PA selected in a multi-PHY configuration, the PA initialization should be done. This is required only once, at initialization, and FSK must be done before OFDM:

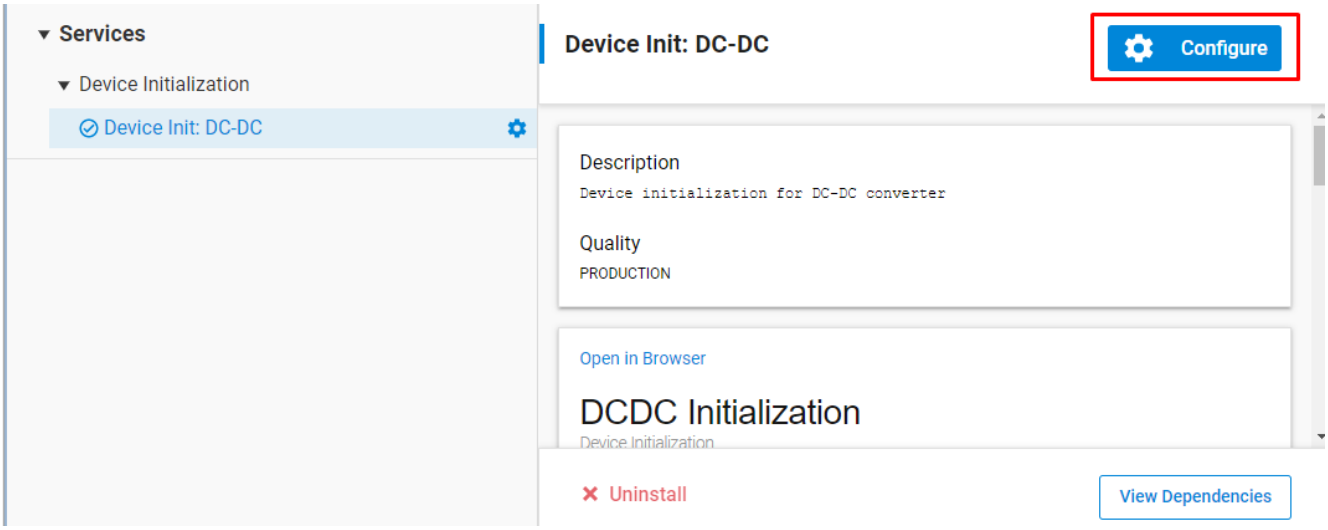
- Set up the PA configuration
- Set the recommended PA level (check the max level depending on the radio board, 16 dBm for EFR32FG25)

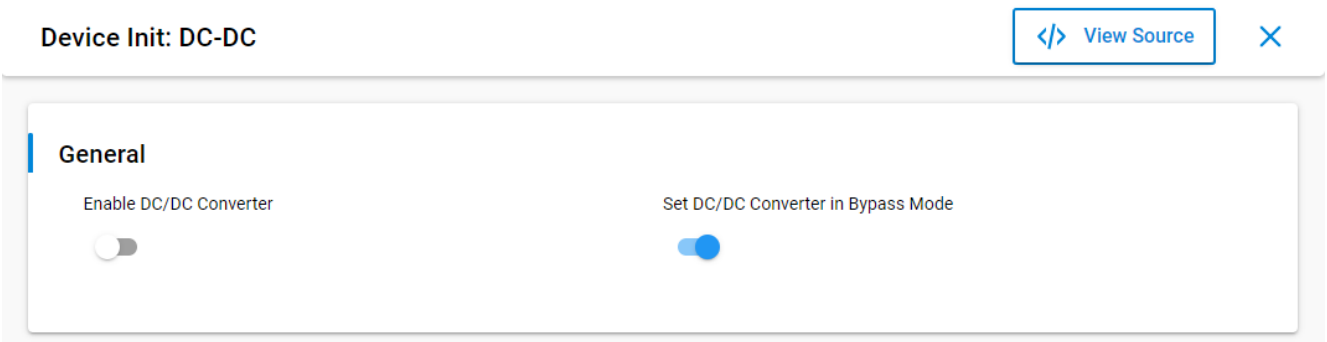
Clocking

Switching PHYs while using the RFPLL as a clock source will incur a clock frequency shift.

DC/DC is On by Default

The sensitivity might be slightly degraded when the DC/DC is enabled. For best sensitivity, Silicon Labs recommends setting DC/DC to bypass mode. In the **Device Init: DC-DC** component, toggle **Enable DC/DC Converter** to off, and toggle **Set DC/DC Converter in Bypass Mode** to on, as shown in the following figures.





Miscellaneous Issues

- RAIL timings are not accurate in comparison with on-air timing for Rx or TX frames. Manual tuning might be required.
- The 802.15.4 MAC address filtering with OFDM PHY for small frame size is not done correctly and then the frame is always received.

Other information

- Silicon Labs recommends using the BRD4270B, BRD4271A, or BRD4272A radio board for best performance:
  - BRD4270B revA06 radio board RF matching network is tuned for 915 MHz and 920 MHz bands.
  - BRD4271A revA06 radio board RF matching network is tuned for 868 MHz bands.
  - BRD4272A revA03 radio board RF matching network is tuned for 470 MHz bands.
- Mode Switch mechanism is available and is documented separately in AN1403: Wi-SUN Mode Switch on EFR32FG25 with RAILtest.