Wi-SUN

- Developing with Wi-SUN
- Release Notes
  - Wi-SUN
- Quick Start Guide
  - Introduction
  - Getting Started
    - Overview
    - Wi-SUN Sample Applications
    - Create a Wi-SUN Network
  - Going Further
- Development Walkthrough
  - Overview
  - Build and Connect
  - API Calls to Connect
  - Add a Custom Application
    - Overview
    - Timestamping
    - Custom Callback
    - JSON Connection Strings
    - Send Status Strings
    - Add CoAP Resources
    - Retrieve UDP Notifications
    - Retrieve Device Information
  - OTA DFU
- Wi-SUN Node
  - Overview
  - Wi-SUN Developer's Guide (PDF)
  - Wi-SUN Configurator
  - FAN 1.0 Node Certification
  - Wi-SUN Limited Function Nodes (LFN)
- Platform Resources
  - Overview
  - Bootloading
    - Overview
    - Bootloader Fundamentals
      - Introduction
      - Overview
      - About the Gecko Bootloader

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Memory Space for Bootloading
- Design Decisions
- Gecko Bootloader User's Guide for Series 3 and Higher
  - Introduction
  - Gecko Bootloader File Format v4
  - Gecko Bootloader Operation Application Upgrade
  - Gecko Bootloader Operation Bootloader Upgrade
  - Gecko Bootloader Operation Secure Engine Upgrade
  - Getting Started with the Gecko Bootloader
  - Configuring the Gecko Bootloader
  - Simplicity Commander and the Gecko Bootloader
  - Gecko Bootloader Security Features
  - Application Interface
- Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher
  - Introduction
  - Overview
  - Gecko Bootloader File Format
  - Gecko Bootloader Operation Application Upgrade
  - Gecko Bootloader Operation Bootloader Upgrade
  - Gecko Bootloader Operation Secure Engine Upgrade
  - Getting Started with the Gecko Bootloader
  - Configuring the Gecko Bootloader
  - Simplicity Commander and the Gecko Bootloader
  - Gecko Bootloader Security Features
  - Gecko Bootloader and Trustzone
  - Gecko Bootloader and Delta DFU Upgrades
  - Application Interface
- Non-Volatile Memory Use
  - Overview
  - Non-Volatile Data Storage Fundamentals
    - Overview
    - Introduction
    - Implementations of Non-Volatile Data Storage
    - Dynamic Data Storage Implementations
    - Comparing Non-Volatile Data Storage Implementations
  - Using Third Generation Non-Volatile Memory (NVM3) Data Storage
    - Introduction
    - Overview
    - NVM3 Default Instance
    - NVM3 in the Simplicity Studio 5 Project Configurator
    - Using NVM3 with Silicon Labs Connect
    - Using NVM3 with Silicon Labs OpenThread Applications
    - Using NVM3 with Zigbee EmberZNet v.7x And Higher
    - Using NVM3 with Silicon Labs Bluetooth Applications

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Using NVM3 In Appbuilder Based Applications
- Using NVM3 with Z Wave Applications
- NVM3 API Options
- Simplicity Commander and NVM3
- Securing NVM3 Data on External Flash Devices
- Best Practices and Optimizations For NVM3

Security

- Overview
- IoT Endpoint Security Fundamentals
  - Introduction
  - Overview
  - No Universal Passwords
  - Secured Interfaces
  - Proven Cryptography
  - Security by Default
  - Signed Software Updates
  - Automatically Applied Updates
  - Vulnerability Reporting Program
  - Security Expiration Date
  - Next Steps
  - Series 2 Device Security Features
- Security Concepts and Design Considerations
- Integrating Crypto Functionality with PSA Crypto vs. Mbed TLS (PDF)

Wi-SUN Border Router

- Overview
- Network Configuration
  - Overview
  - Wi-SUN SoC Border Router
  - Wi-SUN Linux Border Router
  - CPCD and wsbrd
  - External Servers
  - SoC Border Router Backhaul
  - Using CPC SPI on Linux Border Router
- IP Communication
  - Overview
  - Ping and UDP
  - CoAP
  - Multicast
- Border Router GUI

Network Performance

- Overview
- Wi-SUN PAN Defect
- Using the Wi-SUN Network Measurement Application
  - Introduction

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Theory Of Operation
- Lcd Screen Interface
- Command Line Interface
- IPv6 UDP iperf Testing
- CoAP Communication Between The Linux Border Router And The Network Performance Application
- Ping Latency Improvement
- SiLabs Wi-SUN Tools
  - Direct Connect
  - Dynamic Device Provisioning
- API Reference
  - Wi-SUN Services
    - Util Functions
    - Application Core
      - Configurations
      - Type definitions
        - sl_wisun_app_core_current_addr
        - sl_wisun_app_core_reg_thresholds
        - sl_wisun_app_core_time_stat
    - Application Settings
      - Type definitions
        - app_setting_wisun
    - CoAP
      - Configurations
      - Type definitions
        - sl_wisun_coap
      - CoAP Resource Handler API
        - Type definitions
          - sl_wisun_coap_rhnd_resource
          - sl_wisun_coap_rhnd_resource_data
      - CoAP Notification API
        - Type definitions
          - sl_wisun_coap_notify
    - Ping
      - Configurations
      - Type definitions
        - sl_wisun_ping_echo_request
        - sl_wisun_ping_info
        - sl_wisun_ping_stat
    - iPerf
      - Configurations
      - Type definitions
        - sl_iperf_clnt_hdr_ext
        - sl_iperf_clnt_hdr_ext_isoch_settings
        - sl_iperf_clnt_hdr_ext_starttime_fq

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_clnt_hdr_isoch_payload
sl_iperf_conn
sl_iperf_log
sl_iperf_log_str_buff
sl_iperf_opt
sl_iperf_stats
sl_iperf_test
sl_iperf_udp_clnt_hdr
sl_iperf_udp_clnt_hdr_v1
sl_iperf_udp_datagram
sl_iperf_udp_srv_hdr
Portable interface
sl_iperf_socket_addr
sl_iperf_socket_in6addr
sl_iperf_time
Over-The-Air Device Firmware Upgrade (Alpha)
Type definitions
sl_wisun_ota_dfu_error_ctx
sl_wisun_ota_dfu_error_ctx_btl_fw_set
sl_wisun_ota_dfu_error_ctx_btl_fw_verify
sl_wisun_ota_dfu_error_ctx_fw_download
sl_wisun_ota_dfu_settings
sl_wisun_ota_dfu_statistics
Configurations
Trivial File Transfer Protocol (TFTP) Client (Alpha)
Configurations
Type definitions
sl_tftp_ack_pkt
sl_tftp_clnt
sl_tftp_clnt_opt
sl_tftp_data_pkt
sl_tftp_error_pkt
sl_tftp_oack_pkt
sl_tftp_pkt
sl_tftp_pkt_content
sl_tftp_req_pkt
Portable interface
Application Memory Statistic (Alpha)
Configurations
Application Status over CoAP (Alpha)
Configurations
Silicon Labs socket API (deprecated)
Keychain
sl_wisun_keychain_credential_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_keychain_entry_t
sl_wisun_keychain_key_t
Wi-SUN Stack Plugin
Stack Trace and Debug
RF Test
Wi-SUN Stack API
Wi-SUN API events
sl_wisun_evt_t
sl_wisun_msg_connected_ind
sl_wisun_msg_connected_ind_body_t
sl_wisun_msg_connected_ind_t
sl_wisun_msg_pan_defect_ind
sl_wisun_msg_pan_defect_ind_body_t
sl_wisun_msg_pan_defect_ind_t
sl_wisun_msg_network_update_ind
sl_wisun_msg_network_update_ind_body_t
sl_wisun_msg_network_update_ind_t
sl_wisun_msg_socket_data_ind
sl_wisun_msg_socket_data_ind_body_t
sl_wisun_msg_socket_data_ind_t
sl_wisun_msg_socket_data_available_ind
sl_wisun_msg_socket_data_available_ind_body_t
sl_wisun_msg_socket_data_available_ind_t
sl_wisun_msg_socket_connected_ind
sl_wisun_msg_socket_connected_ind_body_t
sl_wisun_msg_socket_connected_ind_t
sl_wisun_msg_socket_connection_available_ind
sl_wisun_msg_socket_connection_available_ind_body_t
sl_wisun_msg_socket_connection_available_ind_t
sl_wisun_msg_socket_closing_ind
sl_wisun_msg_socket_closing_ind_body_t
sl_wisun_msg_socket_closing_ind_t
sl_wisun_msg_disconnected_ind
sl_wisun_msg_disconnected_ind_body_t
sl_wisun_msg_disconnected_ind_t
sl_wisun_msg_connection_lost_ind
sl_wisun_msg_connection_lost_ind_body_t
sl_wisun_msg_connection_lost_ind_t
sl_wisun_msg_socket_data_sent_ind
sl_wisun_msg_socket_data_sent_ind_body_t
sl_wisun_msg_socket_data_sent_ind_t
sl_wisun_msg_error_ind
sl_wisun_msg_error_ind_body_t
sl_wisun_msg_error_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_join_state_ind
sl_wisun_msg_join_state_ind_body_t
sl_wisun_msg_join_state_ind_t
sl_wisun_msg_regulation_tx_level_ind
sl_wisun_msg_regulation_tx_level_ind_body_t
sl_wisun_msg_regulation_tx_level_ind_t
sl_wisun_mode_switch_fallback_level_ind
sl_wisun_msg_mode_switch_fallback_ind_body_t
sl_wisun_msg_mode_switch_fallback_ind_t
sl_wisun_msg_rx_frame_ind
sl_wisun_msg_rx_frame_ind_body_t
sl_wisun_msg_rx_frame_ind_t
sl_wisun_msg_lfn_wake_up_ind
sl_wisun_msg_lfn_wake_up_ind_body_t
sl_wisun_msg_lfn_wake_up_ind_t
sl_wisun_msg_lfn_multicast_reg_ind
sl_wisun_msg_lfn_multicast_reg_ind_body_t
sl_wisun_msg_lfn_multicast_reg_ind_t
sl_wisun_msg_dhcp_vendor_data_ind
sl_wisun_msg_dhcp_vendor_data_ind_body_t
sl_wisun_msg_dhcp_vendor_data_ind_t
sl_wisun_msg_direct_connect_link_available_ind
sl_wisun_msg_direct_connect_link_available_ind_body_t
sl_wisun_msg_direct_connect_link_available_ind_t
sl_wisun_msg_direct_connect_link_status_ind
sl_wisun_msg_direct_connect_link_status_ind_body_t
sl_wisun_msg_direct_connect_link_status_ind_t
sl_wisun_br_msg_stopped_ind
sl_wisun_br_msg_stopped_ind_body_t
sl_wisun_br_msg_stopped_ind_t
sl_wisun_br_msg_routing_table_update_ind
sl_wisun_br_msg_routing_table_update_ind_body_t
sl_wisun_br_msg_routing_table_update_ind_t
Wi-SUN API type definitions
sl_wisun_br_connection_params_t
sl_wisun_br_lfn_params_t
sl_wisun_br_params_configuration_s
sl_wisun_br_params_dhcp_s
sl_wisun_br_params_discovery_s
sl_wisun_br_params_eapol_s
sl_wisun_br_params_lfn_parent_s
sl_wisun_br_params_misc_s
sl_wisun_br_params_mpl_s
sl_wisun_br_params_rpl_s

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_traffic_s
sl_wisun_br_routing_table_entry_t
sl_wisun_channel_mask_t
sl_wisun_connection_params_t
sl_wisun_key_lifetimes_params_t
sl_wisun_lfn_params_connection_t
sl_wisun_lfn_params_data_layer_t
sl_wisun_lfn_params_network_t
sl_wisun_lfn_params_power_t
sl_wisun_lfn_params_t
sl_wisun_mac_address_t
sl_wisun_mac_params_t
sl_wisun_msg_header_t
sl_wisun_neighbor_info_t
sl_wisun_network_info_t
sl_wisun_params_configuration_s
sl_wisun_params_discovery_s
sl_wisun_params_eapol_s
sl_wisun_params_lfn_parent_s
sl_wisun_params_misc_s
sl_wisun_params_mpl_s
sl_wisun_params_rpl_s
sl_wisun_params_traffic_s
sl_wisun_phy_config_custom_fsk_t
sl_wisun_phy_config_custom_ofdm_t
sl_wisun_phy_config_custom_oqpsk_t
sl_wisun_phy_config_explicit_t
sl_wisun_phy_config_fan10_t
sl_wisun_phy_config_fan11_t
sl_wisun_phy_config_ids_t
sl_wisun_phy_config_t
sl_wisun_rate_stats_t
sl_wisun_rate_t
sl_wisun_rpl_info_t
sl_wisun_socket_option_data_t
sl_wisun_socket_option_edfe_mode_t
sl_wisun_socket_option_event_mode_t
sl_wisun_socket_option_multicast_group_t
sl_wisun_socket_option_multicast_hop_limit
sl_wisun_socket_option_send_buffer_limit_t
sl_wisun_socket_option_unicast_hop_limit
sl_wisun_statistics_arib_regulation_t
sl_wisun_statistics_fhss_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_statistics_heap_t
sl_wisun_statistics_mac_t
sl_wisun_statistics_network_t
sl_wisun_statistics_phy_t
sl_wisun_statistics_regulation_t
sl_wisun_statistics_t
sl_wisun_statistics_wisun_t
sl_wisun_trace_group_config_t
sl_wisun_trickle_params_t
Predefined FFN parameter sets
Predefined LFN parameter sets
Predefined BR parameter sets
Predefined BR LFN-parenting parameter sets
Wi-SUN API availability
Wi-SUN Regional Regulation API
sl_wisun_regulation_params_t
Predefined regional regulation parameter sets
Border Router API
Socket API
cmsghdr
in6_addr
in6_pktinfo
iovec
ip6_mtuinfo
ipv6_mreq
msghdr
sockaddr
sockaddr_in6
Select API
fd_set
Wi-SUN Stack Changelog
Wi-SUN Sockets
Deprecated List
Wi-SUN Overview
Wi-SUN PHY
Overview
Getting Started with Wi-SUN PHY
Overview
How To Use Wi-SUN FAN 1.0 PHY
How To Use Wi-SUN FAN 1.1 On EFR32FG25
Wi-SUN Configuration With RAILtest For EFR32FG25
Known Issues With EFRFG25
Mode Switch (PDF)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Concurrent Mode (PDF)

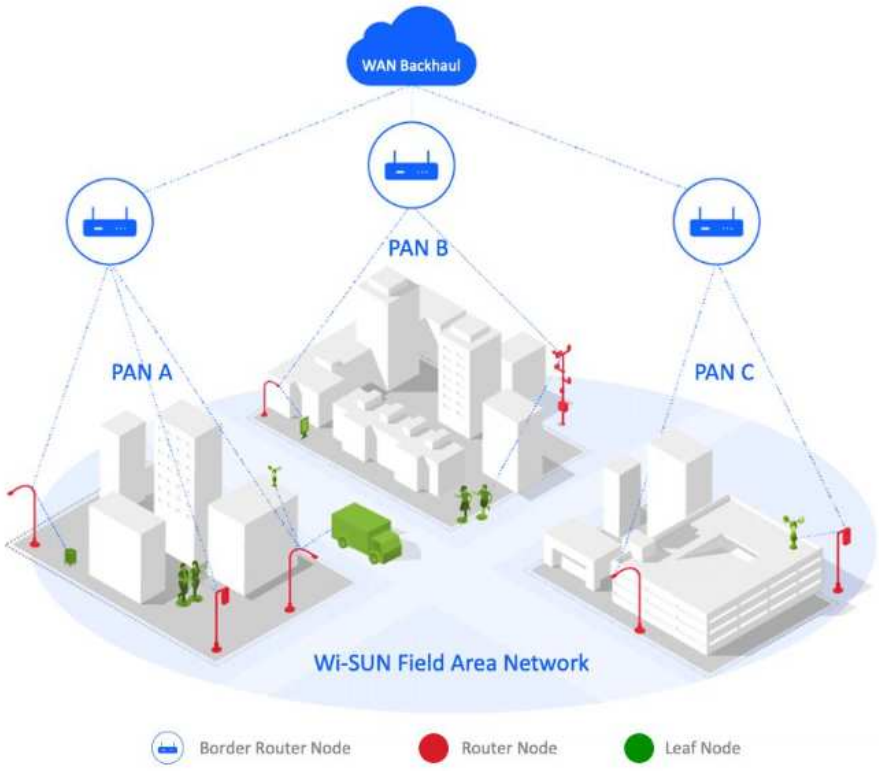
Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN

Wi-SUN SDK Version 2.7.0 (July 21, 2025) - Release Notes

Simplicity SDK Version 2025.6.1

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' Wi-SUN hardware is certified by the Wi-SUN Alliance, a global industry association devoted to seamless LPWAN connectivity. Wi-SUN builds upon open standard internet protocols (IP) and APIs, enabling developers to extend existing infrastructure platforms to add new capabilities. Built to scale with long-range capabilities, high-data throughput and IPv6 support, Wi-SUN simplifies wireless infrastructure for industrial applications and the evolution of smart cities.

Release Summary

Key Features | API Changes | Bug Fixes | Chip Enablement

Key Features

Added in 2.7.0

None.

Added in 2.6.0

- Added Wi-SUN+BLE dynamic multi-protocol (DMP) support.
- Added concurrent detection support.

API Changes

Changed in 2.7.0

- Added new APIs to fine tune the regional regulation mechanisms
- Extended the connection parameters with lower-MAC configurations

Changed in 2.6.0

Added sl_wisun_set_preferred_pan() API that sets a FFN preferred PAN. A router will try to roam to its preferred PAN if it is available.

Bug Fixes

Fixed in 2.7.0

Fixed a security vulnerability in the Border Router SoC authenticator. It was introduced in 2.6.0.

Fixed in 2.6.0

- Fixed an issue causing devices to be stuck in join state 3.

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Fixed multiple MAC issues causing RCP instabilities.

Chip Enablement

Added in 2.7.0

None.

Added in 2.6.0

EFR32FG23 as RCP.

Key Features

New Features | Enhancements | Removed Features | Deprecated Features

New Features

Added in 2.7.0

- Added `sl_wisun_set_regulation_parameters()` API. It is now recommended to use this new interface to configure the regional regulation mechanisms used in the stack. While `sl_wisun_set_regulation()` is not yet deprecated, using `sl_wisun_set_regulation_parameters()` is strongly encouraged. The new interface offers finer control over the parameters that impact compliance with regional regulations and is decoupled from the Regulator Domain set in the connection parameters. Presets per region are exposed in sl_wisun_regulation_api.h and could be used as-is or copied and edited.
- Added `sl_wisun_reset_regulation_duty_cycle()` API. This API resets the transmission duty cycle counters. It is recommended to call this API after a maintenance operation if the regional regulation permits excluding certain operations from duty cycling constraints.
- Added `sl_wisun_set_eap_identity()`. This function sets the identity used in the EAP Identity Response during EAP authentication.
- Extended the capacities of sl_wisun_set_connection_parameters() with low-MAC configurations in a backward-compatible manner. Is is now possible to fine-tune the CSMA and retransmission configurations. It is recommended to lower the min_be value to reach higher throughput.
- Made the statistic type definition public and added a new statistic field to distinguish between file transfer time and total firmware update time.

Added in 2.6.0

- Added concurrent detection support. EFR32FG25 is capable of receiving and processing FSK and OFDM without any prior negotiation, optimizing the mode switch procedure by making the mode switch PHR optional. Refer to AN1410 Getting Started with Concurrent Mode on EFR32FG25 using RAILtest for more information.
- Added `sl_wisun_config_concurrent_detection()` API. It configures whether an alternate PHY can be used concurrently for transmissions instead of the standard mode switch.
- Added Wi-SUN/BLE dynamic multiprotocol (DMP) support.
- Updated BZ and SG channel plans according to Wi-SUN PHY TPS 2v03.
- Ownership of `chan_params_table` is now transferred to the application. Each entry of `chan_params_table` defines a specific configuration containing PHY information such as the frequency, the channel numbers, and the modulation. It is recommended to adapt the content of the table to the end application use to optimize flash consumption. It is critical to ensure the content of the table is well aligned with the content of the configuration set in Simplicity Studio Radio Configurator.
- Added `sl_wisun_set_preferred_pan()` API. It sets a preferred PAN ID that the device will always select during connection if available, or seek to roam to while connected to another PAN.
- Added `sl_wisun_config_neighbor_table()` API. It configures the maximum number of RPL children, neighbors, and security neighbors, allowing fine-tuning of memory usage and network scalability.
- Added `sl_wisun_br_trigger_global_repair()` border router API to manually trigger a global repair. It allows manual triggering of a global repair operation on the border router.
- Added `sl_wisun_br_get_routing_table_entry_count()` and `sl_wisun_br_get_routing_table()` for the SoC Border Router. They can be used to retrieve the number of routing table entries and the routing table itself in order to render the network topology.

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Added an event `SL_WISUN_MSG_BR_ROUTING_TABLE_UPDATE_IND_ID` that is triggered when the Border Router (BR) routing table is updated, with differentiation by update type. The structure returned by this event contains details about the routing table change, including the type of update and relevant routing entry information.
- Introduced `sl_wisun_lfn_params_t` v3, with new `eapol_unicast_interval_ms` parameter.
- Extended the network statistics (`sl_wisun_statistics_network_t`) with MPL statistics.
- Extended the network statistics (`sl_wisun_statistics_network_t`) with the node's estimated hop count to the BR. Paired with `sl_wisun_set_leaf()`, it could be used to limit the depth of the network.
- A `SL_WISUN_MSG_NETWORK_UPDATE_IND` event is now triggered upon a hop count estimation update. Added a new value `SL_WISUN_NETWORK_UPDATE_FLAGS_HOP_COUNT` to `sl_wisun_msg_network_update_ind_body_t` `flag` field.
- Extended the MAC statistics (`sl_wisun_statistics_mac_t`) with the time spent in idle and TX states.
- Extended the Heap statistics (`sl_wisun_statistics_heap_t`) with output highest used, currently used, free, and total heap space.
- Extended the connection parameters (`sl_wisun_connection_params_t`) with fragmentation and EDFE configuration parameters.
- Extended the LFN configuration parameters (`sl_wisun_lfn_params_t`) with EAPOL unicast wake-up interval.
- Added multi-fragment EDFE support. EDFE is used automatically on all fragmented packets.
- Introduced Wi-SUN NVM versioning.
- ETSI Adaptive Power Control is now available for EU PHYs. It was previously limited to Indian configurations.
- FFN and LFN now store frame counters for up to 15 different PANs. It was necessary to be able to navigate between two PANs either in the context of a switch to a preferred PAN or following multiple power outages.

Enhancements

Added in 2.7.0

None.

Added in 2.6.0

- Rework the supplicant leading to significant RAM and Flash usage diminution.
- Enhanced debug and trace capabilities for RCP and SoC. MAC traces are now enabled by default in the RCP.
- ACK timeout is now measured to the start of the response frame, allowing faster RX channel return if no response is received.
- EDFE FC-IE transmit flow control now matches estimated frame duration. It was previously set to the maximum value and was not used on the receiver's side.
- Wi-SUN Header IEs (WH-IEs) from duplicate frames are now processed.
- Delayed initial DIS to avoid overlap with PC sequence.
- Added more meaningful status codes for `SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID` indication.
- Temporary denial of routers unable to parent. Do not send unicast DIS to denied neighbors.

Removed Features

None.

Deprecated Features

None.

API Changes

New APIs | Modified APIs | Removed APIs | Deprecated APIs

New APIs

Added in 2.7.0

New API Signature	Deprecated API replaced by this (if any)
sl_wisun_set_regulation_parameters()	*None*
sl_wisun_reset_regulation_duty_cycle()	*None*

Copyright © 2025 Silicon Laboratories. All rights reserved.

New API Signature	Deprecated API replaced by this (if any)
sl_wisun_set_eap_identity()	*None*
sl_wisun_ota_dfu_get_fw_update_statistics()	*None*

Added in 2.6.0

New API Signature	Deprecated API replaced by this (if any)
sl_wisun_set_preferred_pan()	*None*
sl_wisun_config_neighbor_table()	sl_wisun_set_neighbor_table_size()
sl_wisun_br_trigger_global_repair()	None
sl_wisun_config_concurrent_detection()	None
sl_wisun_br_get_routing_table_entry_count()	None
sl_wisun_br_get_routing_table()	None

Modified APIs

Modified in 2.7.0

- Extended the capacities of sl_wisun_set_connection_parameters() with low-MAC configurations in a backward-compatible manner. Is is now possible to fine-tune the CSMA and retransmission configurations. It is recommended to lower the min_be value to reach higher throughput.

Modified in 2.6.0

- Extended the capacities of sl_wisun_set_connection_parameters() with fragmentation and EDFE configuration in a backward-compatible manner.
- Extended the capacities of sl_wisun_set_lfn_parameters() with EAPOL unicast wake-up interval in a backward-compatible manner.

Removed APIs

None.

Deprecated APIs

Deprecated in 2.7.0

None.

Deprecated in 2.6.0

Deprecated API Name	Planned Removal Date
sl_wisun_set_neighbor_table_size()	Not before June 5, 2026

Bug Fixes

Fixed in 2.7.0

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1474061	Fixed authenticator vulnerability to installation of predictable keys.		BR

Copyright © 2025 Silicon Laboratories. All rights reserved.

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1466748	Corrected an issue where the authenticator was dropping supplicant retries under high congestion conditions. This fix ensures that supplicants re-initiating TLS while already on the active list are no longer dropped.		
1465055	Fixed asynchronous fragments splitting with DFE/EDFE frames.		BR, FFN
1464297	Fixed socket TX buffer filling in case of routing or memory error.		BR, FFN, LFN
1468808	Fixed segmentation fault when unexpected EDFE final frame is received.		BR, FFN
1473077	Fixed a fault when an LFN fails to complete multiple handshakes in join state 5.		LFN
1458827	Fixed node failing to connect after obtaining an inactive GTK first.		FFN
1432653	Fixed an invalid hop limit when forwarding multicast packets to LFNs. It was not decremented		BR, FFN
1476119	Fixed CCA threshold value used for O-QPSK PHYs.		BR, FFN, LFN
1478072	Fixed memory corruption when returning to join state 1 while proxying RPL for LFNs.		FFN
1479483	Fixed MAC TX confirmation overwriting previous one in case of consecutive failures.		BR, FFN, LFN, RCP
1464597	Fixed timeout management on asynchronous packets. It was causing blockage in the RPC when trying to start new asynchronous transmissions.		RCP

Fixed in 2.6.0

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1373901	Fixed mode switch from router to LFN.		FFN
1384392	Fixed a PSA crypto key leak that prevented authentication after multiple attempts during the device's lifetime.		FFN
1385981	Fixed a critical failure when receiving a GKH in join state 1 by dropping security handshake frames from previous connections.		FFN
1385791	Fixed an issue that prevented LFNs from sending LPAS.		LFN
1388948	Fixed LFNs not unregistering from multicast groups when the corresponding socket was closed.		LFN
1144019	Fixed MPL messages being transmitted more times than the MPL "TimerExpirations" configuration allowed.		BR, FFN
1392144	Fixed DHCP preferred lifetime being invalid when the address lifetime was infinite.		FFN
1393742	Fixed SoC BR task priority starvation by the Wi-Fi backhaul task.		BR
1366711	Fixed unregistering when changing parent (sent NS(ARO) with no lifetime).		FFN
1394368	Fixed RPL Global Repair not transmitting DAO if the same preferred parent was selected.		FFN
1390717	Fixed memory corruption causing authentication failures on Secure Vault Mid devices.		BR, FFN, LFN
1399233	Fixed neighbor cache refresh procedure.		BR, FFN

Copyright © 2025 Silicon Laboratories. All rights reserved.

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1389186	Fixed LFN being blocked in reception state for the duration of a broadcast interval.		LFN
1394656	Fixed aborted frames not generating confirmation, which caused many RCP stalls.		BR, FFN, LFN
1401857	Fixed LFN PAN Advertisement transmissions sometimes being stuck until timeout.		LFN
1230806	Fixed EDFE retransmissions not performing retries with CCA backoff.		BR, FFN
1412771	Fixed an FHSS synchronization issue caused by incorrect UFSI values.		BR, FFN, LFN
1412117	Fixed a radio configuration file error with SG PHYs.		BR, FFN, LFN
1412791	Fixed an issue where border router address/prefix lifetime was advertised as 0 instead of infinite in DIO packets.		BR
1411742	Fixed an issue where nodes roamed back and forth with PAN Defect and Preferred PAN enabled. A device could not re-join its previous PAN before a given duration hardcoded in the stack.		FFN
1414646	Fixed an issue by analyzing inner IPv6 headers to refresh neighbor address registrations.		BR, FFN
1419046	Fixed an issue that caused devices to be stuck in join state 3 when they could not decrypt a PC from their previous PAN.		FFN
1271915	Fixed an issue that prevented DAO path sequence counter from resetting when going to join state 1 and 3.		FFN
1426446	Fixed a rescheduling issue with LFN frames that would make a node completely unable to operate.		FFN, LFN
1425204	Fixed invalid duplicated frame detection from LFN neighbors as an FFN.		LFN
1425204	Fixed invalid duplicated frame detection from FFN parent as an LFN.		FFN
1438701	Fixed an LFN issue when multiple frames advertised an updated LUS-IE.		LFN
1440283	Fixed an issue handling invalid 4WH message 3 and GKH message 1.		FFN, LFN
1437500	Fixed the return option value of getsockopt() for SO_SNDLOWAT option.		BR, FFN, LFN
1436839	Fixed send/sendto setting the wrong errno when sendq was full and socket was not in blocking mode.		BR, FFN, LFN
1436839	Fixed send/sendto not returning the correct number of sent bytes in the case of a partial packet send.		BR, FFN, LFN
1435786	Fixed setting errno values for socket API. It was missing for some of the socket APIs.		BR, FFN, LFN
1442451	Fixed synchronization to broadcast schedule after LFN parent selection.		FFN, LFN
1429143	Fixed an issue by deleting ND entries upon the reception of a NS with ARO containing a lifetime of 0.		BR, FFN
1445797	Fixed incorrect backoff slot duration with SUN OFDM.		BR, FFN, LFN
1441981	Fixed an issue where LFN failed in join state 3 when it failed to decrypt an FFN-specific packet.		LFN
1424293	Fixed an issue where PAN Defect IE was relayed twice by FFNs.		FFN
1446315	Fixed ETSI Adaptive Power Control with FAN TPS version 1.0.		BR, FFN, LFN
1452100	Fixed invalid default k trickle parameter.		BR

Copyright © 2025 Silicon Laboratories. All rights reserved.

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1453003	Fixed an issue that caused the connections to be significantly longer when an error occurred while trying to send a DAO. Routers were unable to send a second DAO until PAN timeout.		FFN
1454000	Fixed an issue where Preferred PAN and PAN Defect generated too much traffic.		FFN
1387966	Fixed an issue where RCP reset was triggered when MAC purged LFN frames upon LFN PAN Advertisement transmission. It was an abusive critical failure.		RCP
1402855	Fixed Source Addressing Mode of EDFE Response Frames in the RCP.		RCP
1377305	Fixed a wrong error message when reconnecting during an ongoing disconnection.		CLI
1390370	Fixed handling of SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID.		BR
1484837	Fixed a memory leak in the authenticator.		BR

Chip Enablement

Added in 2.7.0

None.

Added in 2.6.0

Chip Family	OPNs / Boards / OPN Combinations	Supported Software Variants (if applicable)	Supported Modes	Supported Host Interfaces
EFR32xG23	• OPN: efr32fg23a010f512gm48, efr32fg23a020f512gm40, efr32zg23b010f512im48, efr32fg23b010f512im40, efr32fg23b021f512im40, efr32fg23a011f512gm40, efr32fg23b010f512gm48, efr32zg23a020f512gm48, efr32fg23b021f512im48, efr32zg23b020f512im48, efr32fg23b020f512im48, efr32fg23b020f512im40, efr32zg23b010f512im40, efr32zg23a010f512gm48, efr32fg23a010f512gm40, efr32zg23a020f512gm40, efr32fg23a021f512gm40, efr32fg23a020f512gm48, efr32zg23a010f512gm40, efr32zg23b021f512im40, efr32zg23b020f512im40, efr32zg23b011f512im40, efr32fg23b010f512im48 • Boards: brd4204d, brd4210a, brd4263c, brd4264c, brd4265b, brd4269b	N/A	RCP	UART

Application Example Changes

New Examples | Modified Examples | Removed Examples | Deprecated Examples

New Examples

Copyright © 2025 Silicon Laboratories. All rights reserved.

Added in 2.7.0

None.

Added in 2.6.0

Example Name	Description	Supported Software Variants (if applicable)	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
Wi-SUN BLE DMP - SoC Empty Application Readme	The Wi-SUN Bluetooth DMP - SoC Empty example project serves as a starting point for developing Wi-SUN and BLE DMP applications. This example can be tested using the Simplicity Connect smartphone application, available on iOS and Android.	N/A	SoC	• Parts: EFR32xG28 • Boards: BRD2705A BRD4400A BRD4400B BRD4400C BRD4401A BRD4401B BRD4401C	N/A
Wi-SUN BLE DMP - SoC SPP Application Readme	The Wi-SUN BLE DMP - SoC SPP example demonstrates how to use the Serial Port Profile (SPP) over Bluetooth Low Energy (BLE) to enable wireless serial communication between devices, providing a wireless alternative to traditional wired serial connections. This application reroutes the command line interface (CLI) over BLE, allowing users to interact with the device remotely. You can test this example using the Serial Bluetooth Terminal app for Android.	N/A	SoC	• Parts: EFR32xG28 • Boards: BRD2705A BRD4400A BRD4400B BRD4400C BRD4401A BRD4401B BRD4401C	N/A

Modified Examples

Modified in 2.7.0

None.

Modified in 2.6.0

Example Name	Changes	Supported Software Variants if applicable	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
All examples	Migrated all the examples to the new System Setup component.	N/A	N/A	• All supported parts • All supported boards	N/A
All examples	Refactored polling loops in the CoAP, TFTP Client, OTA DFU, and Ping components. Introduced a new **sl_tftp_clnt_wait_op_finished** API to wait for the operation finished event flag.	N/A	N/A	• All supported parts • All supported boards	N/A
All examples	Enabled RTS/CTS flow control in the applications.	N/A	N/A	• All supported parts • All supported boards	N/A

Copyright © 2025 Silicon Laboratories. All rights reserved.

Example Name	Changes	Supported Software Variants if applicable	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
All examples	Introduced new CLI command configurating RTT Trace level and forwarding the content of RTT traces.	N/A	N/A	• All supported parts • All supported boards	N/A
Wi-SUN - SoC Network Measurement	Added support for displaying estimated hop count to the Border Router in the measurement output.	N/A	N/A	• Parts: EFR32FG25 • Boards: BRD2705A BRD4270A BRD4270B BRD4271A BRD4272A BRD4276A	N/A
Wi-SUN - SoC Socket application	Updated socket_read CLI command for UDP. Removed the misleading read size argument. The whole received buffer should be read, this argument is unnecessary.	N/A	N/A	• All supported parts • All supported boards	N/A
OTA DFU component	Increased SL_WISUN_OTA_DFU_TFTP_TIMEOUT_SEC to 8 seconds in OTA DFU config	N/A	N/A	• All supported parts • All supported boards	N/A
OTA DFU component	Fixed TFTP Error code typos and added additional debug logs to the error handling in OTA DFU component.	N/A	N/A	• All supported parts • All supported boards	N/A

Removed Examples

None.

Deprecated Examples

None.

Known Issues and Limitations

ID	Issue or Limitation Description	Workaround (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
	Wi-SUN Configuration is only capable of provisioning routers using the new Dynamic Data Provisioning methodology. The generation of border router certificates is not supported.	Use the previous provisioning methodology.	All supported parts and boards
1119464	Packets sent when FSK FEC is enabled can infringe ARIB regulation.	Until this problem has been addressed correctly, the stack will refuse to start if both ARIB enforcement and FEC are enabled.	All supported parts and boards

Copyright © 2025 Silicon Laboratories. All rights reserved.

ID	Issue or Limitation Description	Workaround (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1067978	Packets sent using a PHY with a bandwidth larger than the base PHY's bandwidth can infringe ARIB regulation.	Limit the communications when using mode switch with ARIB enforcement enabled.	EFR32FG25
1176014	FG25 asserts with a RAIL_ASSERT_FAILED_RTCC_SYNC_STALE_DATA error code when entering EM2.	Do not allow the power manager to go to EM2 by adding a requirement on EM1 in the application.	EFR32FG25
1373879	The RCP is sometimes unable to send packets using mode switch.	None	EFR32FG25
	Throughput is reduced compared to previous releases due to the fix for 1445797. The previous CSMA slot duration was too short, which artificially increased throughput. The impact is more noticeable with high-throughput modulations. Further investigation will be run to better understand and improve the current performances.	None	EFR32FG25
	Wi-SUN RCP operation on EFR32FG23 requires hardware flow control to be enabled. Without hardware flow control, CPC requires an additional buffer, which exceeds the available RAM on this device.	None	EFR32FG23

Impact of Release Changes

Impact Statements | Migration Guide

Impact Statements

None.

Migration Guide

Previous versions of "Wi-SUN - LFN CLI" were using SL_POWER_MANAGER_EVENT_TRANSITION_ENTERING_EM3 that was removed in that version 2025.6 of the SiSDK. Applications generated with an older release of the SiSDK will not compile against version 2025.6. All references to SL_POWER_MANAGER_EVENT_TRANSITION_ENTERING_EM3 can be safely deleted.

Using This Release

What's in the Release?

This release contains the following:

- Wi-SUN stack library
- Wi-SUN sample applications
- Wi-SUN border router and RCP pre-compiled demos
- Documentation

If you are a first-time user, see the Wi-SUN Getting Started Guide.

Compatible Software

Software	Compatible Version or Variant
Wi-SUN Linux Border Router (wsbrd)	v2 and more recent.
Wi-SUN Linux Router (wsrd)	v2.2 and more recent.

Installation and Use

Copyright © 2025 Silicon Laboratories. All rights reserved.

The Wi-SUN SDK is distributed as part of the Simplicity SDK, Silicon Labs' unified software development kit suite. To get started quickly, install Simplicity Studio 5, which provides a complete development environment, including project and resource management, software configuration tools, a full IDE with GNU toolchain, and analysis utilities. Step-by-step installation instructions are available in the Simplicity Studio 5 User's Guide.

Alternatively, you can manually install the Simplicity SDK by downloading or cloning the latest release from GitHub.

By default, Simplicity Studio installs the Simplicity SDK to:

- **Windows:** `C:\Users\<NAME>\SimplicityStudio\SDKs\simplicity_sdk`
- **macOS:** `/Users/<NAME>/SimplicityStudio/SDKs/simplicity_sdk`

Documentation for each SDK version is included with the SDK installation.

For information about Secure Vault Integration, see Secure Vault.

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

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

Help and Feedback

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

> **Note: Ask AI** is experimental.

- Get help from our developer community.

SDK Release and Maintenance Policy

See our SDK Release and Maintenance Policy.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

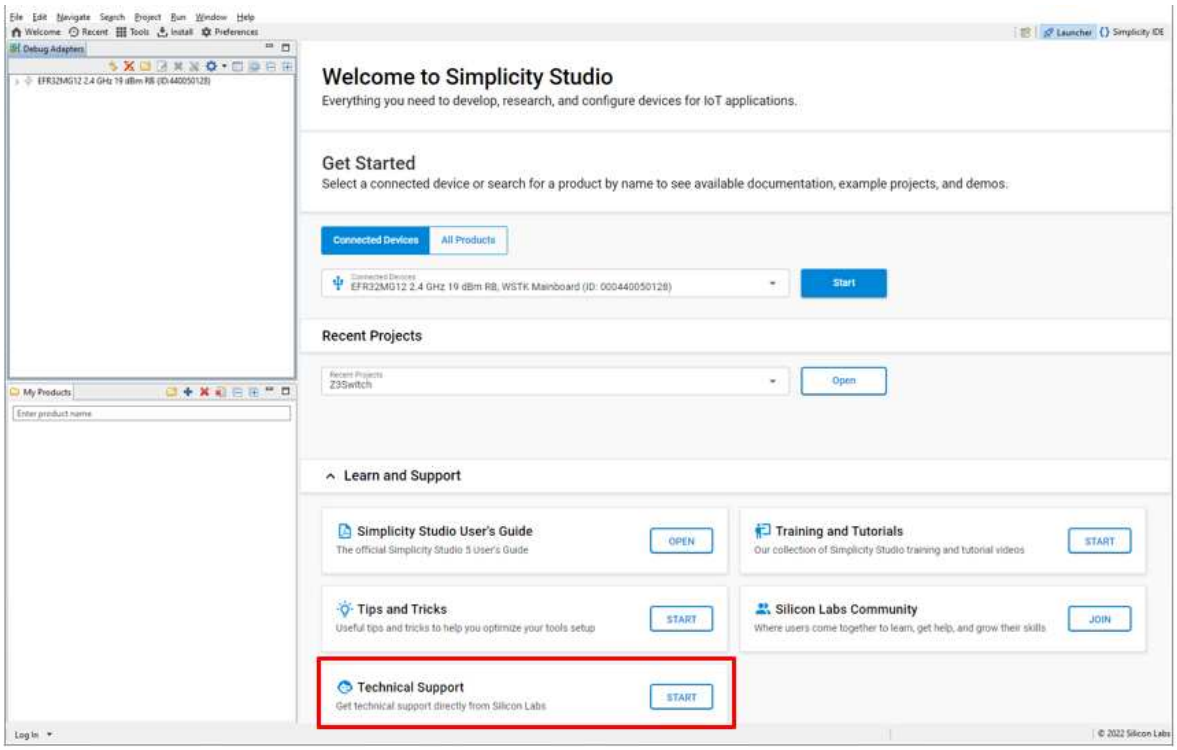
Install Simplicity Studio 5 (SSv5), which will set up your development environment and walk you through GSDK installation. Installation instructions are provided in the Simplicity Studio 5 online User's Guide. This is the recommended method for getting started with development.

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

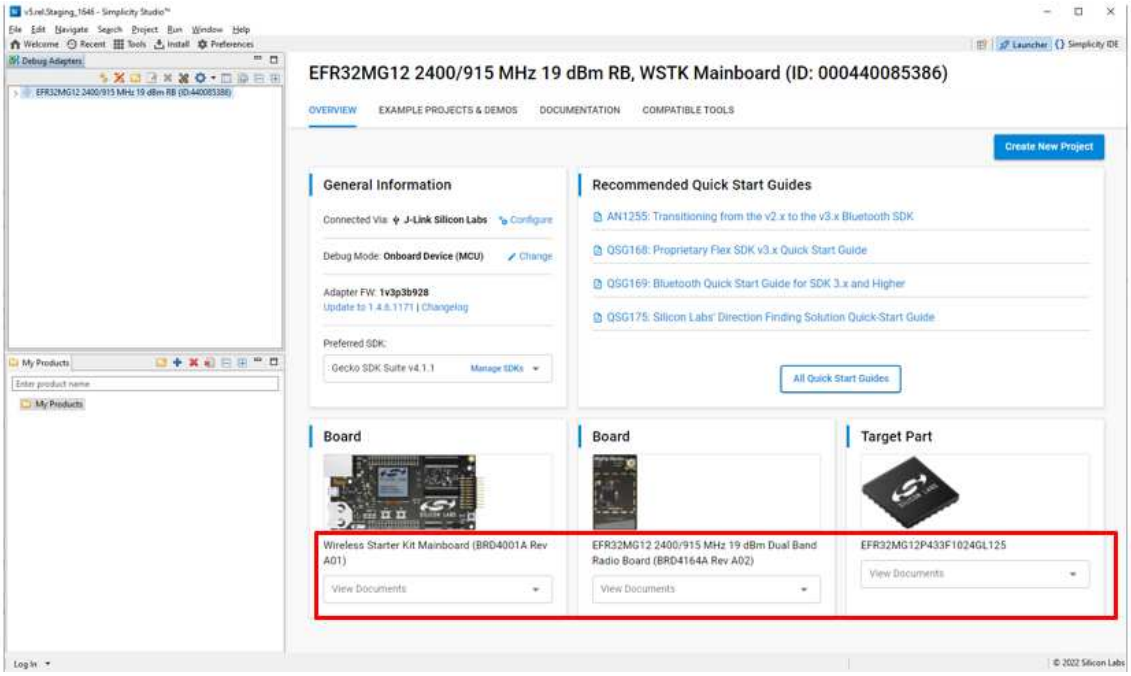
Access the Silicon Labs support portal at https://www.silabs.com/support through SSv5's Welcome view under **Learn and Support**. Use the support portal to contact Customer Support for any questions you might have during the development process.



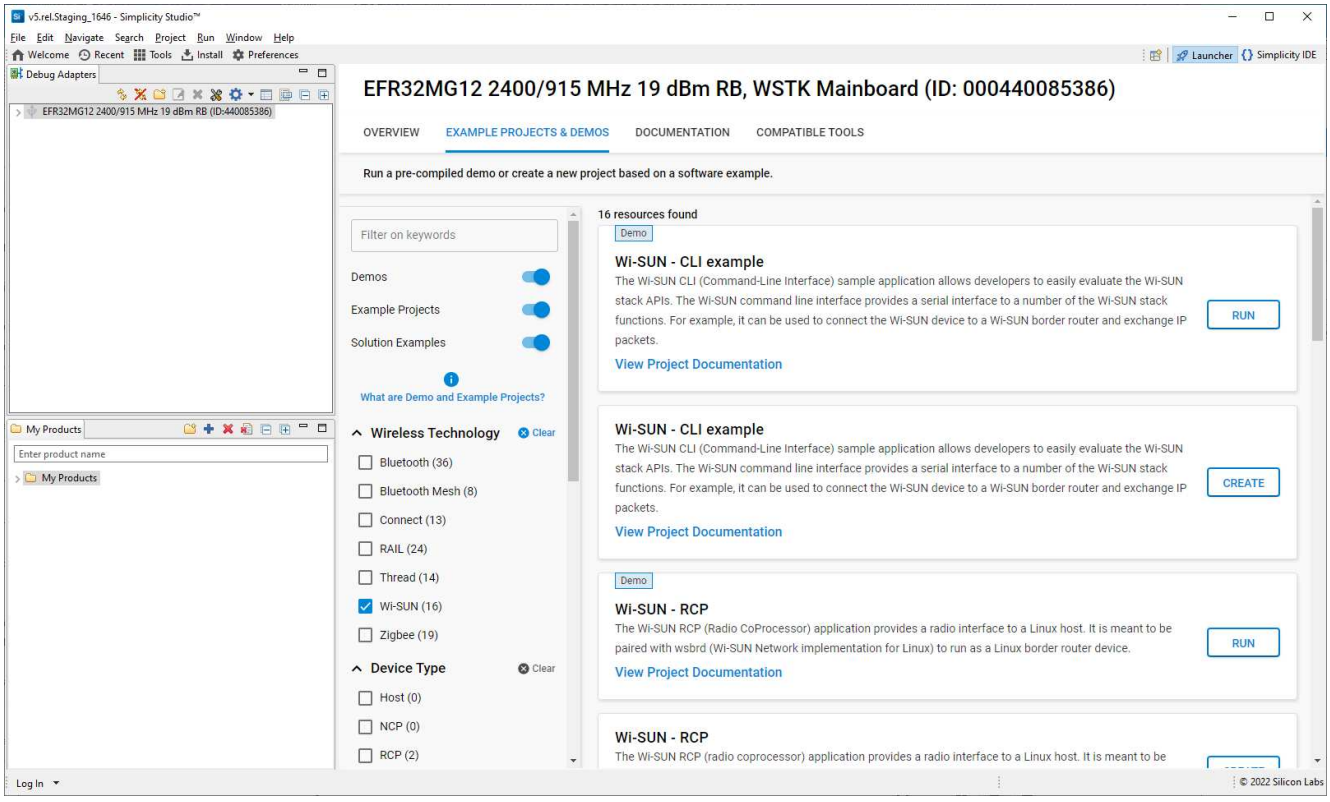
Documentation

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

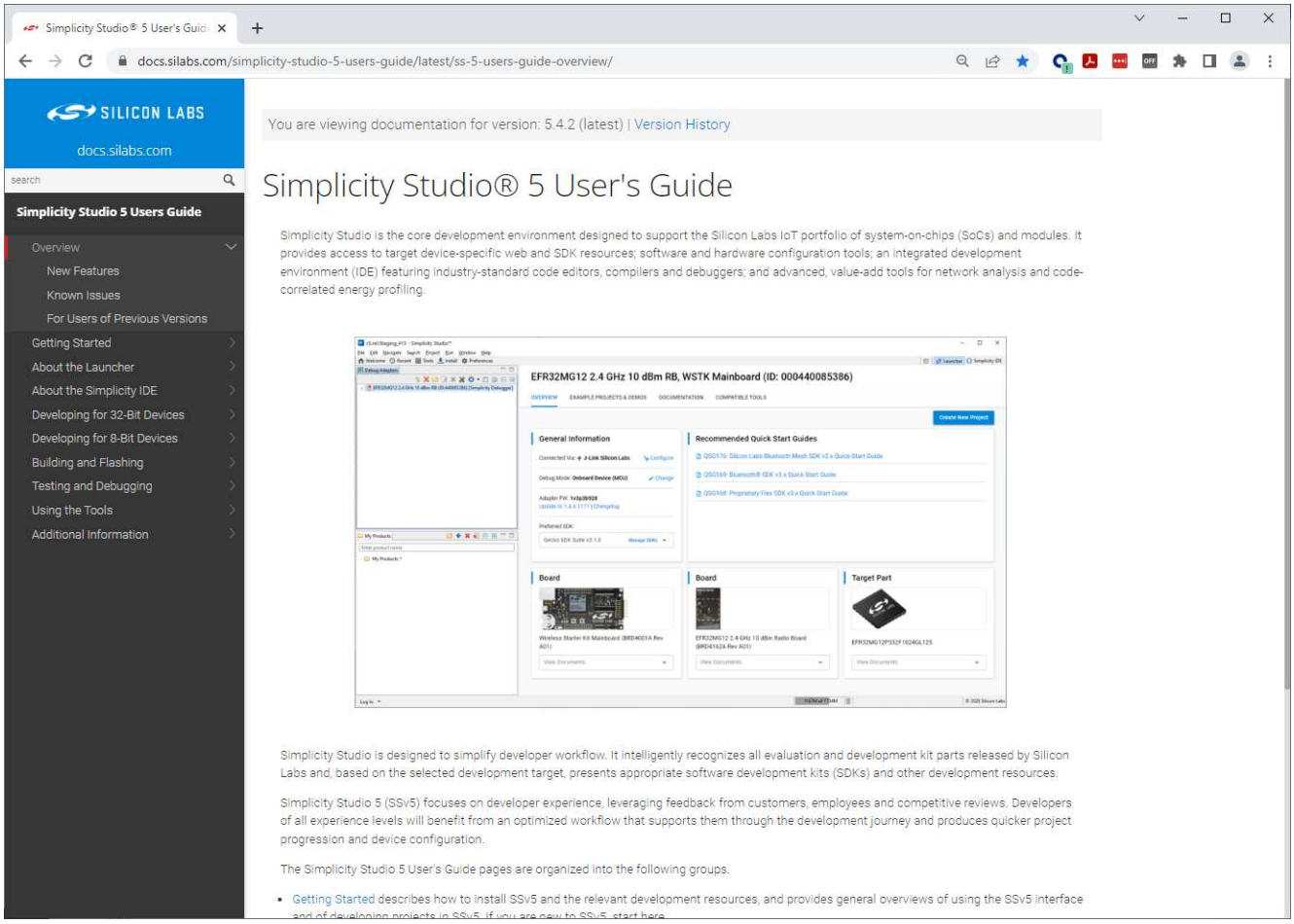


SDK documentation and other references are available through the **DOCUMENTATION** tab. Filter with the **Wi-SUN Technology Type** checkbox to see documentation most closely related to the Wi-SUN SDK. To see documents specific to Wi-SUN, select **Wi-SUN** under **Wireless Technology**.



Copyright © 2025 Silicon Laboratories. All rights reserved.

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



Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN Sample Applications

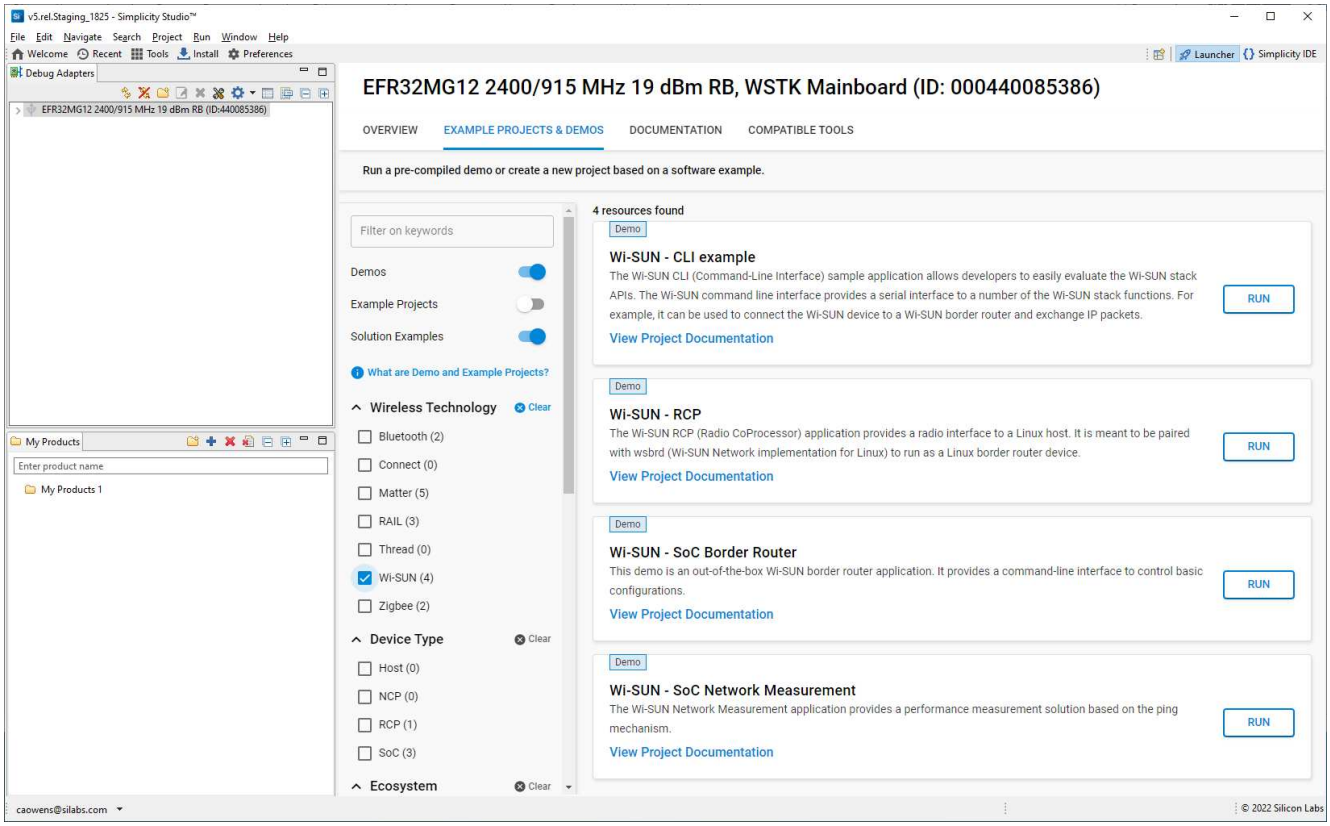
About Sample Applications and Demos

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

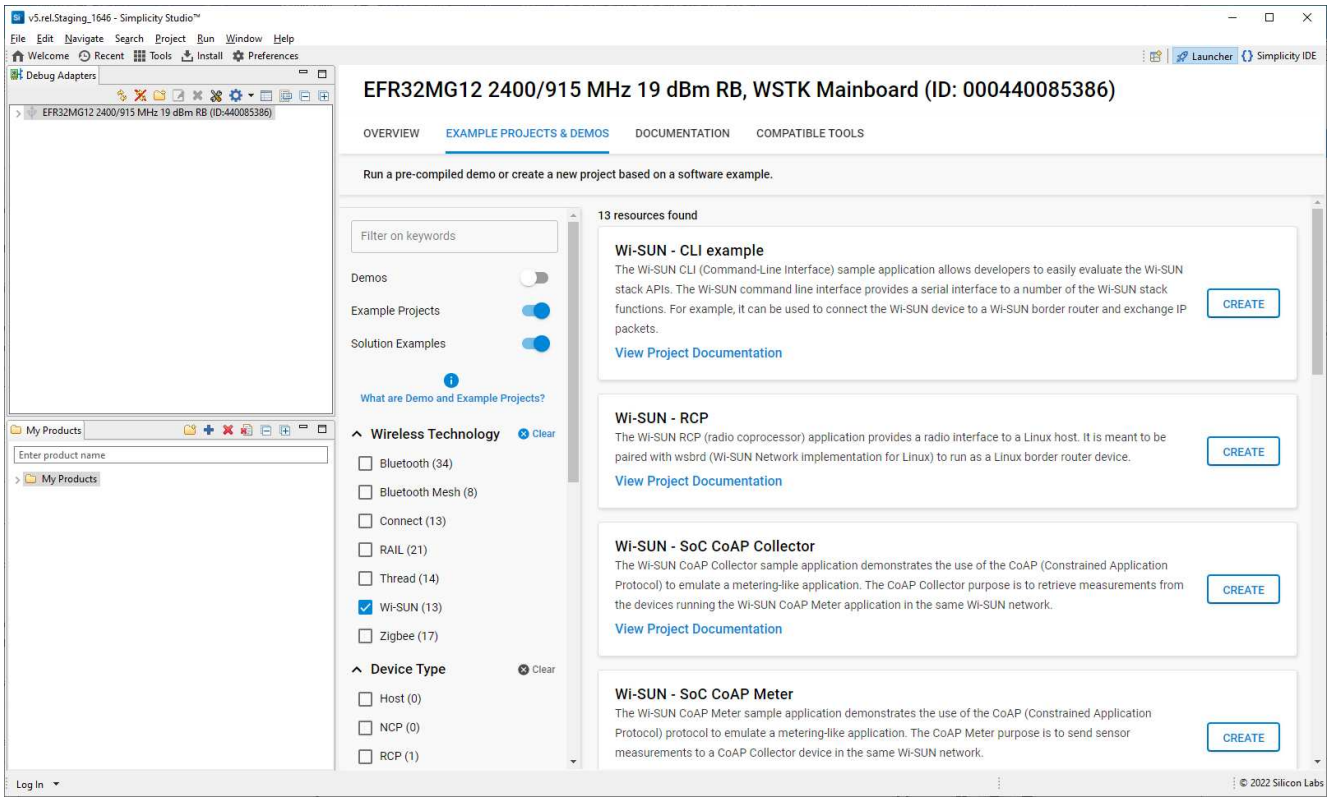
When you work with examples, the process is:

- Select an example
- Change the example configuration (if needed)

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

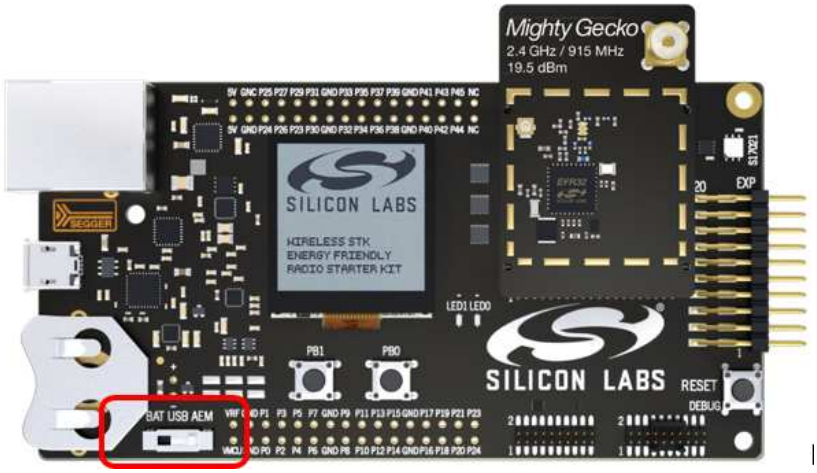
When working with an example application in Simplicity Studio, you will be executing the steps in the following order:

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.

> 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.

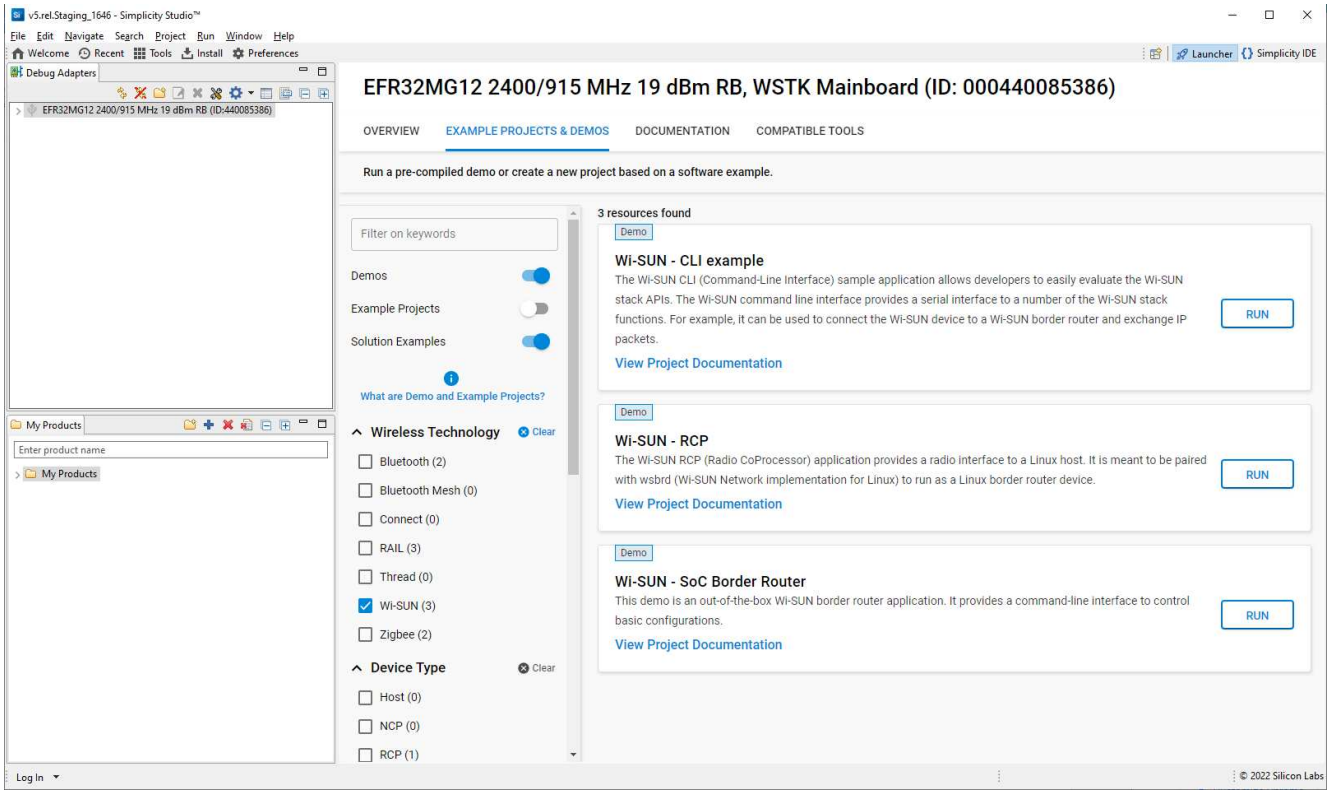
Note: Make sure your mainboard has the latest “Adapter FW” to avoid any issue when using the example CLI. To do so:

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.

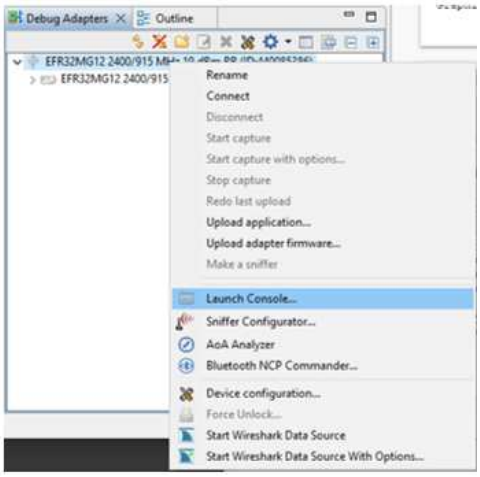
Copyright © 2025 Silicon Laboratories. All rights reserved.

3. If Simplicity Studio 5 proposes to update the firmware, do so.

The Wi-SUN SoC Border Router demonstration is delivered in a binary format for an out-of-the-box experience. The implementation scales for a production-grade SoC Border Router maintaining several hundreds of Wi-SUN nodes. For a production-grade Linux border router solution, refer to the Wi-SUN Linux 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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
> wisun start_fan11
```

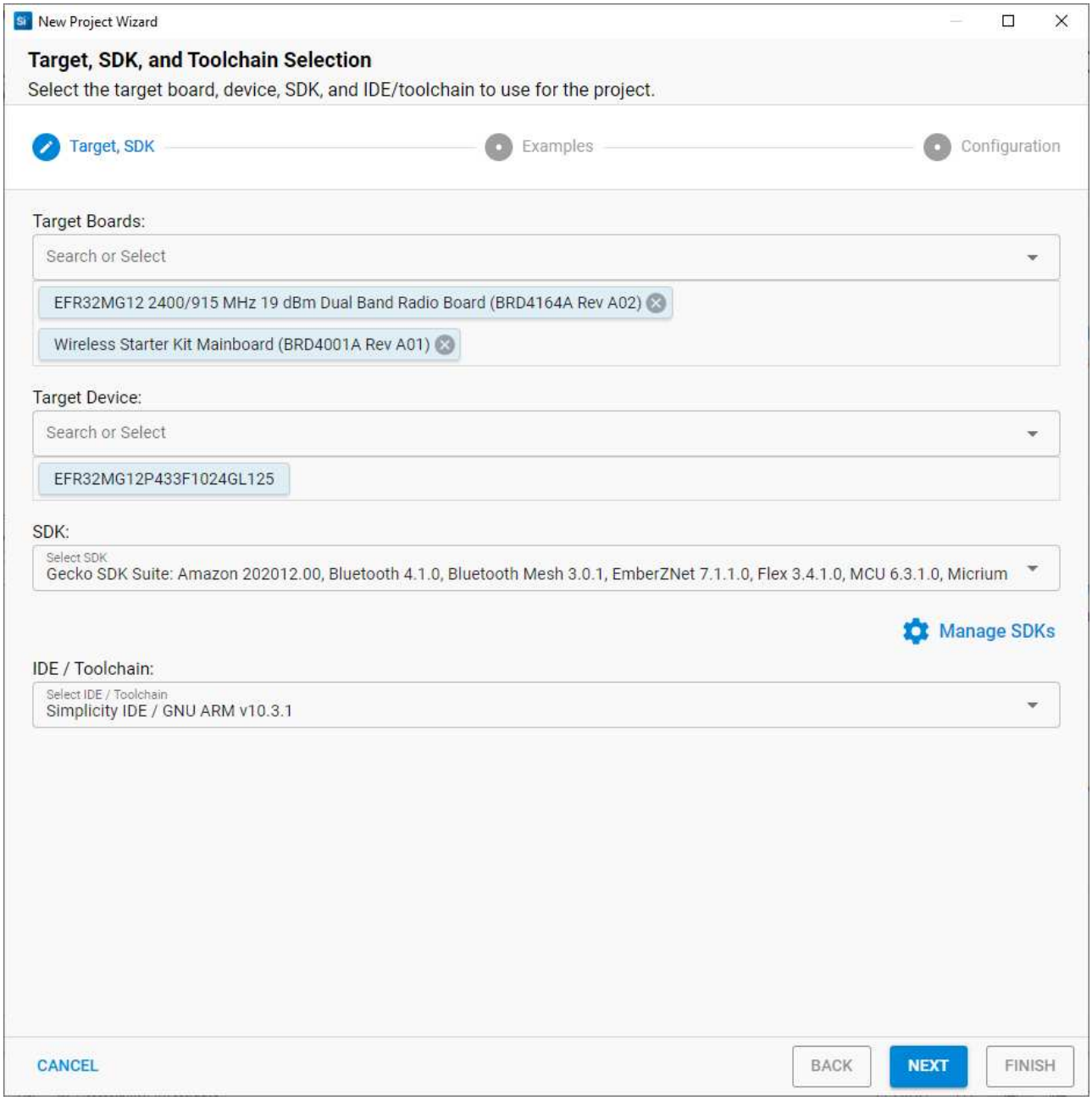
Or with default FAN 1.0 PHY using the following command:

```
> wisun start_fan10
```

Creating a Project Based on an Example

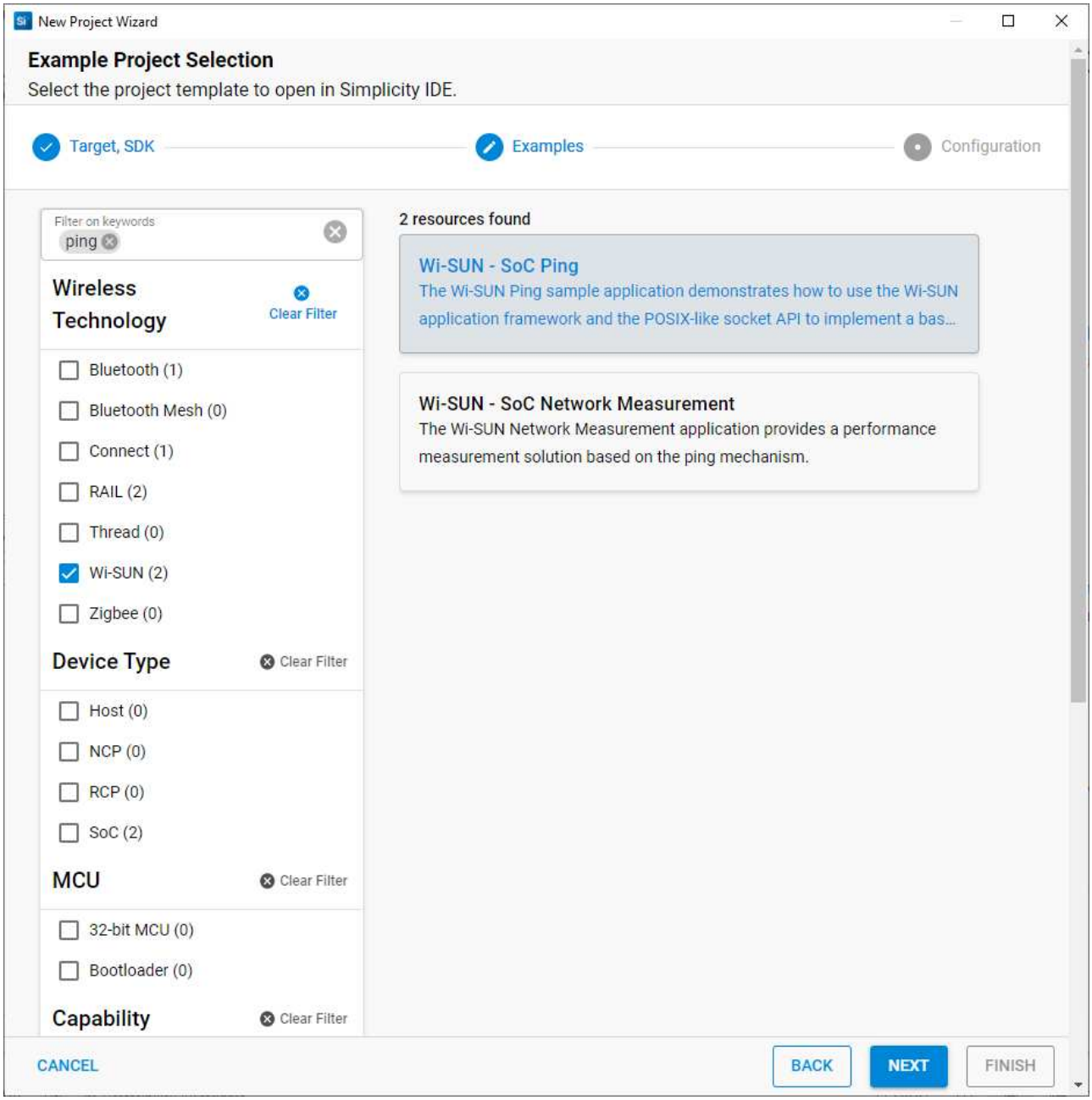
Simplicity Studio 5 (SSv5) offers a variety of ways to begin a project using an example application. The online Simplicity Studio 5 User's Guide describes them all. This guide uses the **File > New > Silicon Labs Project Wizard** method because it takes you through all three of the Project Creation Dialogs.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



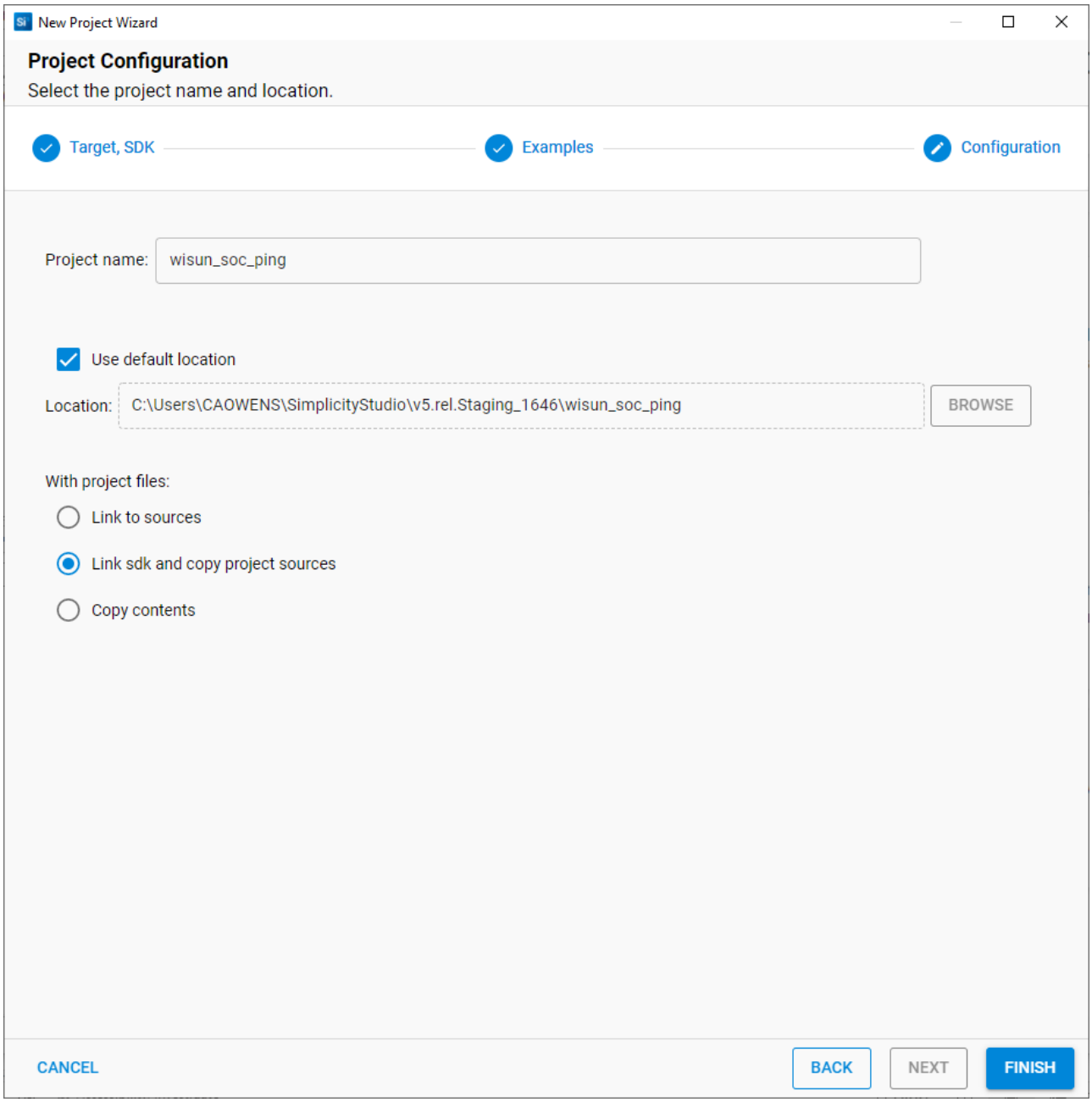
Copyright © 2025 Silicon Laboratories. All rights reserved.

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**.



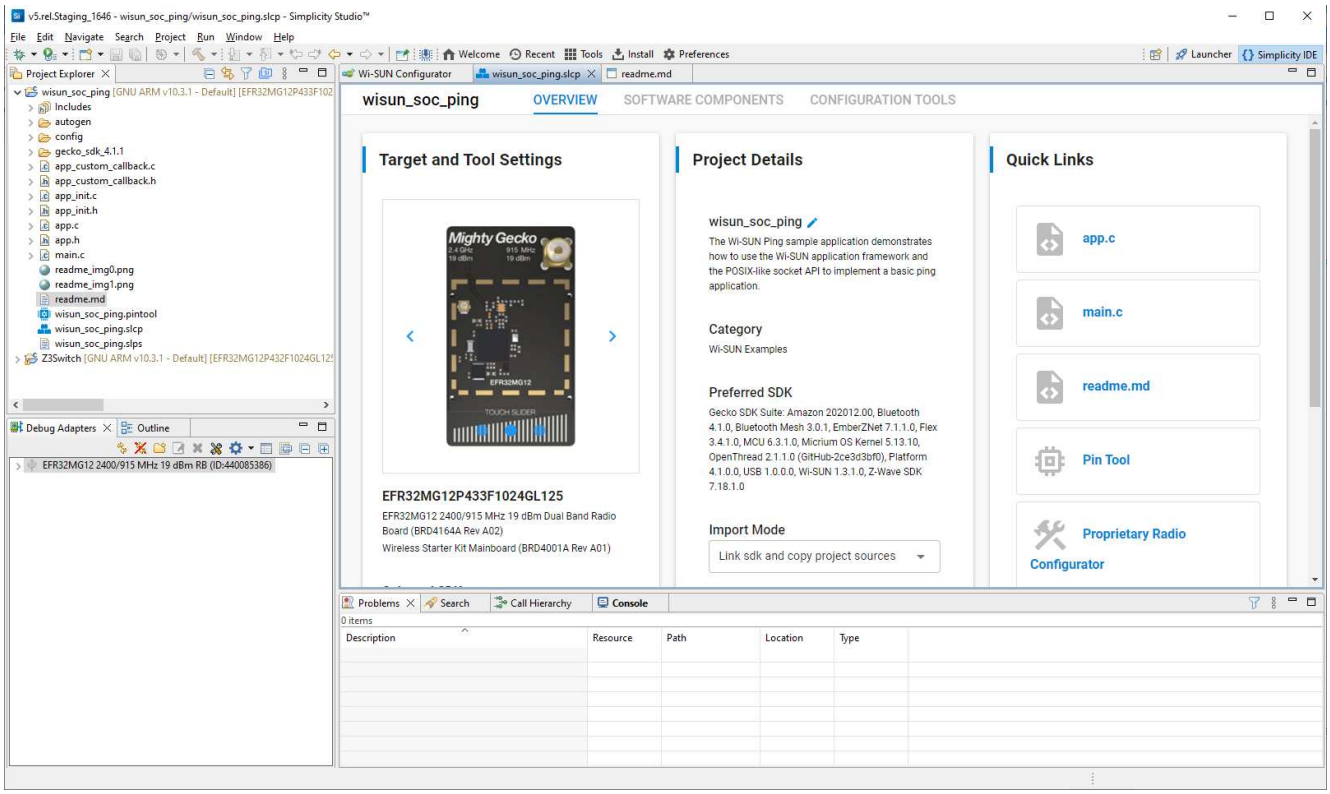
4. The Project Configuration dialog opens. Here you can rename your project, change the default project file location, and determine if you will link to or copy project files. Note that if you change any linked resource, it is changed for any other project that references it. Click **FINISH.**

Copyright © 2025 Silicon Laboratories. All rights reserved.



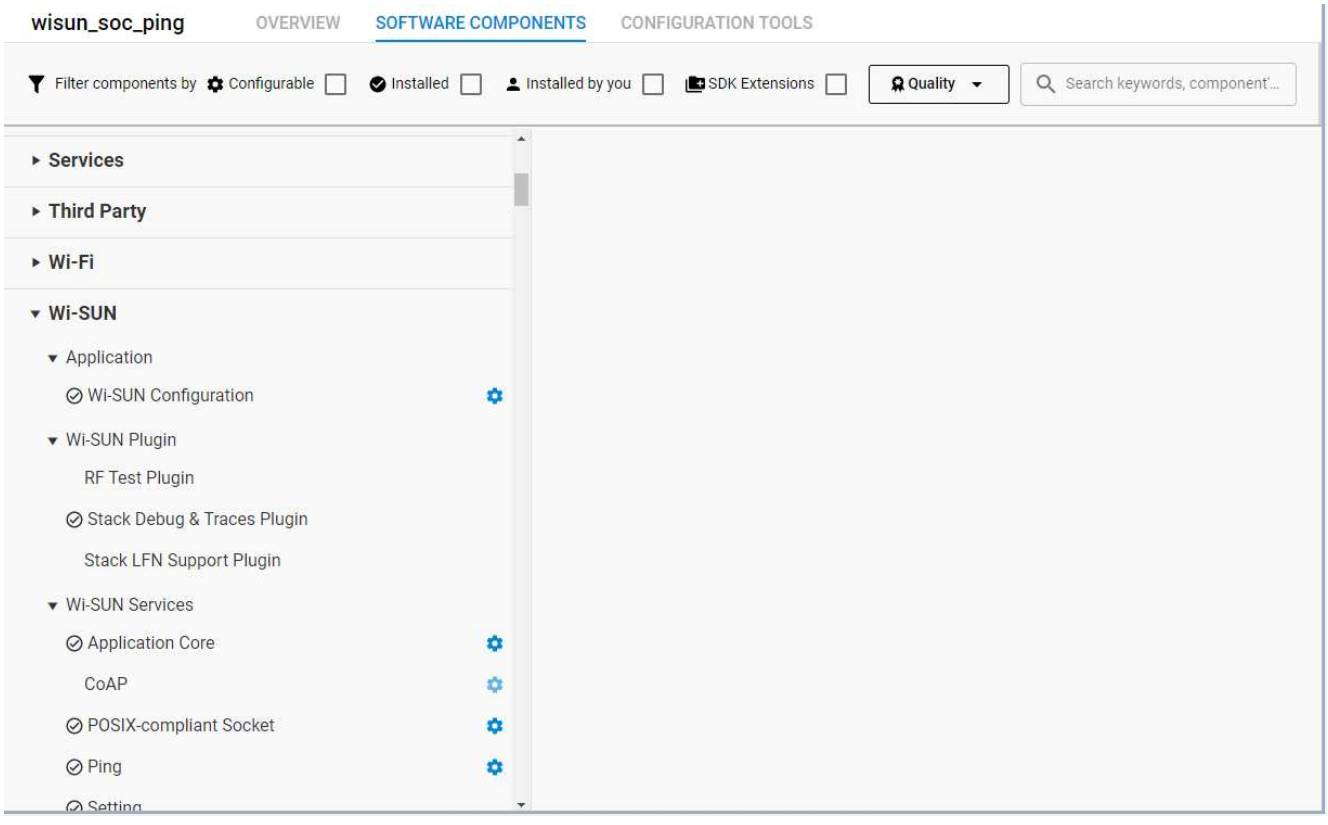
5. The Simplicity IDE Perspective opens with the project documentation (readme.md). Click the <project>.slcp tab to see the Project Configurator OVERVIEW tab. See the online Simplicity Studio 5 User's Guide for details about the functionality available through the Simplicity IDE perspective and the Project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

Copyright © 2025 Silicon Laboratories. All rights reserved.



Configuring the Project

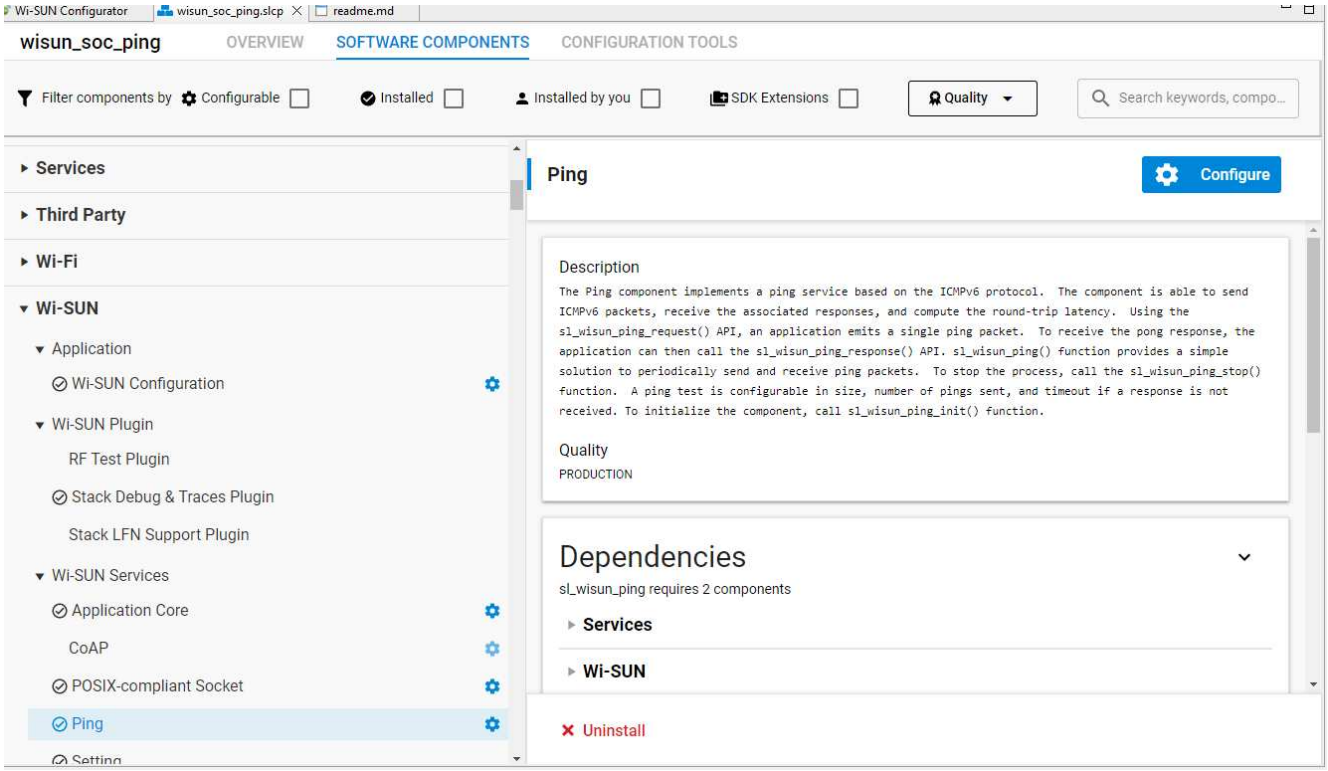
Silicon Labs Wi-SUN applications are built on a Gecko Platform component structure. Click the SOFTWARE COMPONENTS tab to see a complete list of component categories.



Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

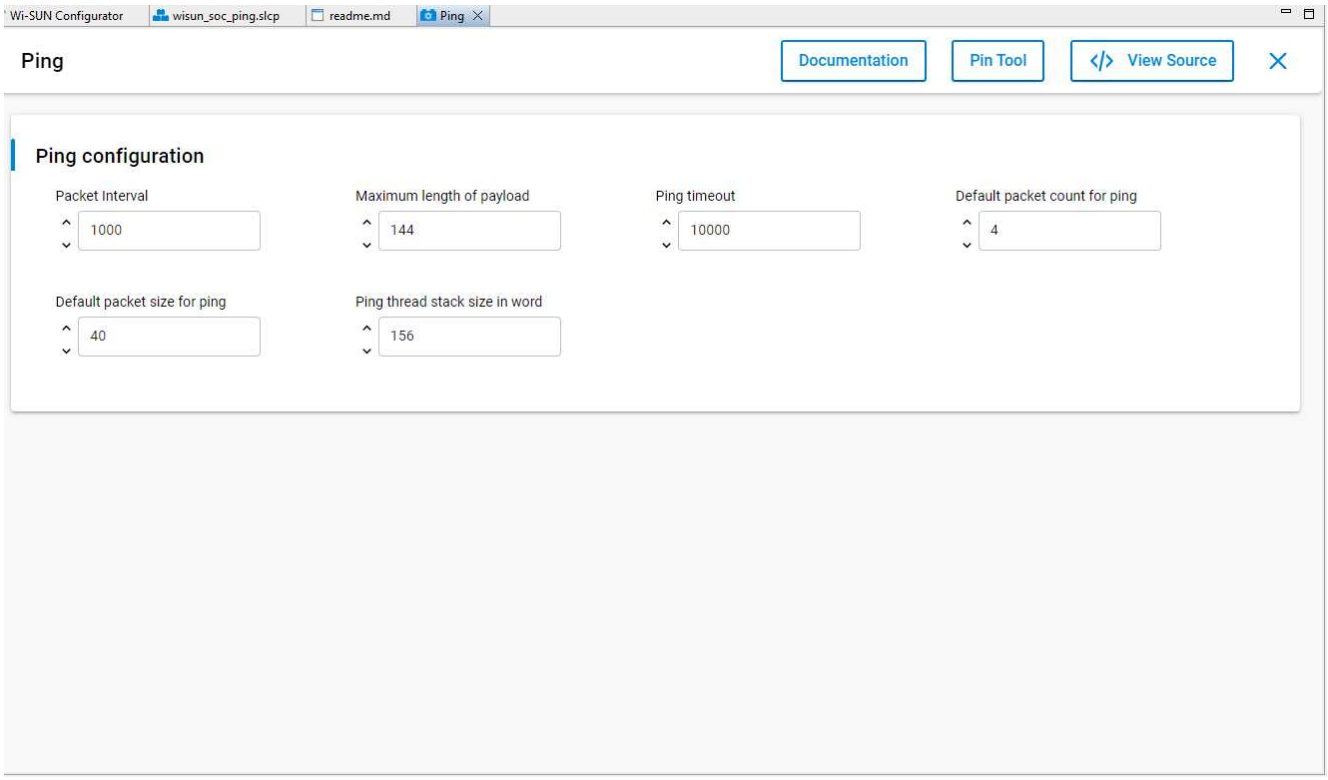
Configurable components have a gear symbol on their right. Select a component to see information about it.



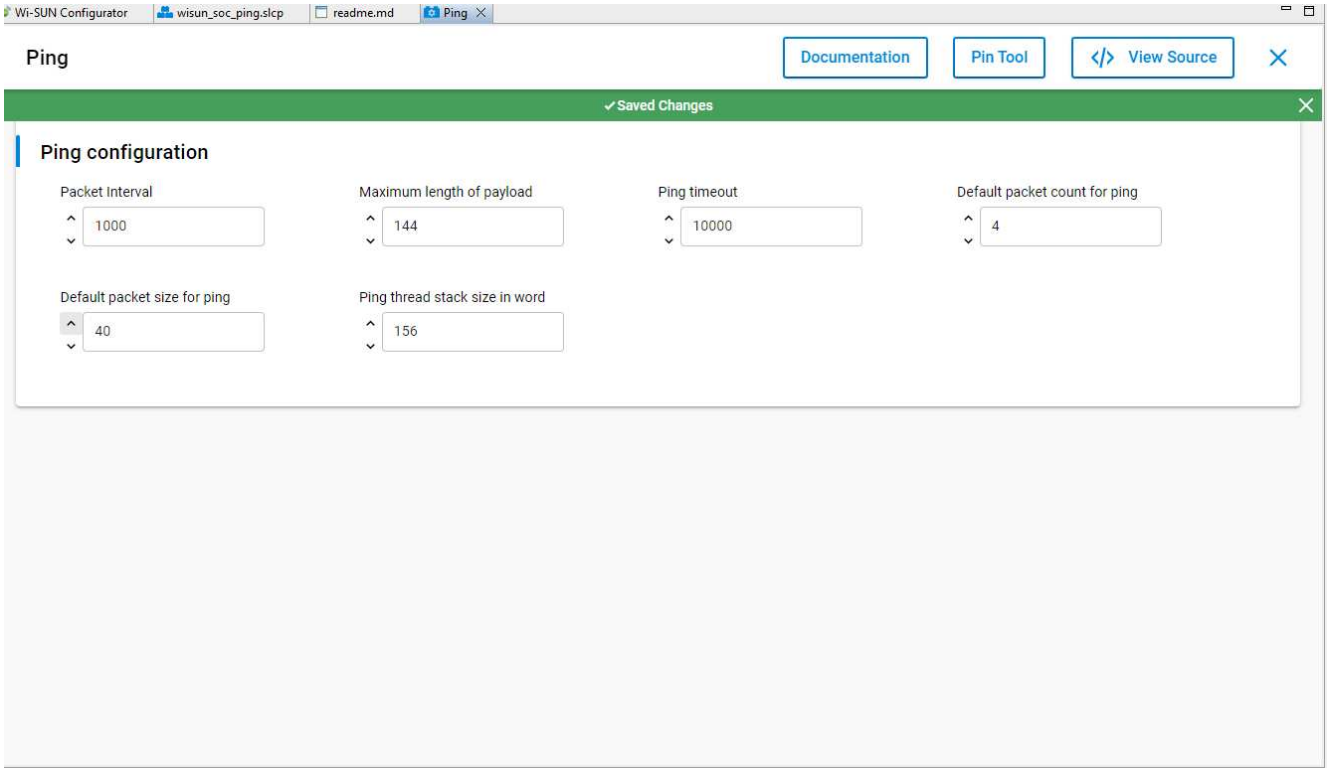
If the component is configurable, click **CONFIGURE** to open the Component Editor in a new tab.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

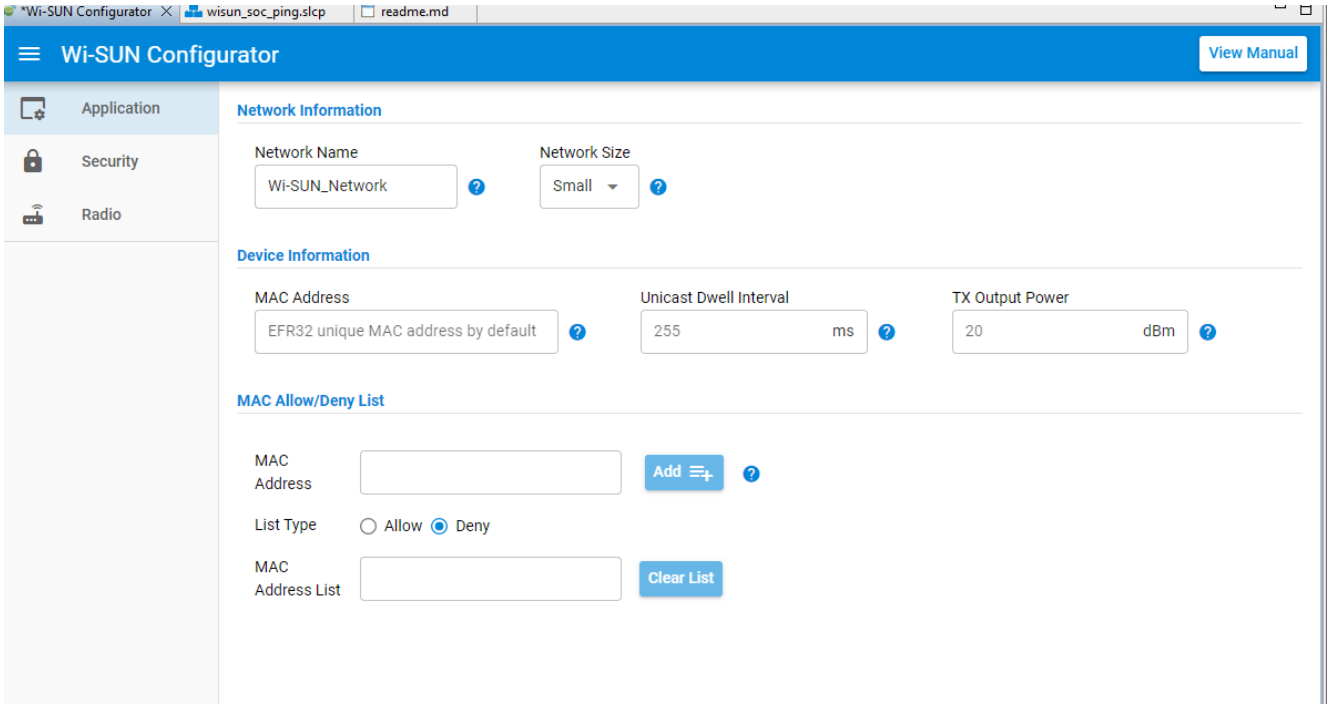


Any changes you make are autosaved, and project files are autogenerated.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



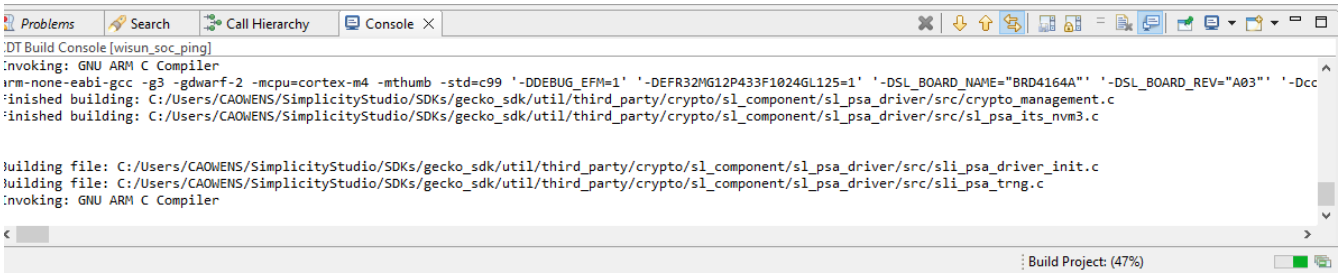
If you make changes, the Wi-SUN Configurator tab has an asterisk to the left. Save changes (CTRL-S) when you are finished updating Wi-SUN Configurator settings. If you build the project without saving changes, the changes are saved automatically.

Building the Project

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

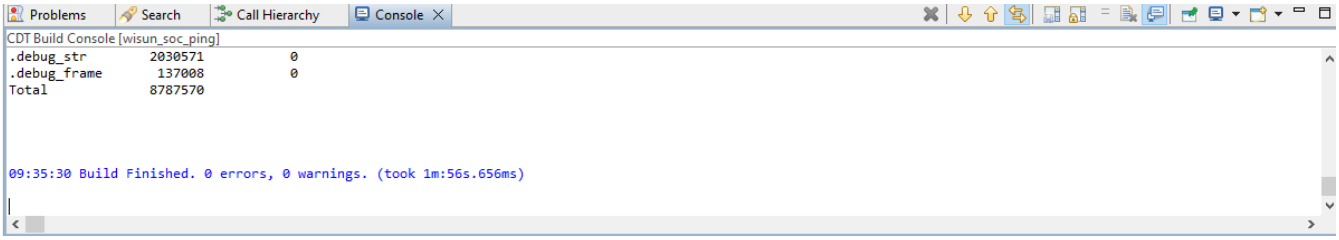
Automatically Compile and Flash

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



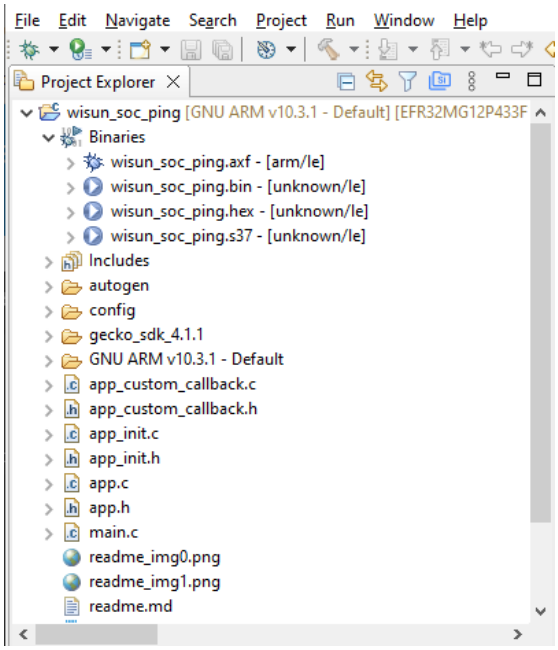
3. When building and flashing are complete a Debug perspective is displayed. Click **Resume** ( ) to start the application running on the WSTK.

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

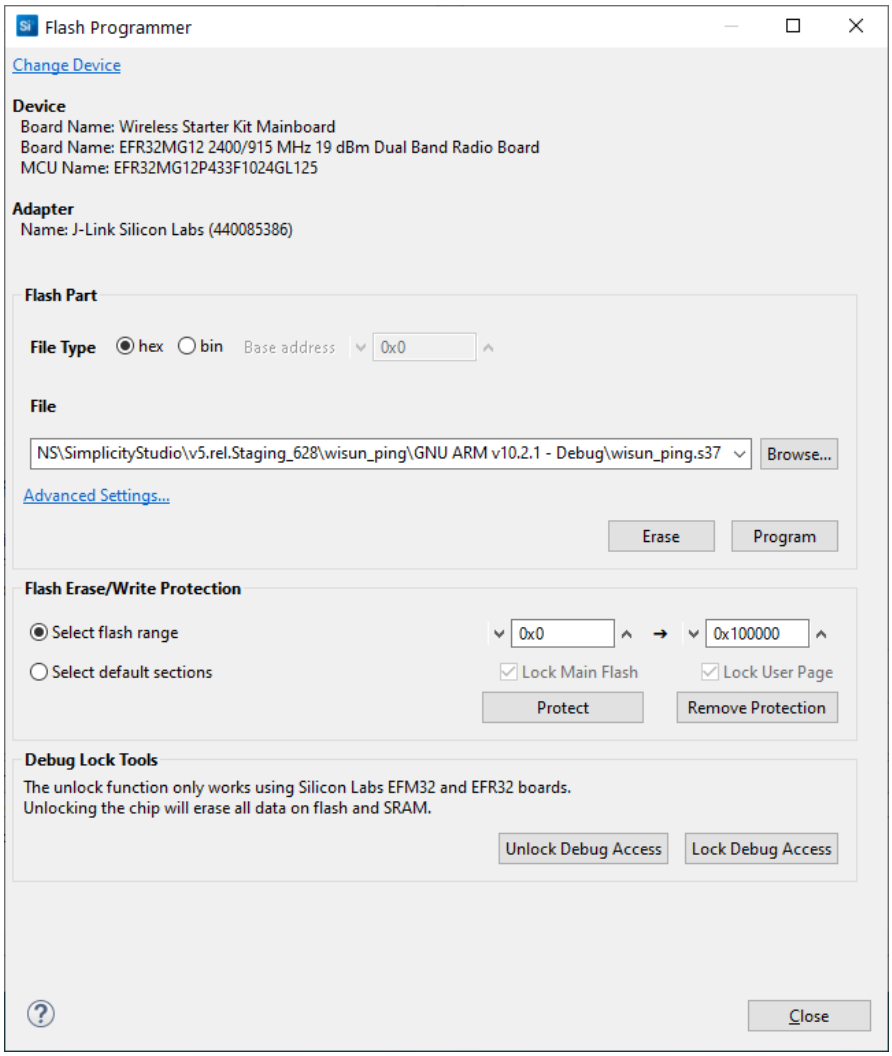
1. After you generate your project files, instead of clicking Debug, click **Build** ( ) in the top tool bar.
2. You can load the binary image through Project Explorer view.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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 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 Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) or Silicon Labs Gecko Bootloader User's Guide for Series 3 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 the *GBL File Creation* section of the Simplicity Commander Reference Guide. The

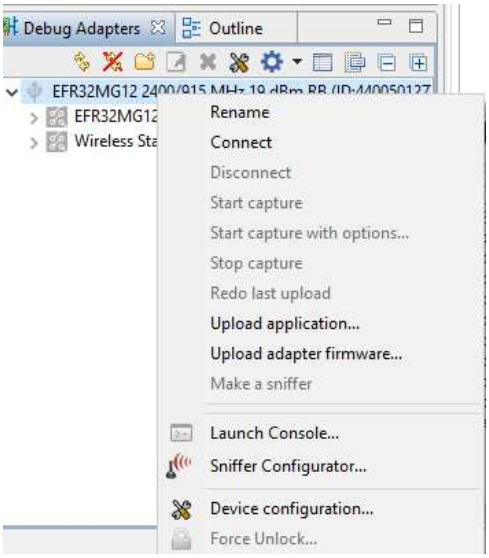
Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

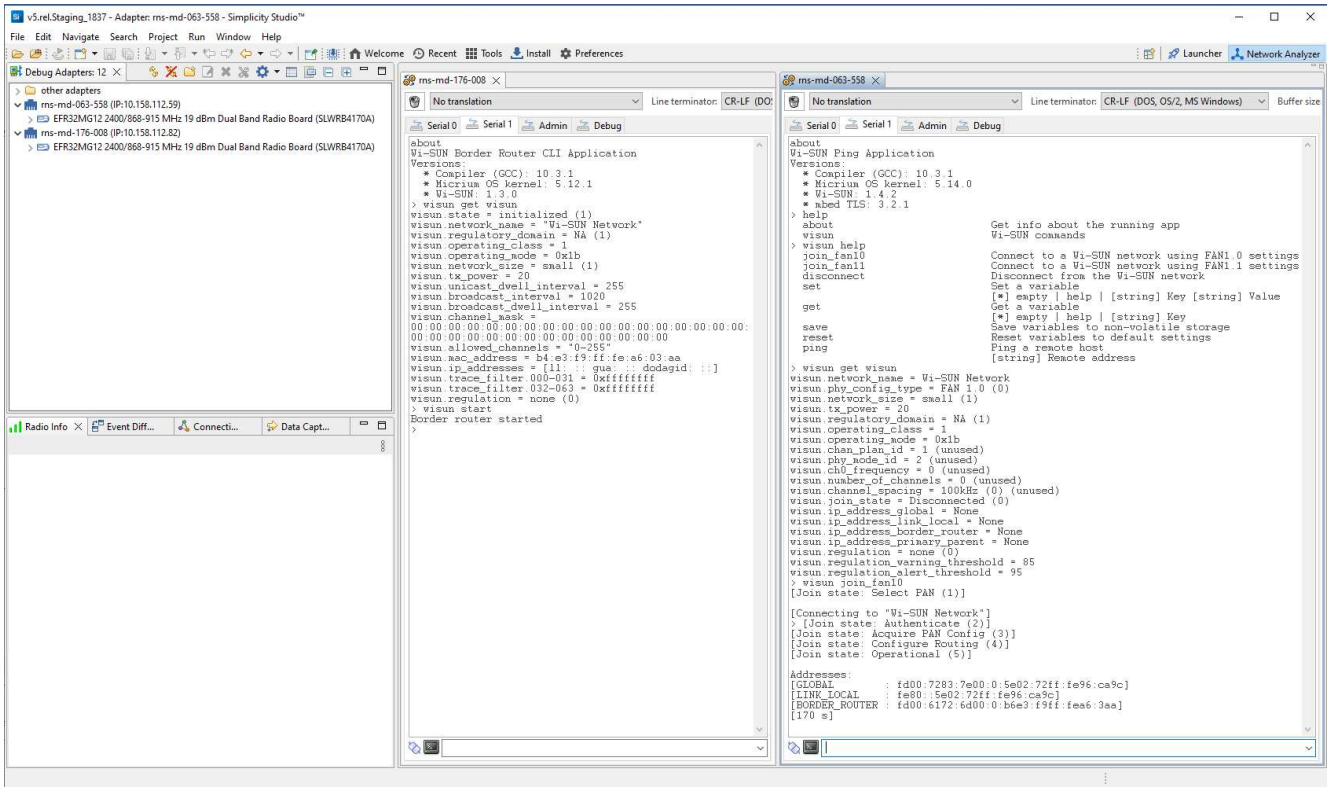
Connect the Wi-SUN Ping WSTK

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

`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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

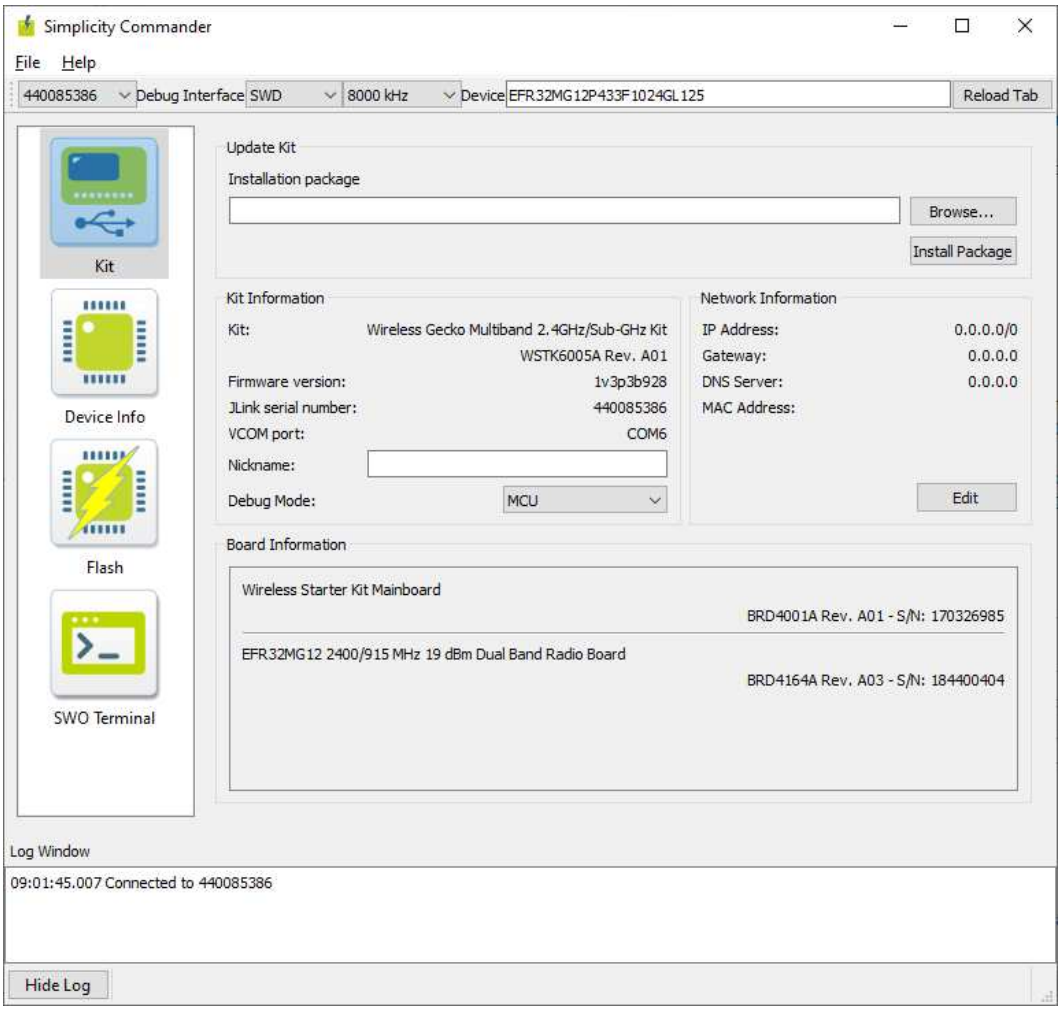
Gecko Platform

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 Reference Guide** provides more information.

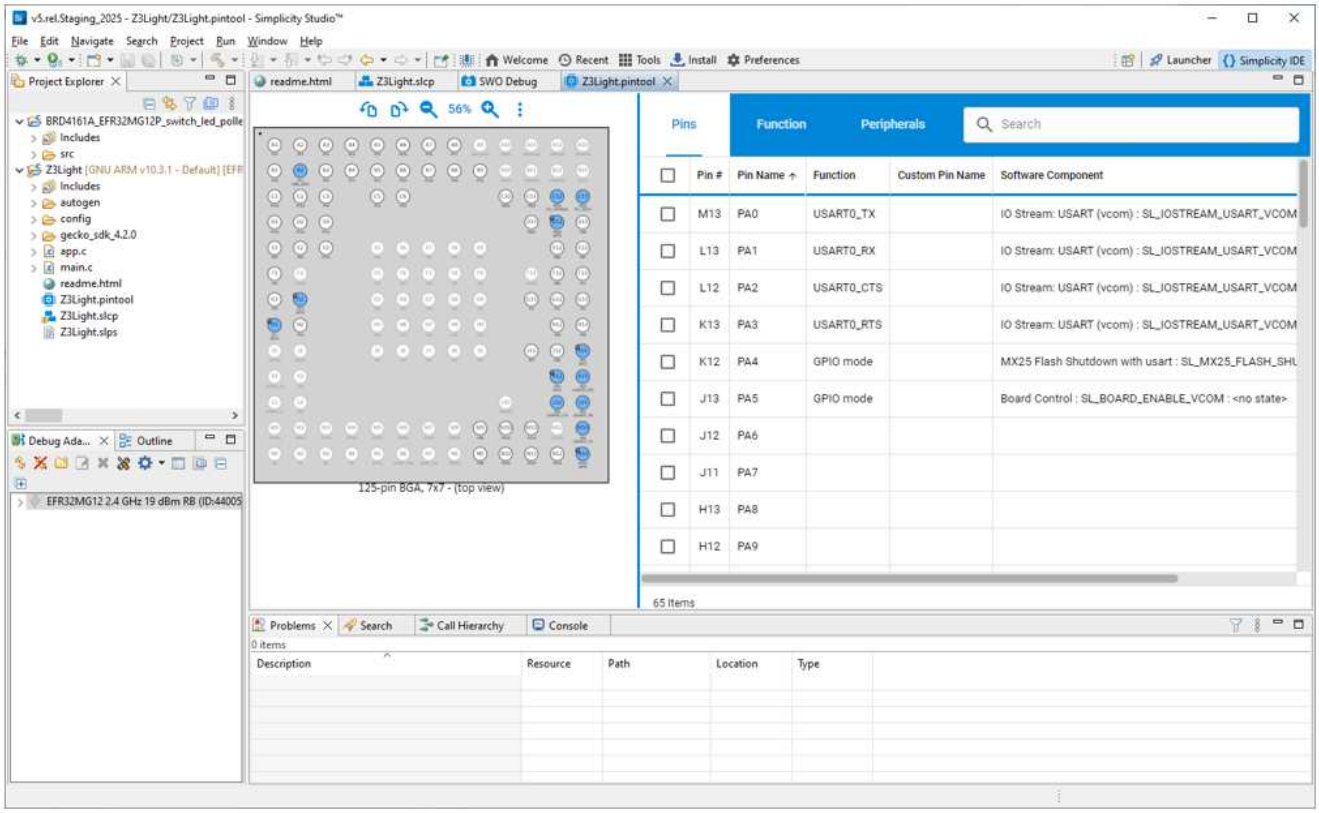
Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

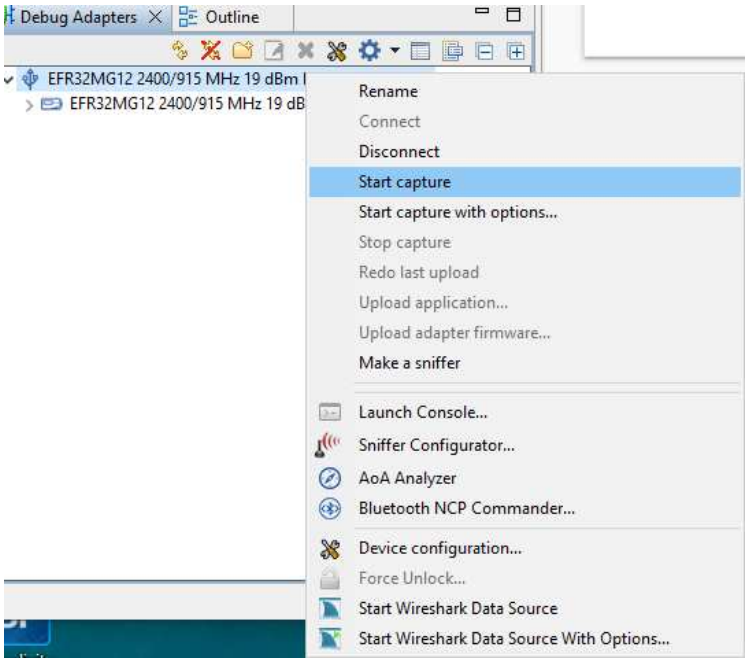
Network Analyzer

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.

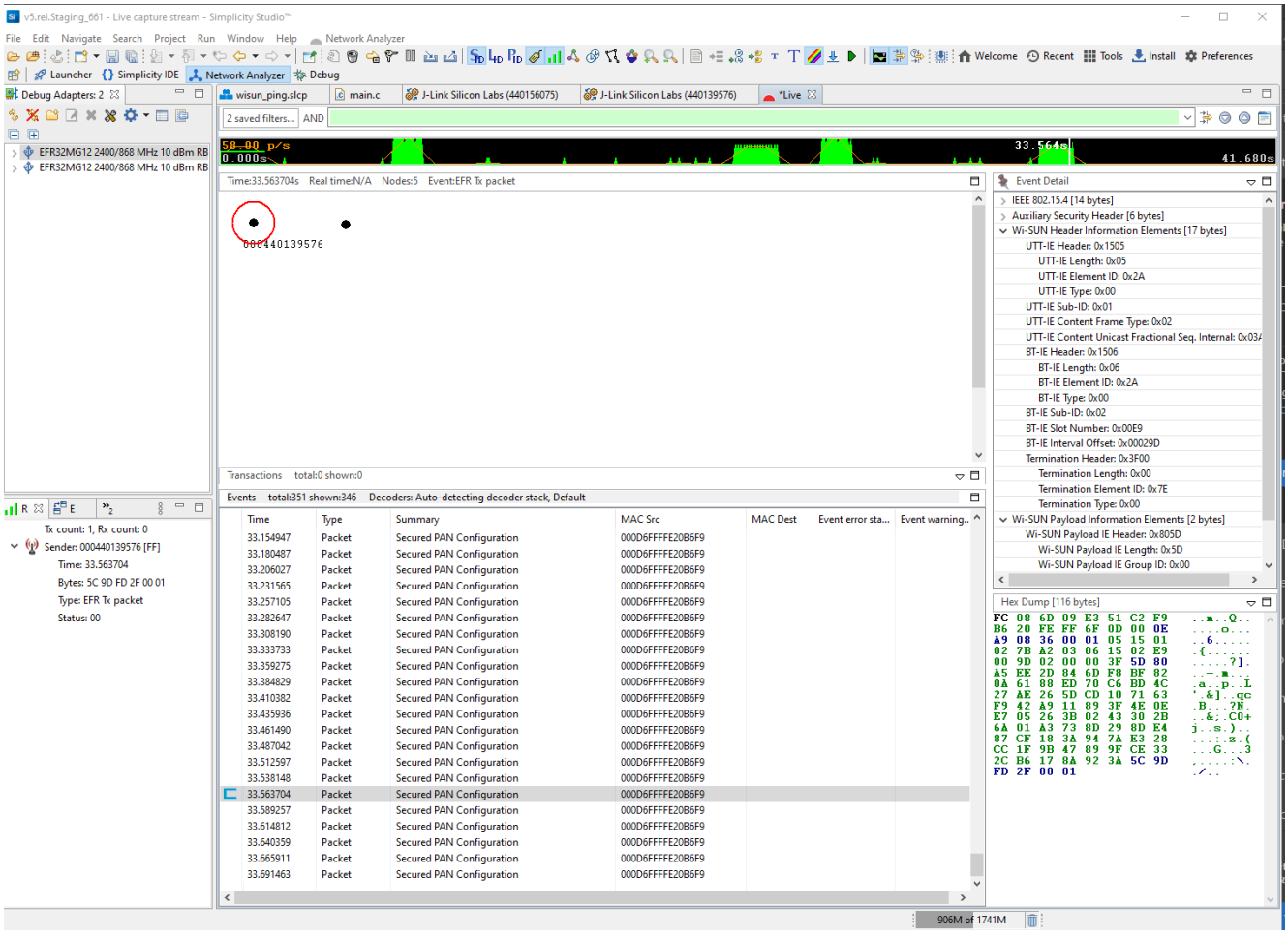
The Packet Trace application captures the packets directly from the Packet Trace Interface (PTI) available on the Wireless Gecko SoCs and modules. It therefore provides a more accurate capture of the packets compared to air-based capture.

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**.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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).



Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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**

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Build and Connect

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

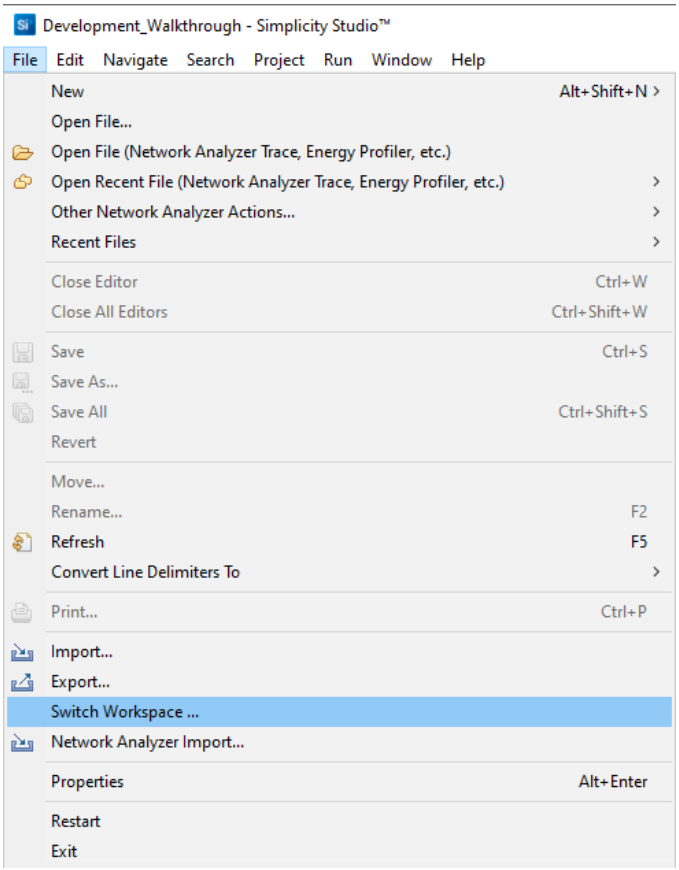
The Development Walkthrough begins with Simplicity Studio.

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.

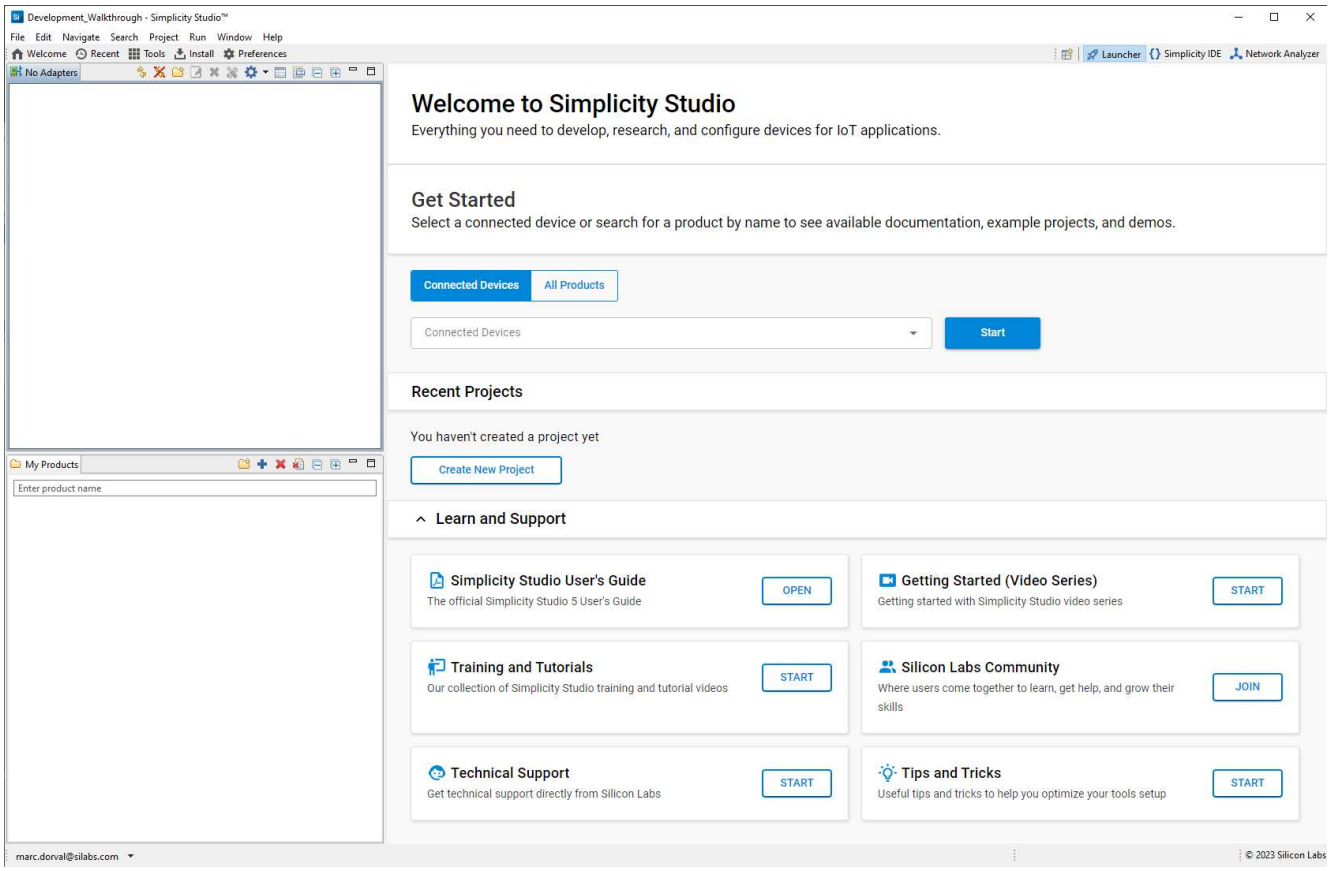
In Windows, the default path for Simplicity Studio workspaces is `C:\Users\username\SimplicityStudio`, so create `C:\Users\username\SimplicityStudio\Development_Walkthrough`.

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



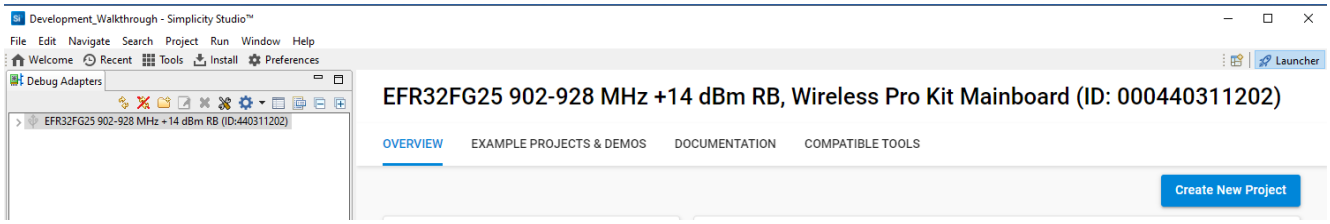
Once in the new workspace, you see the Simplicity Studio Welcome Screen.

Copyright © 2025 Silicon Laboratories. All rights reserved.

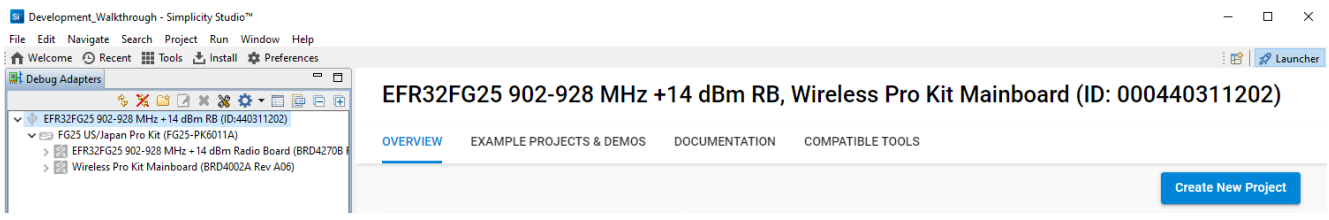


Connecting a Wi-SUN Development Kit

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.



Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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:

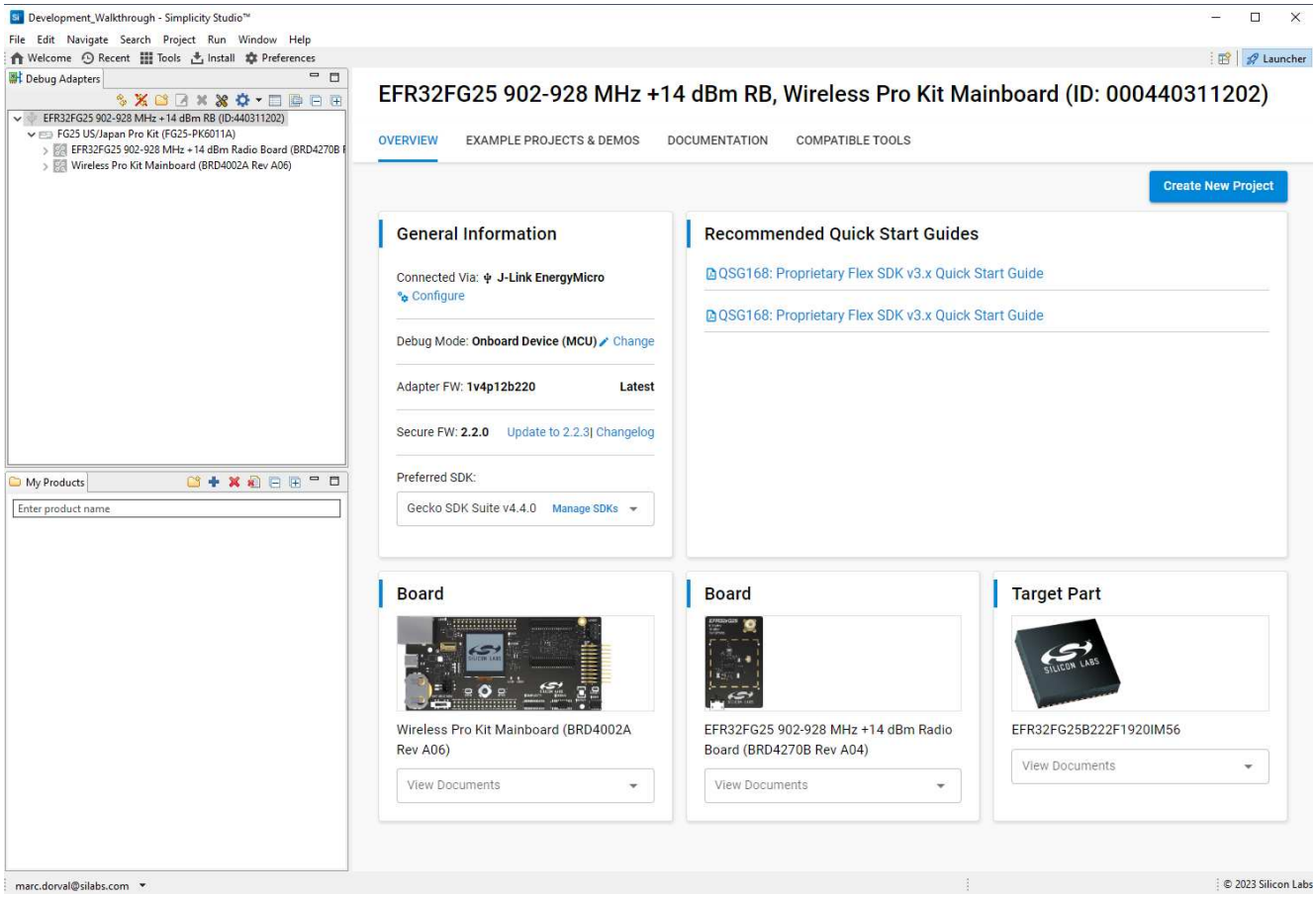
- Use console `printf` calls to help follow the application progress. It makes application development easier to get console traces. Any C developer will know how to make these optional in a 'release' build, if required, although minimal traces are always helpful for maintenance reasons, if only to display 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.

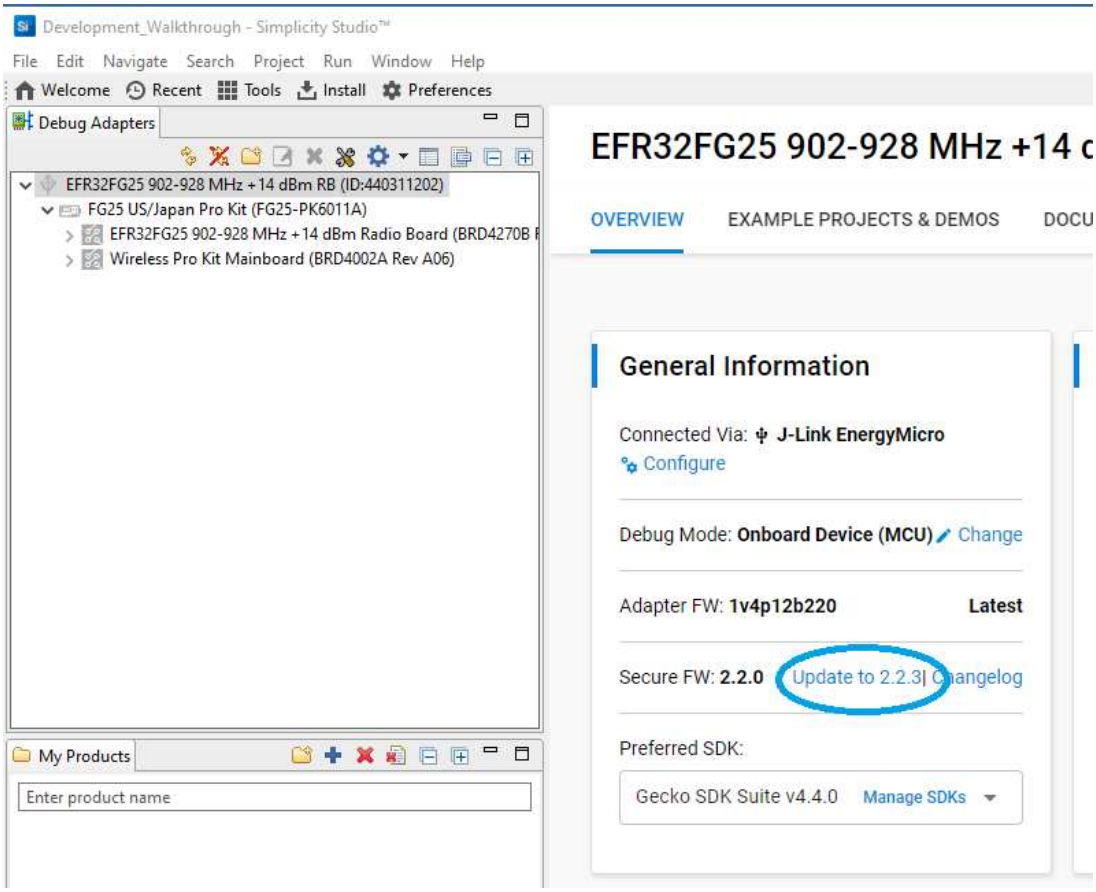
Simplicity Studio only displays information relevant to the context it is in, so here, because the target Debug Adapter is selected, it shows the Launcher perspective and the **OVERVIEW** tab for the evaluation kit. You also have easy access to the corresponding documentation and tools.

Copyright © 2025 Silicon Laboratories. All rights reserved.



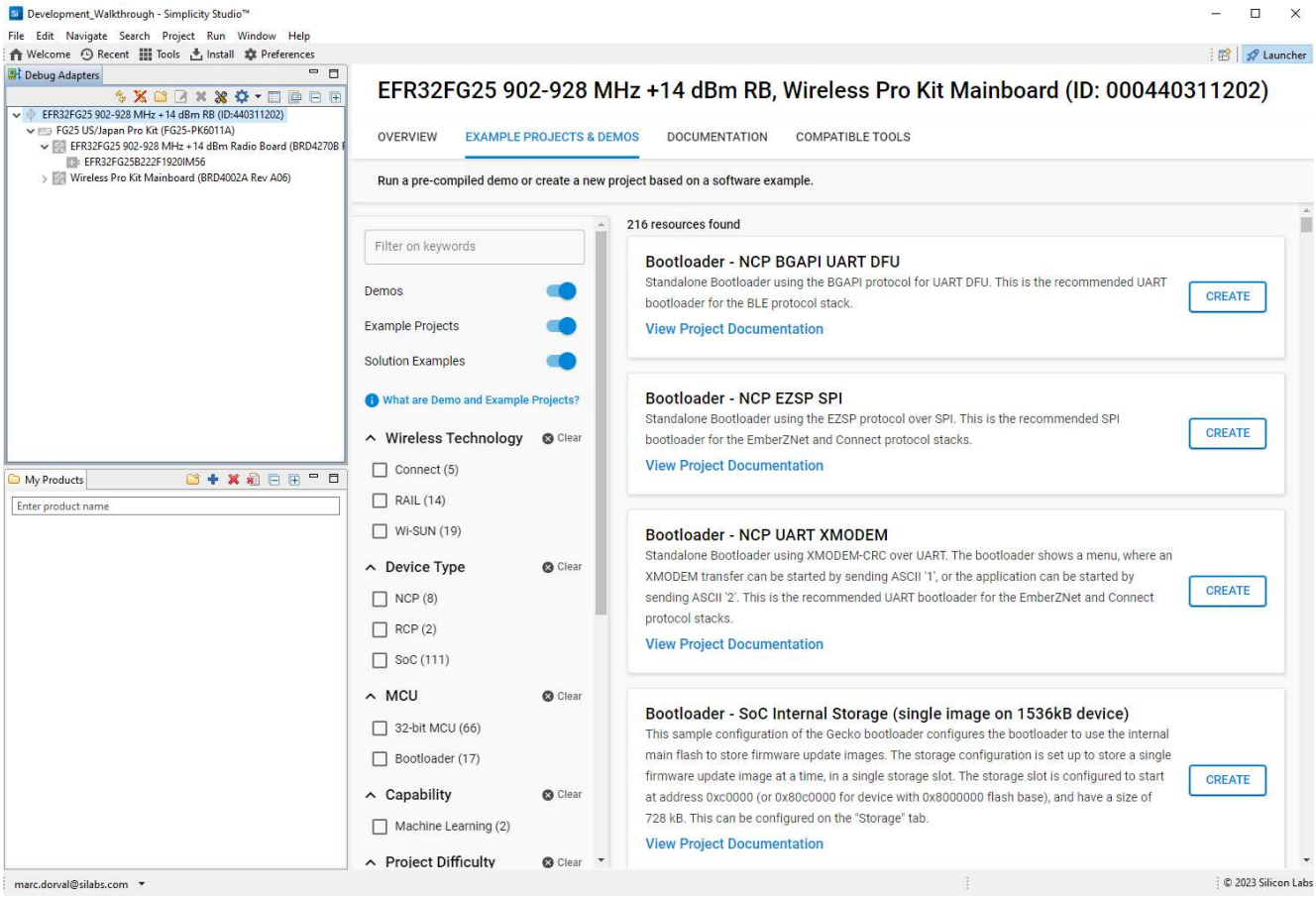
> TIP: When the **General Information** box indicates the need to update the Adapter FW or the Secure FW to newer versions, it is recommended to do so. New releases generally improve stability.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

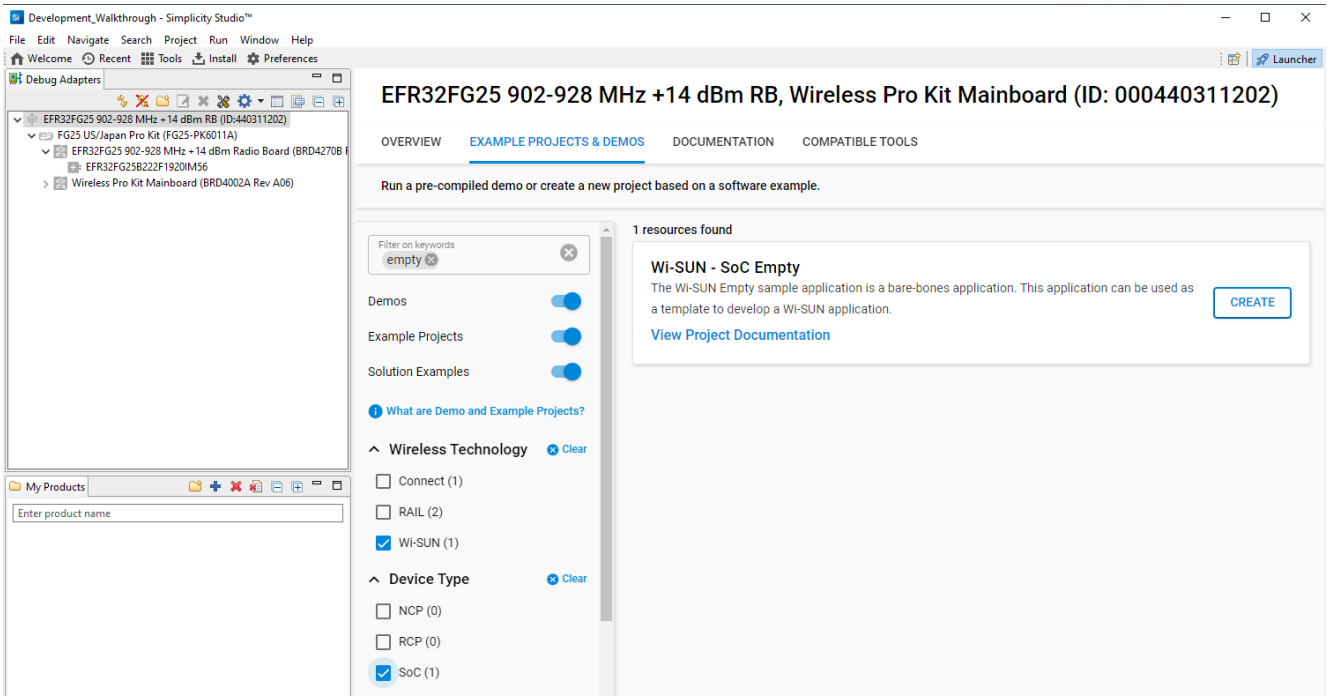


Filter the projects by clicking the following items:

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

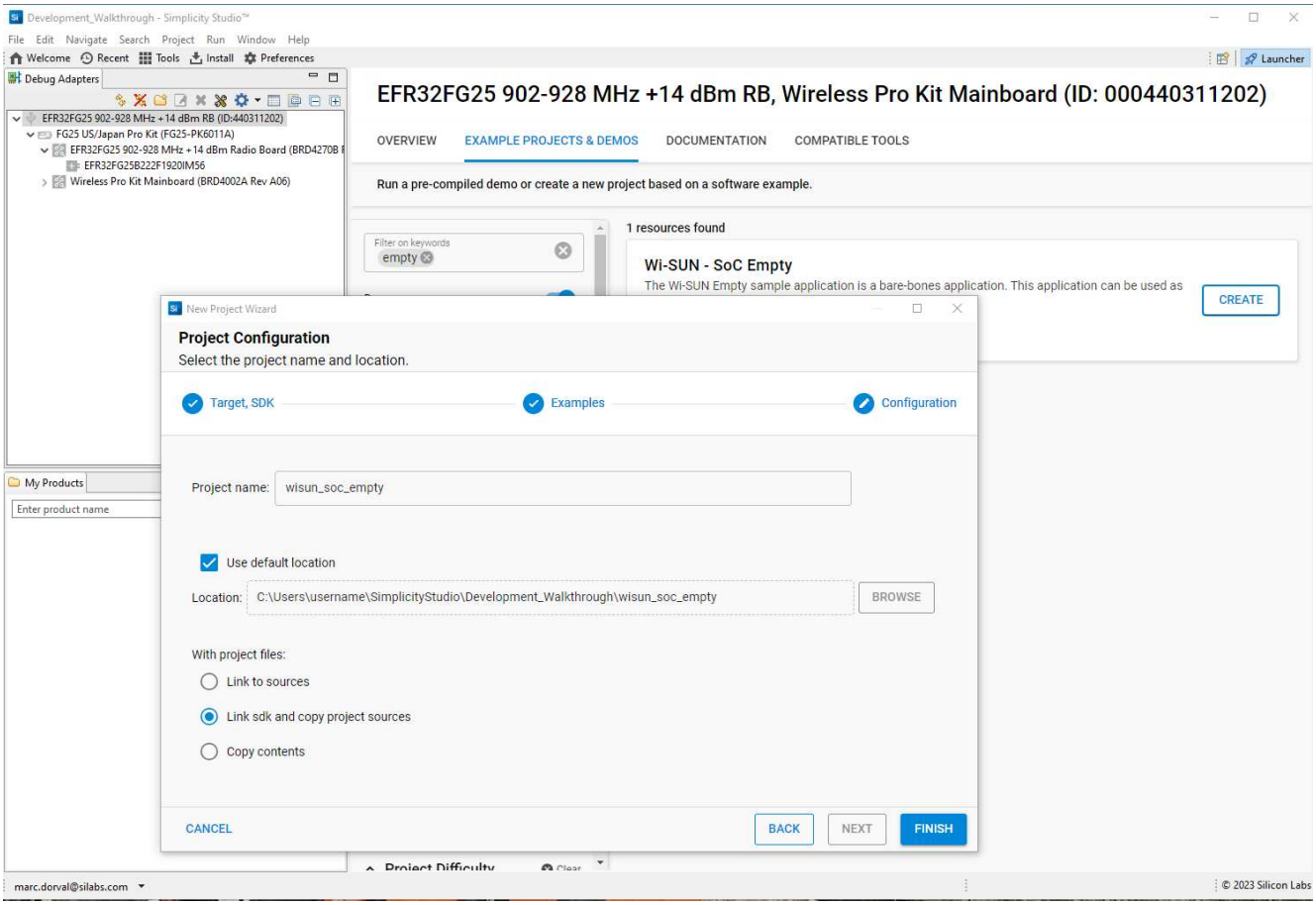
To further limit to items containing 'empty', type 'empty' in the **Filter on keywords** box, and press **Enter**.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.



Copyright © 2025 Silicon Laboratories. All rights reserved.

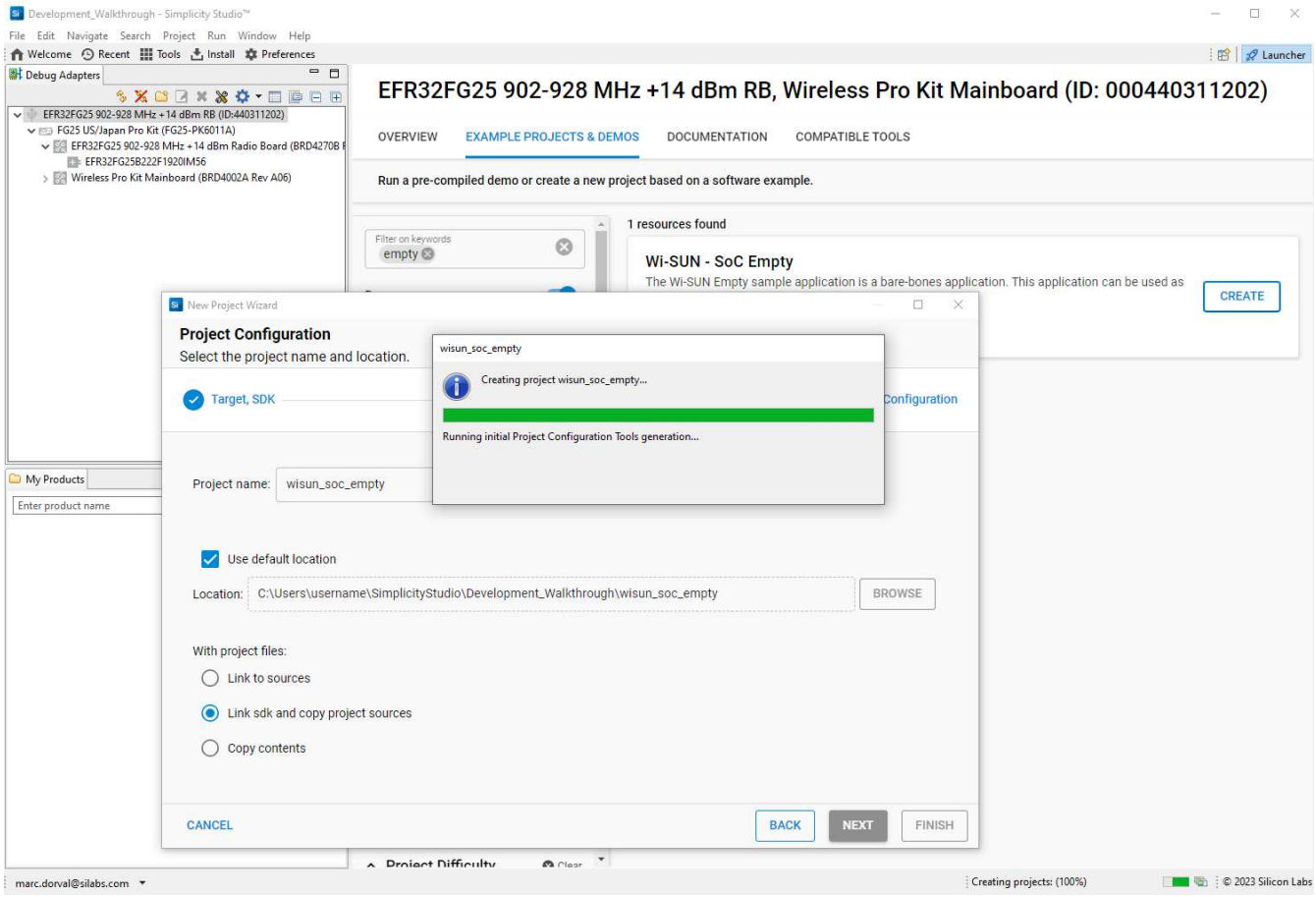
Since you are happy with the default project name and location, click **FINISH** to create the project.

> 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.

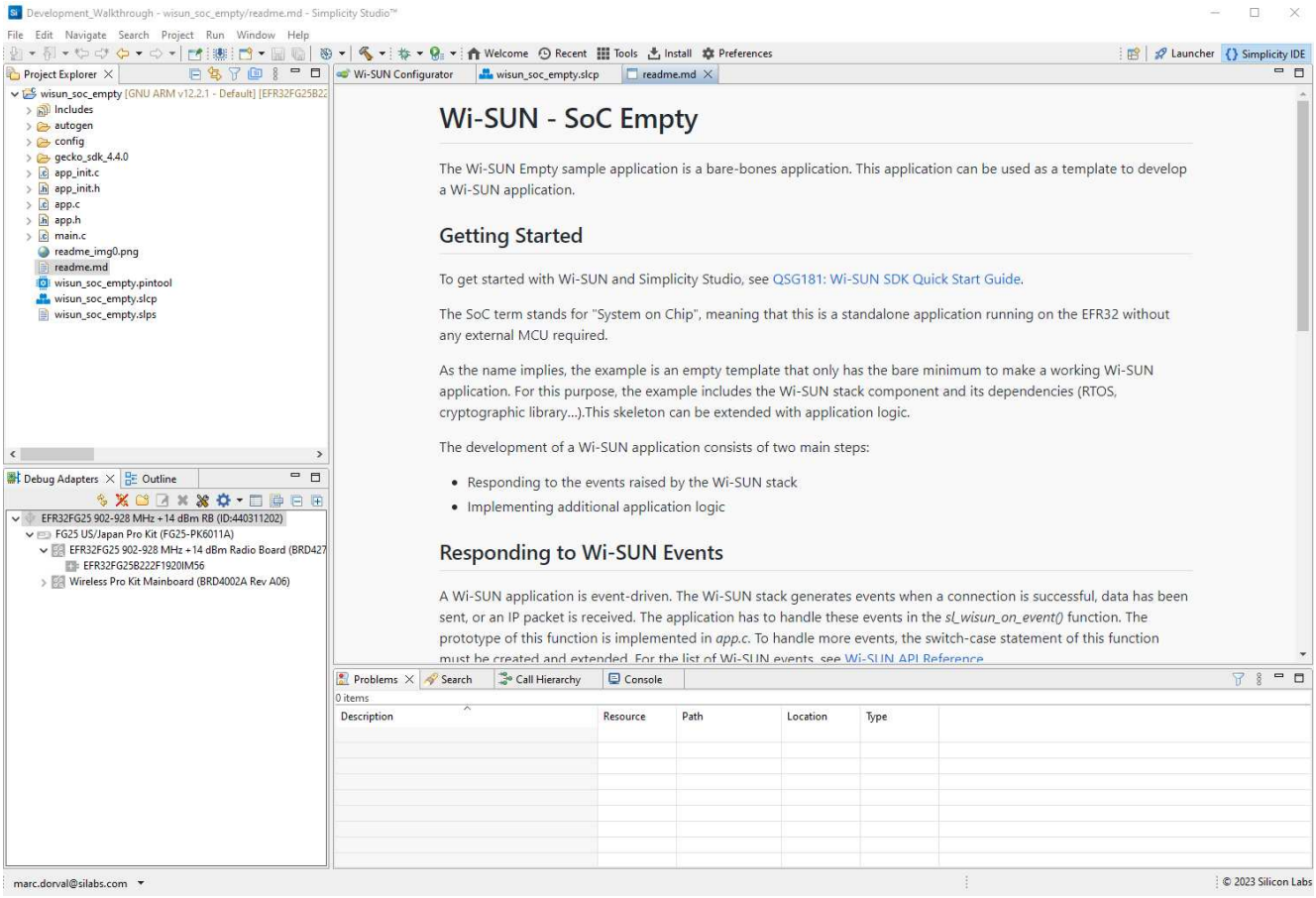
Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

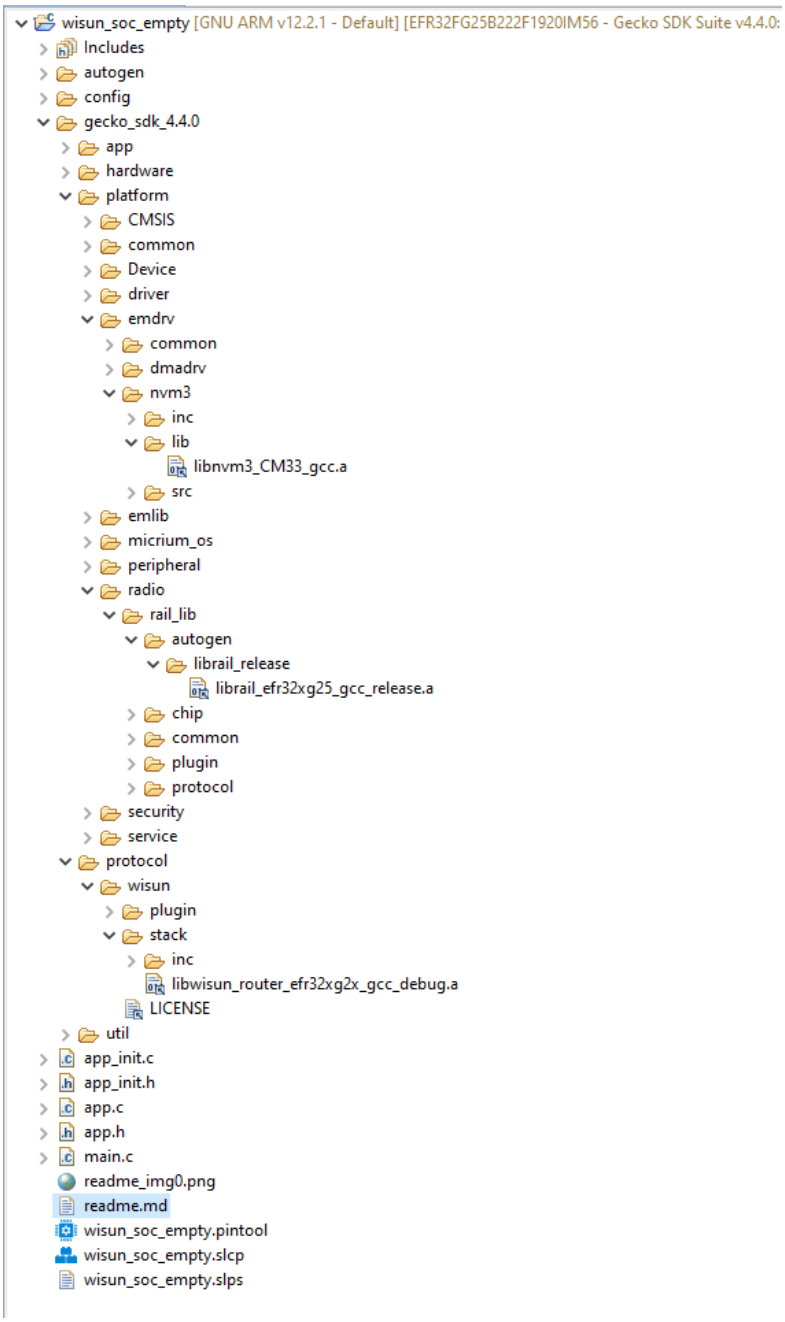


Three tabs are present in this perspective:

- **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.

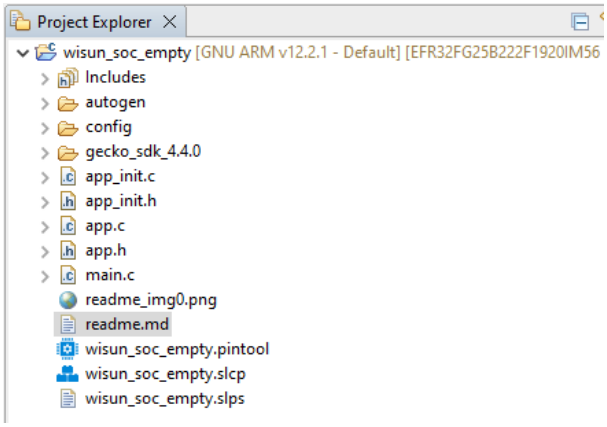
Copyright © 2025 Silicon Laboratories. All rights reserved.



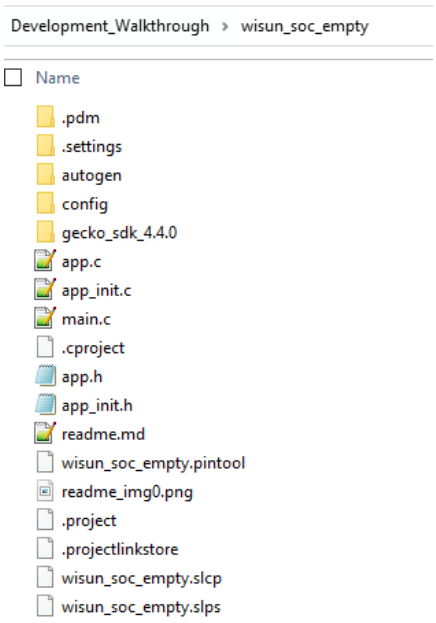
Project Explorer

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

Copyright © 2025 Silicon Laboratories. All rights reserved.



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,

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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:

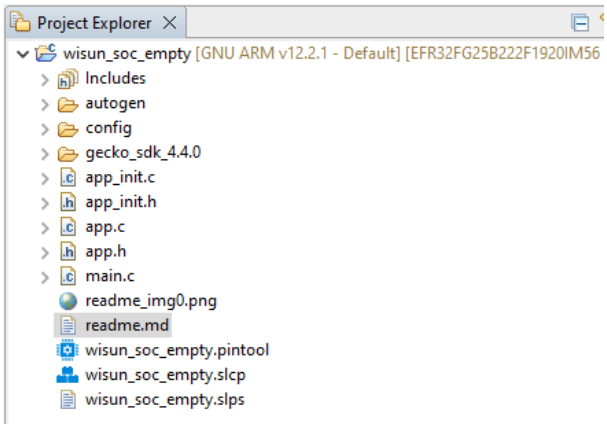
1. Open a Git Bash command window in the `Development_Walkthrough/wisun_soc_empty` folder.
2. Make it a Git-controlled folder using `git init`.
3. Copy the `gitignore` file to this folder and rename it to `.gitignore`. This is to avoid tracking object files, git files. Renaming the file is necessary 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

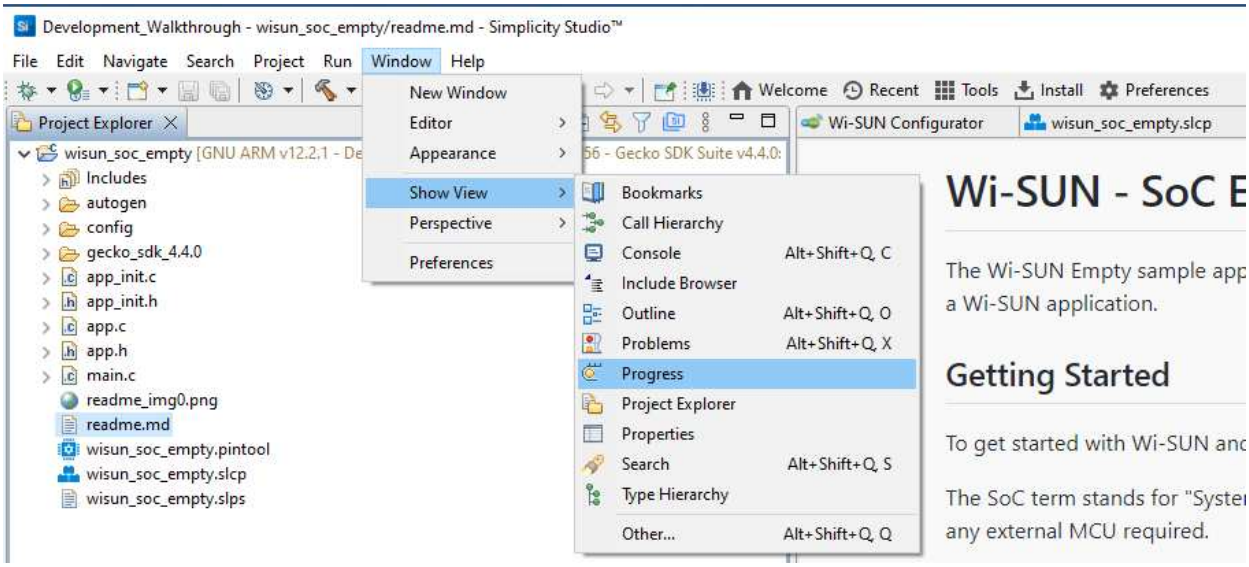
Back to Simplicity Studio's **Simplicity IDE** perspective, with your project selected, build your project with all settings by default.

> 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.



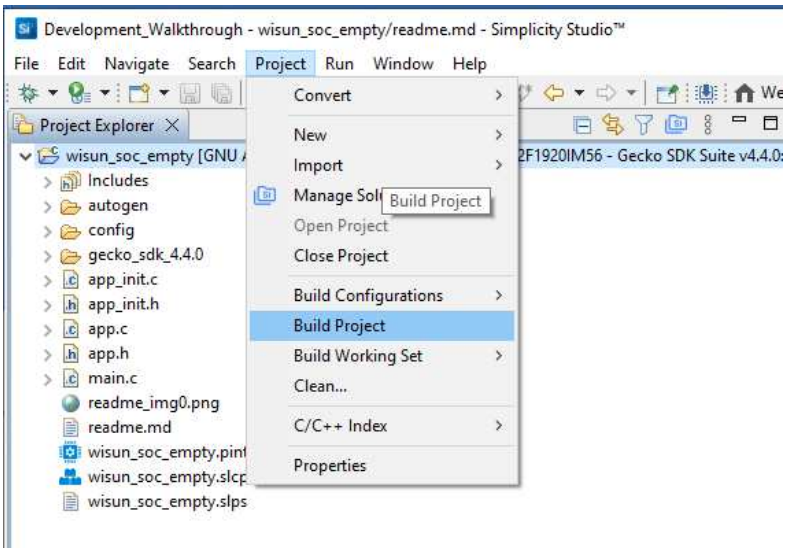
> TIP: Open the **Progress** view to check what Simplicity Studio is working on, and check when compilation is complete. Undock this window and move it next to Simplicity Studio to see progress at any time, while reducing the window dimensions. For multi-screen users, move it to a second screen.

Copyright © 2025 Silicon Laboratories. All rights reserved.

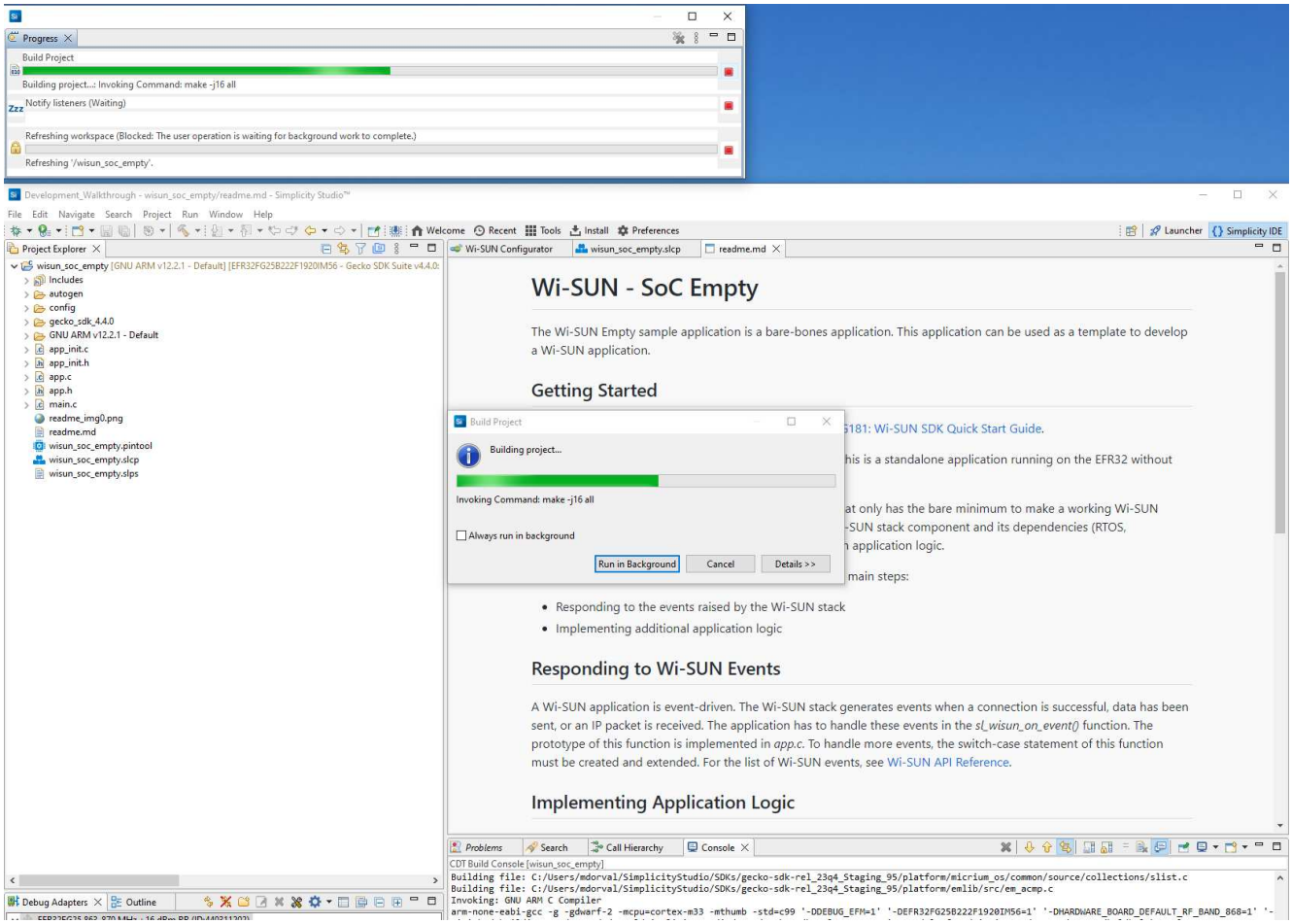


Time to Build

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



Copyright © 2025 Silicon Laboratories. All rights reserved.

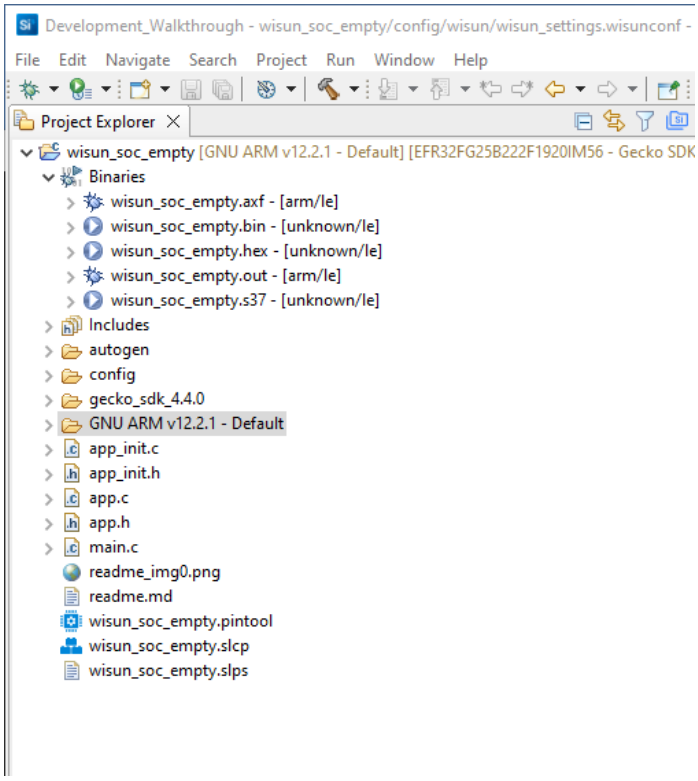


Hopefully, the project compiles without errors. Since this was a first-time full build, it takes approximately 1 min. Subsequent builds are generally smaller, if only adding minor changes to a couple files (only files impacted by the changes are recompiled).

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

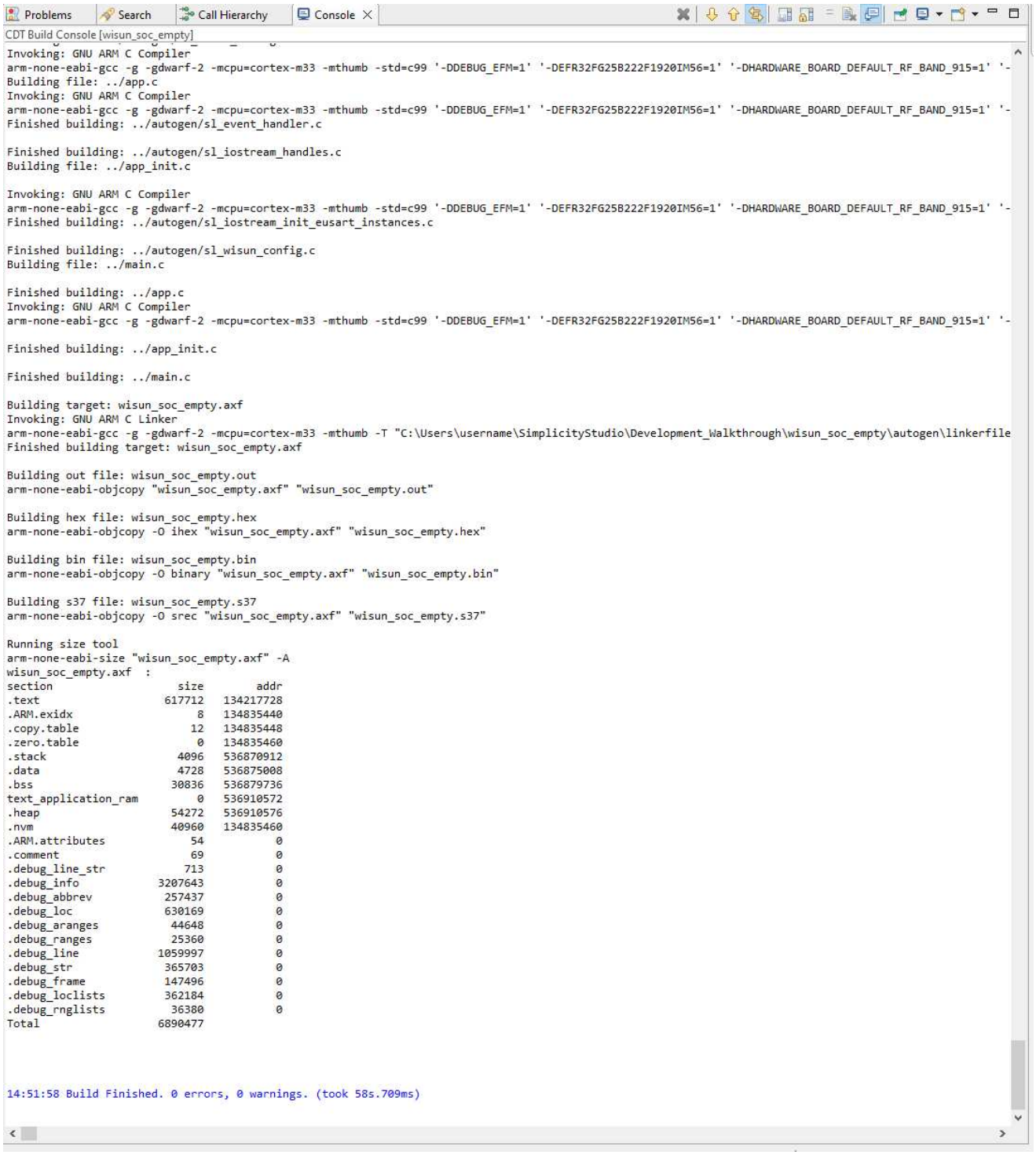
What Happened

Copyright © 2025 Silicon Laboratories. All rights reserved.



- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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 ...
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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).

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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.

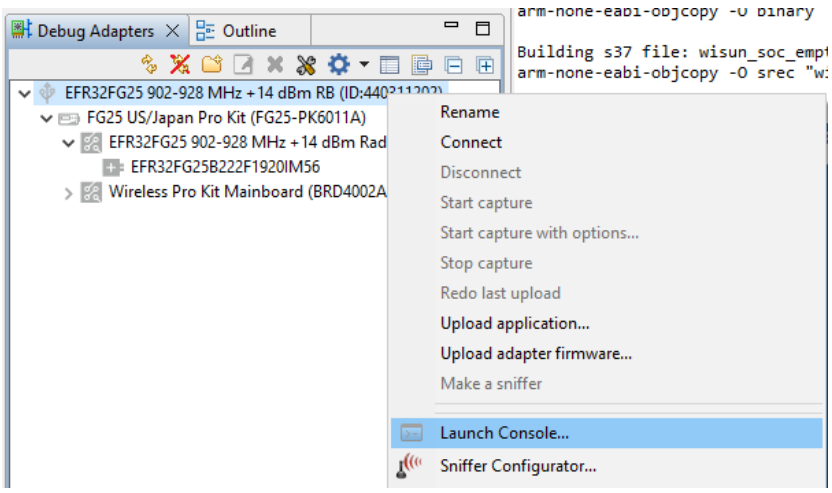
Flashing the Binary to the Adapter Board

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

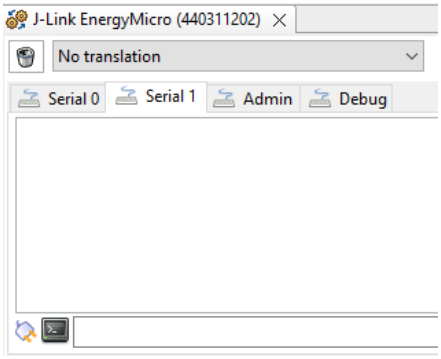


A new tab will open, with the name of the J-Link adapter and it's ID.

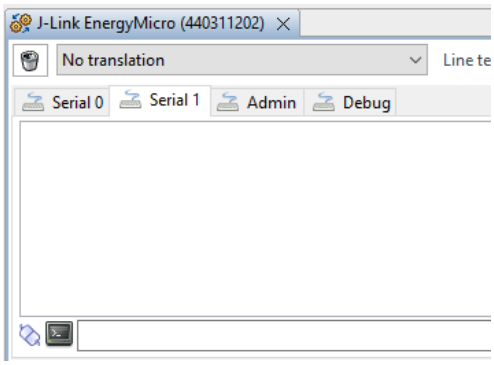
3. In this new tab, select the **Serial 1** tab to access the application's UART console. This will show all strings printed from the application using `printf()`.

> 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.



Copyright © 2025 Silicon Laboratories. All rights reserved.

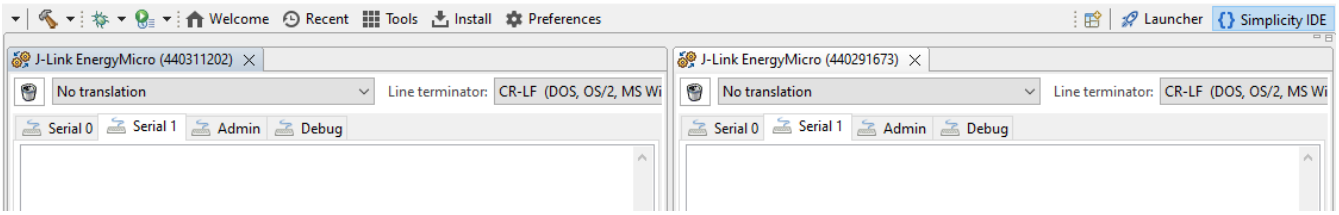
The Console is now connected. You can flash your application, but you need to start a Border Router first to be able to connect over your Wi-SUN network.

> 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

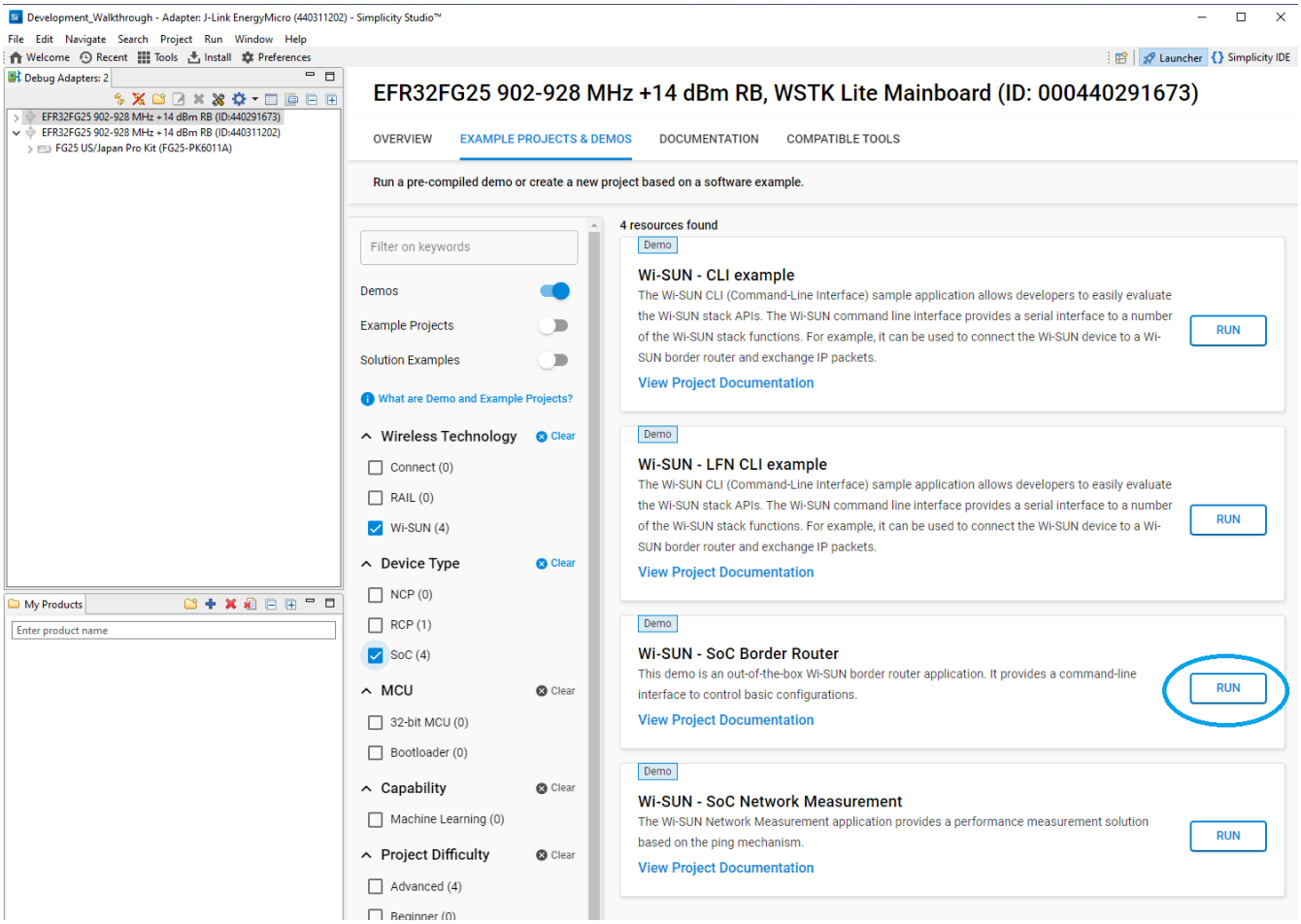
To make your development walkthrough more pleasant, it's time now to start a Border Router with default settings, such that your Empty application will be able to connect to an existing Wi-SUN network.

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.

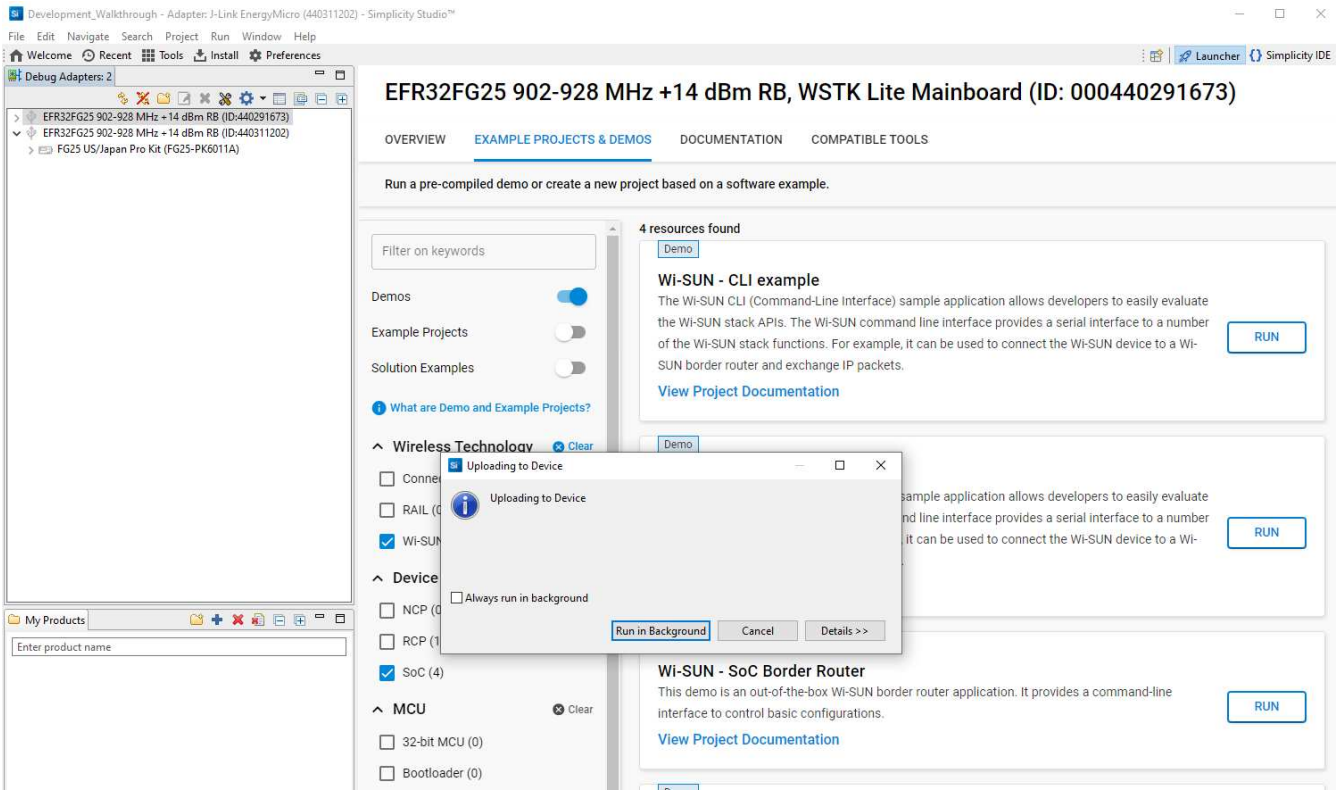


5. Click **Enter** in the new console to connect the UART.
6. Click **Launcher** at the top right to open the Launcher perspective.
7. Select the **EXAMPLE PROJECTS AND DEMOS**.
8. Leave only **Demos** selected.
9. Filter on **Wi-SUN**.
10. Filter on **SoC**.
    Only three applications match.
11. Move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

Copyright © 2025 Silicon Laboratories. All rights reserved.

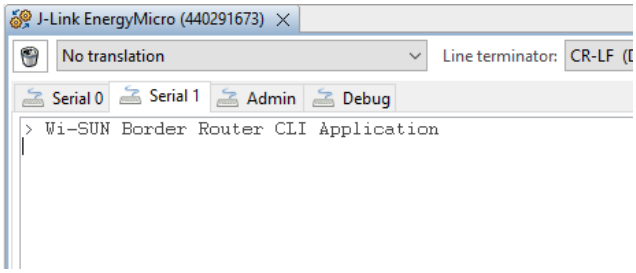


This immediately triggers flashing the corresponding pre-compiled binary to the target.

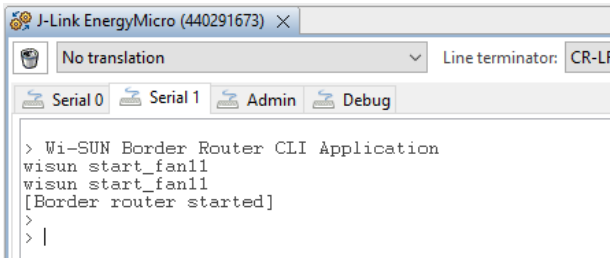


Copyright © 2025 Silicon Laboratories. All rights reserved.

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



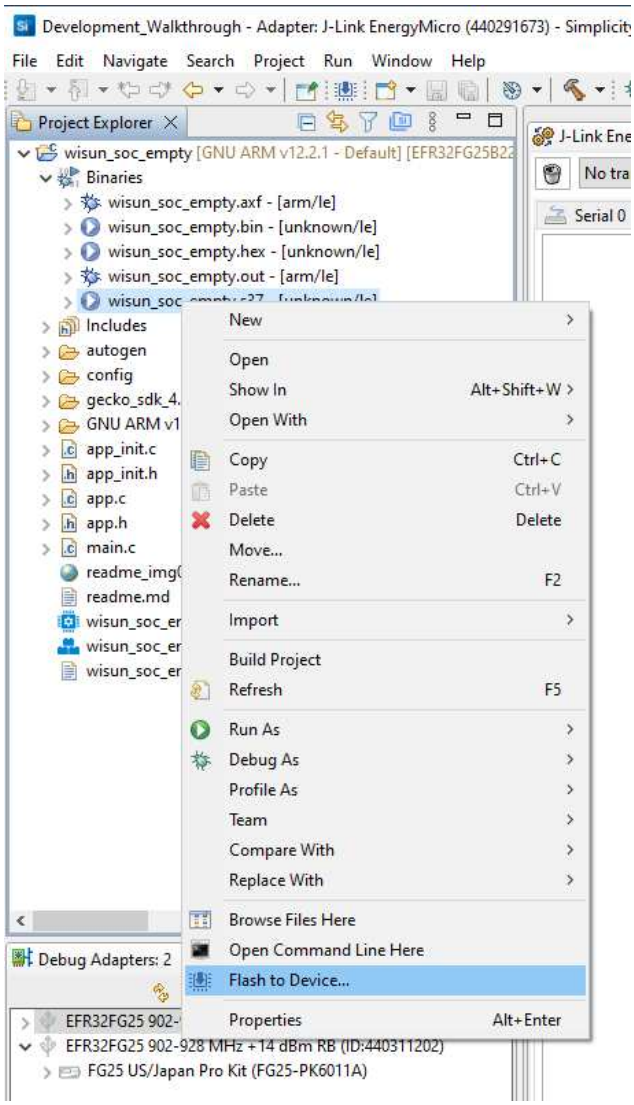
14. In this console, use `wisun start_fan11` to start the Border Router with default FAN1.1 settings.



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**.

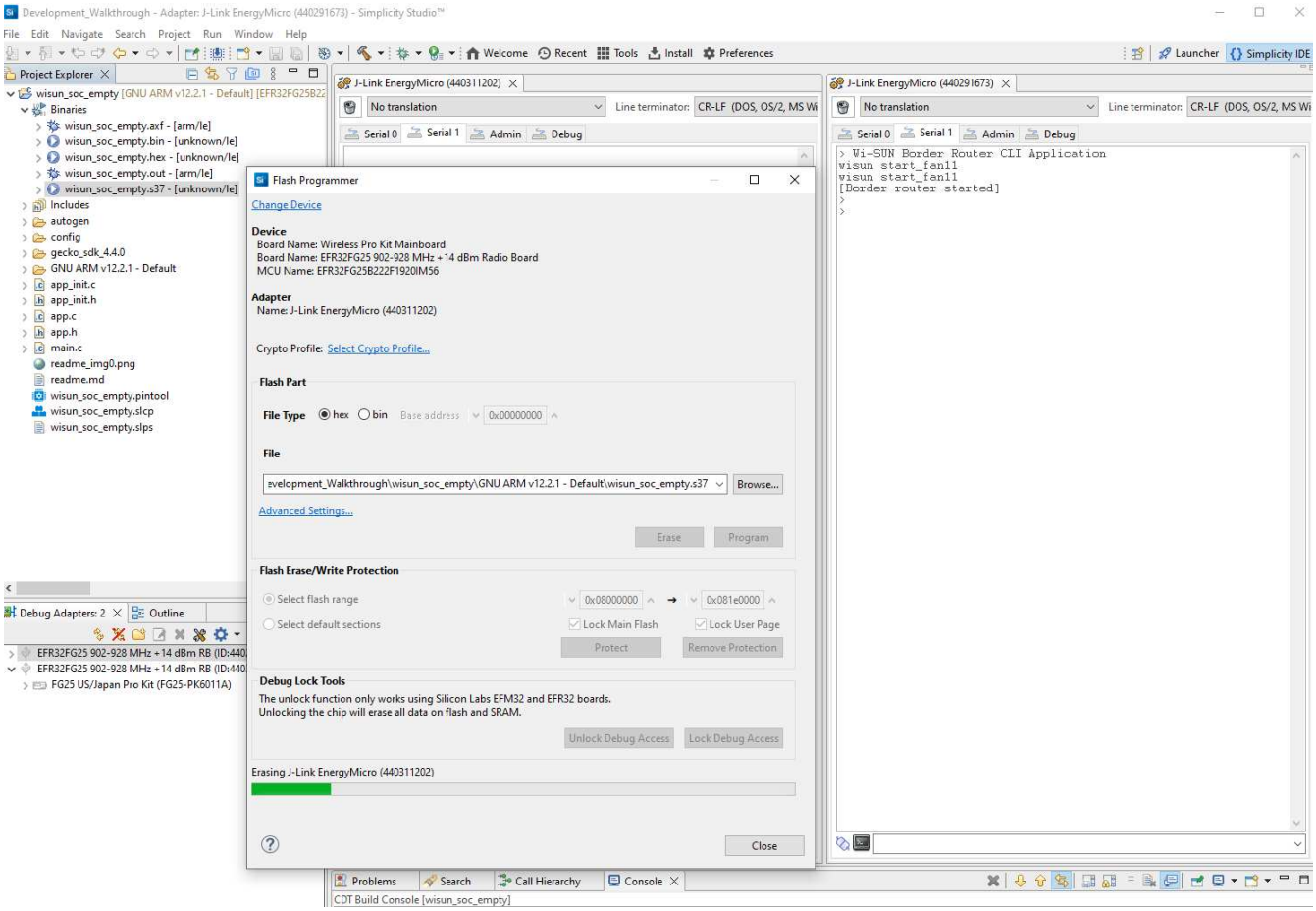
Copyright © 2025 Silicon Laboratories. All rights reserved.



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

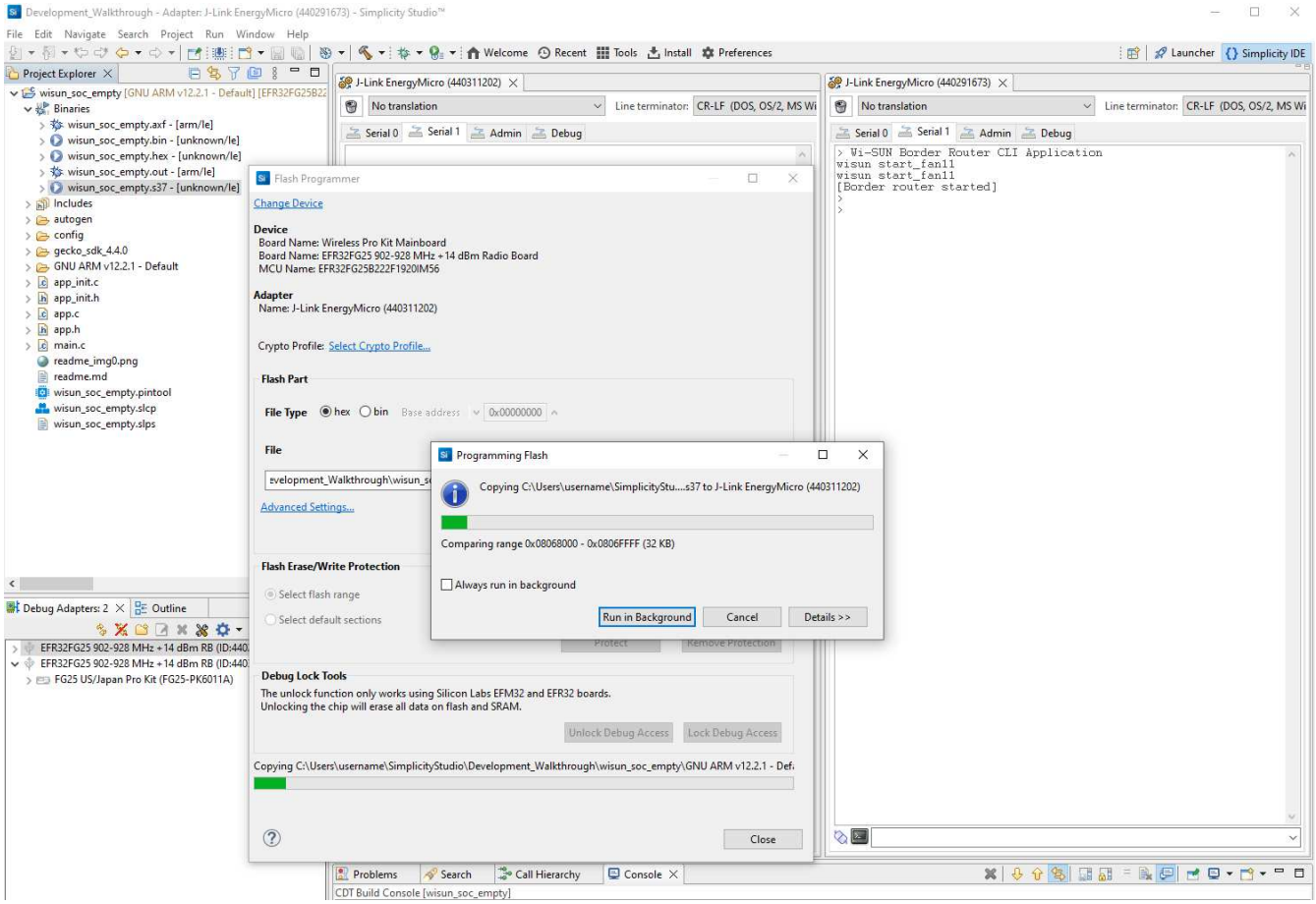
> TIP: For a first flash of a new application on a Radio Board where other application may have been running before, use the **Erase** button to clear any data possibly stored in flash. A progress bar appears and closes when done.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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

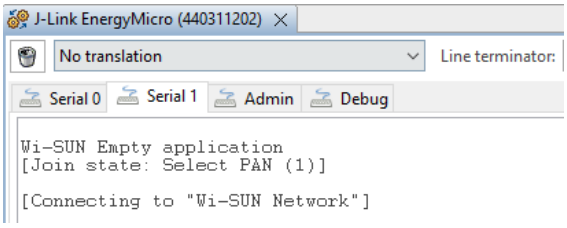
Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
app.c
75  /*App task function*/
76  void app_task(void *args)
77  {
78    (void) args;
79
80    printf("\nWi-SUN Empty application\n");
81
82  #ifdef SL_CATALOG_WISUN_APP_CORE_PRESENT
83    // connect to the wisun network
84    app_wisun_connect_and_wait();
85  #endif
86
87    while (1) {
88      ////////////////////////////////////////////////////////////////////////////
89      // Put your application code here!                                        //
90      ////////////////////////////////////////////////////////////////////////////
91      app_wisun_dispatch_thread();
92    }
93  }
```

- 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.

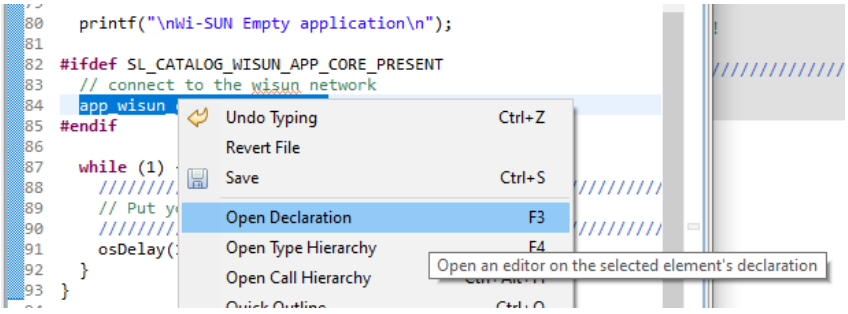
> TIP: When a component is present in the project, the corresponding code is shown with normal colors. If a component is not present, the corresponding code is grayed out in the editor, meaning that it is not compiled and can be ignored.

- 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

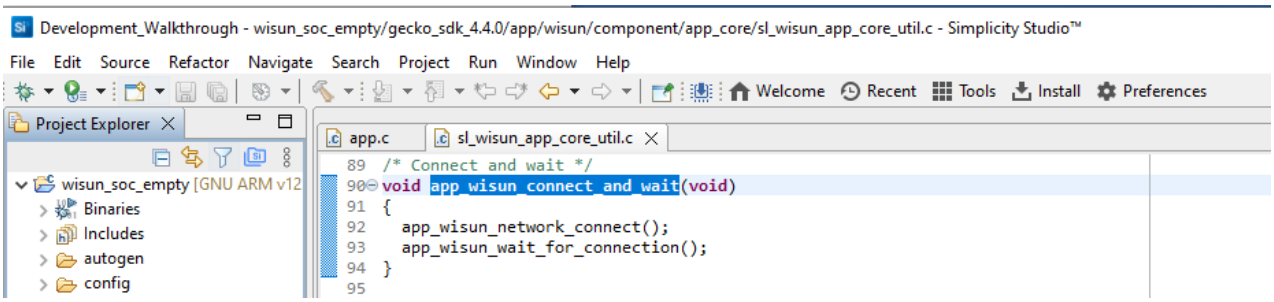
For users not familiar with Simplicity Studio and C code in general, the following are some elementary code checks:

- 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.

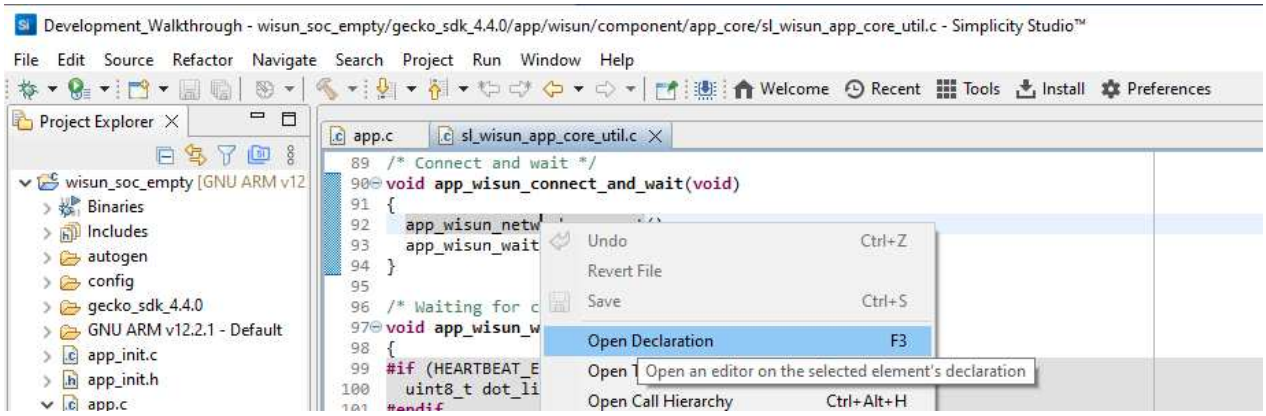


Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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.

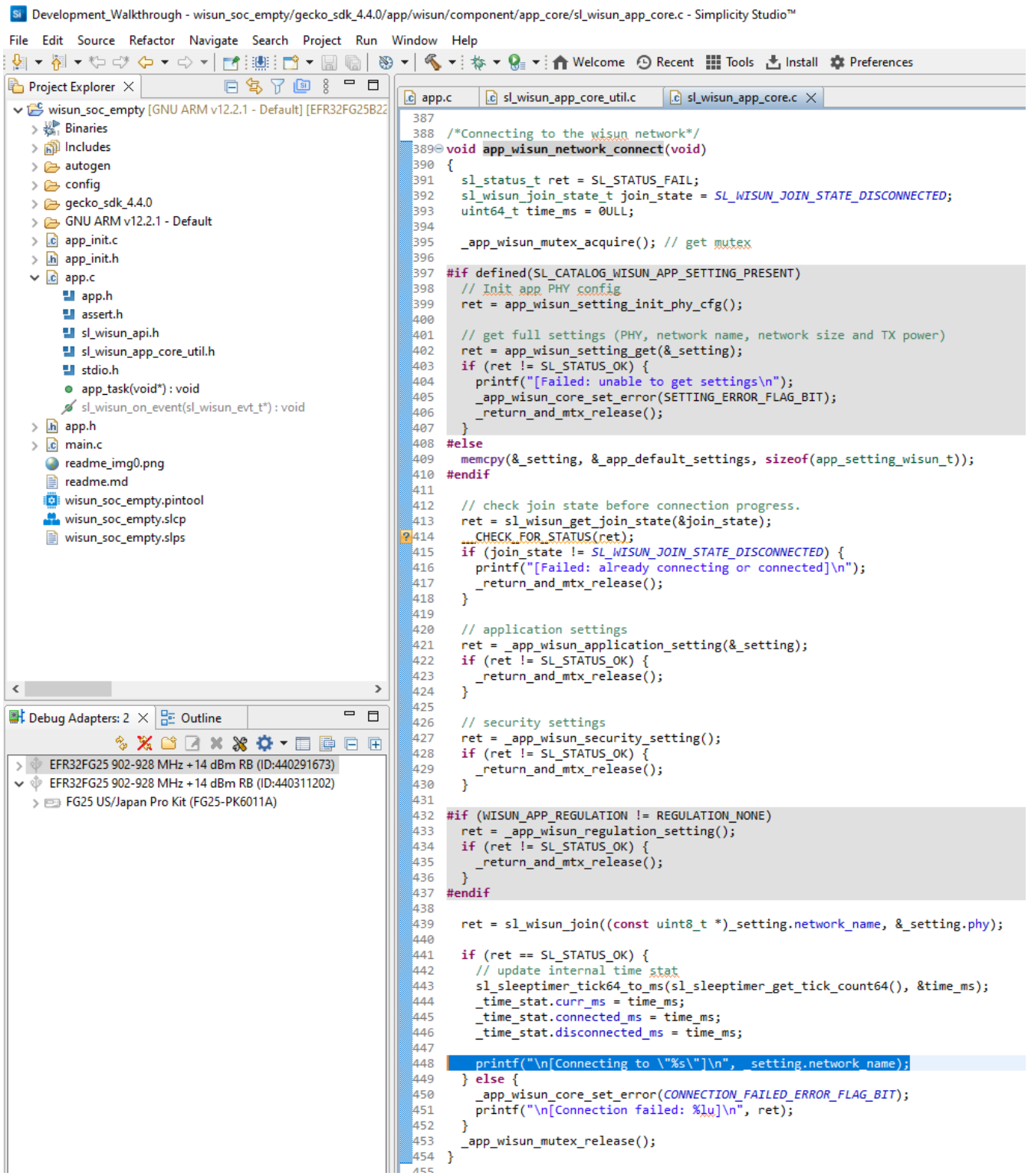


- You find two function calls here. Given the function names, you get a pretty good idea of what they do. Since the **Connecting to...** message probably comes from `app_wisun_network_connect()`, look for it's declaration as well.



- 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).

Copyright © 2025 Silicon Laboratories. All rights reserved.



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

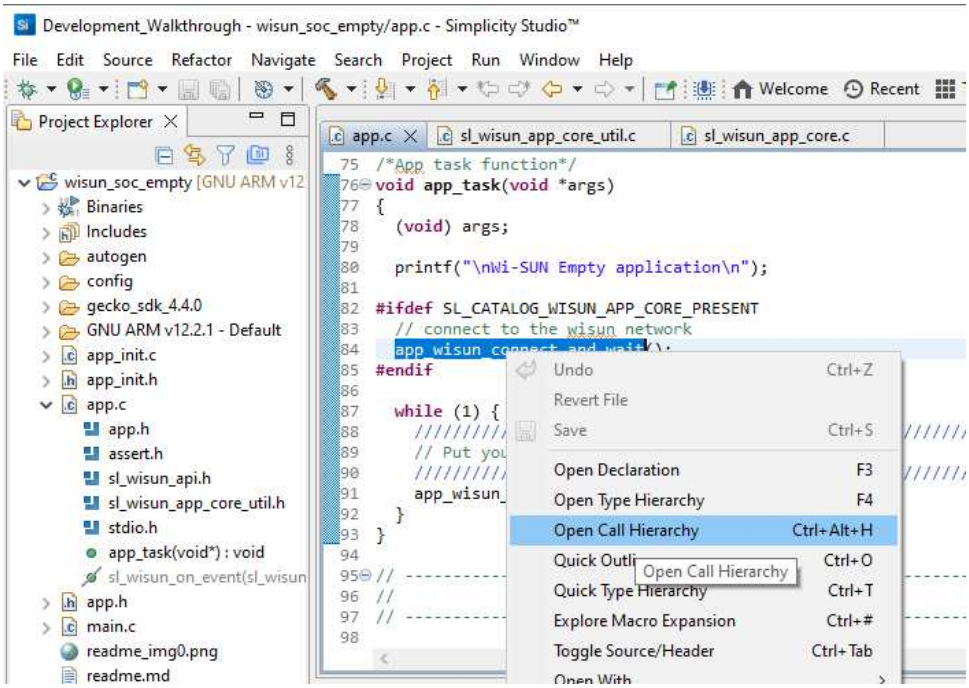
While you are looking at `app_wisun_network_connect()`, you can see that there are several steps during the connection preparation:

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

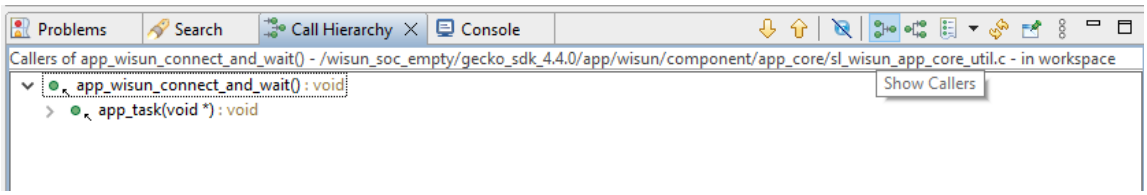
Copyright © 2025 Silicon Laboratories. All rights reserved.

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

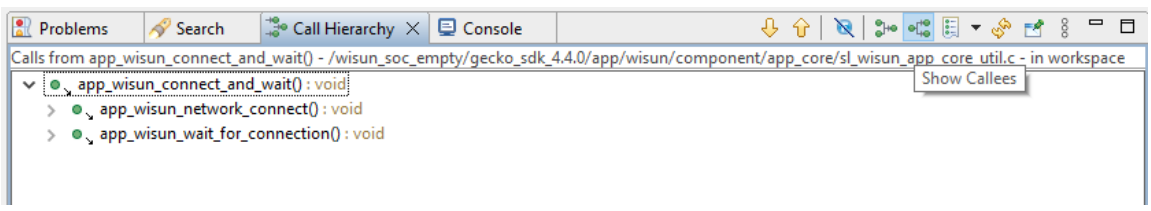
- An alternative method to follow the function calls is to use the **Open Call hierarchy** for the `app_wisun_connect_and_wait()` function.



- 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.

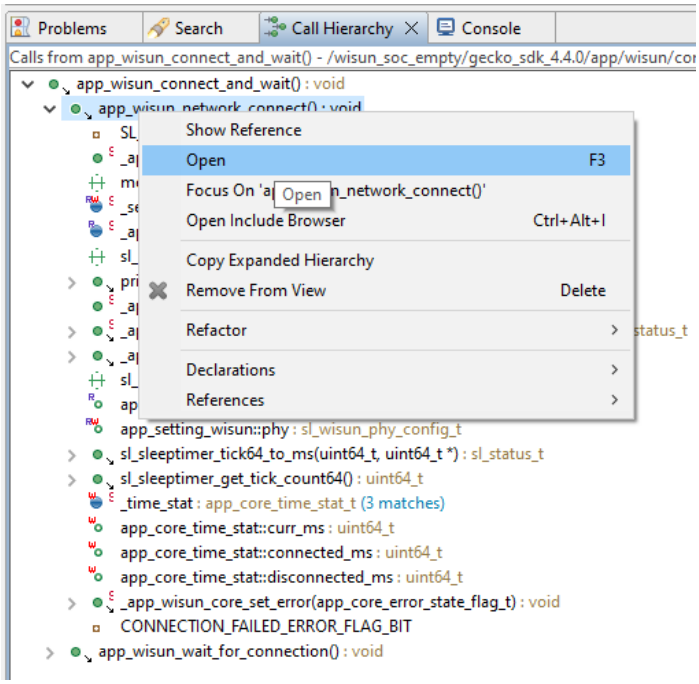


  - The **Show Callees** view shows which functions are called by `app_wisun_connect_and_wait()` :



    - 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



- 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.

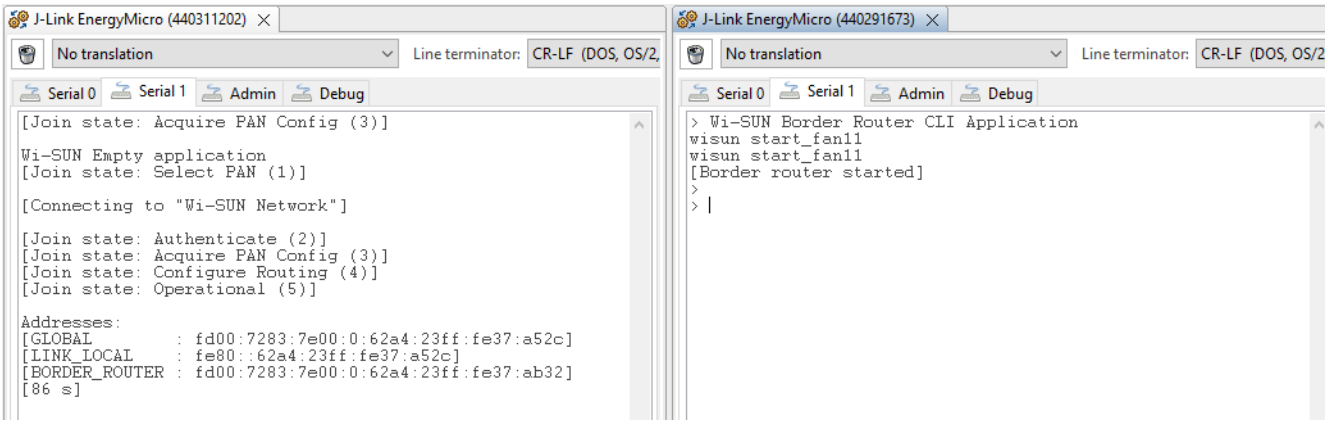
Use:

Copyright © 2025 Silicon Laboratories. All rights reserved.

- `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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

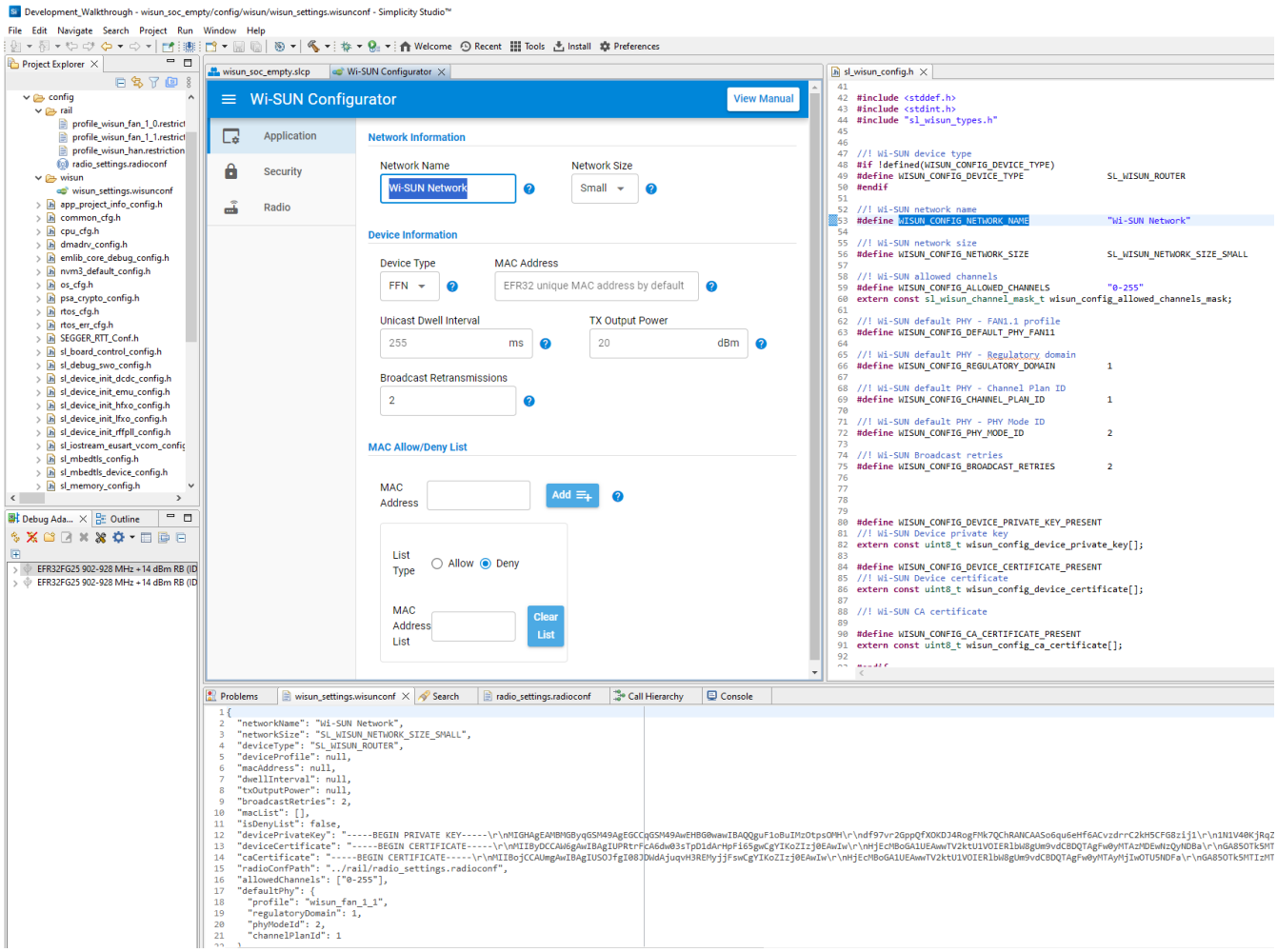
First observe what you have by default:

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`

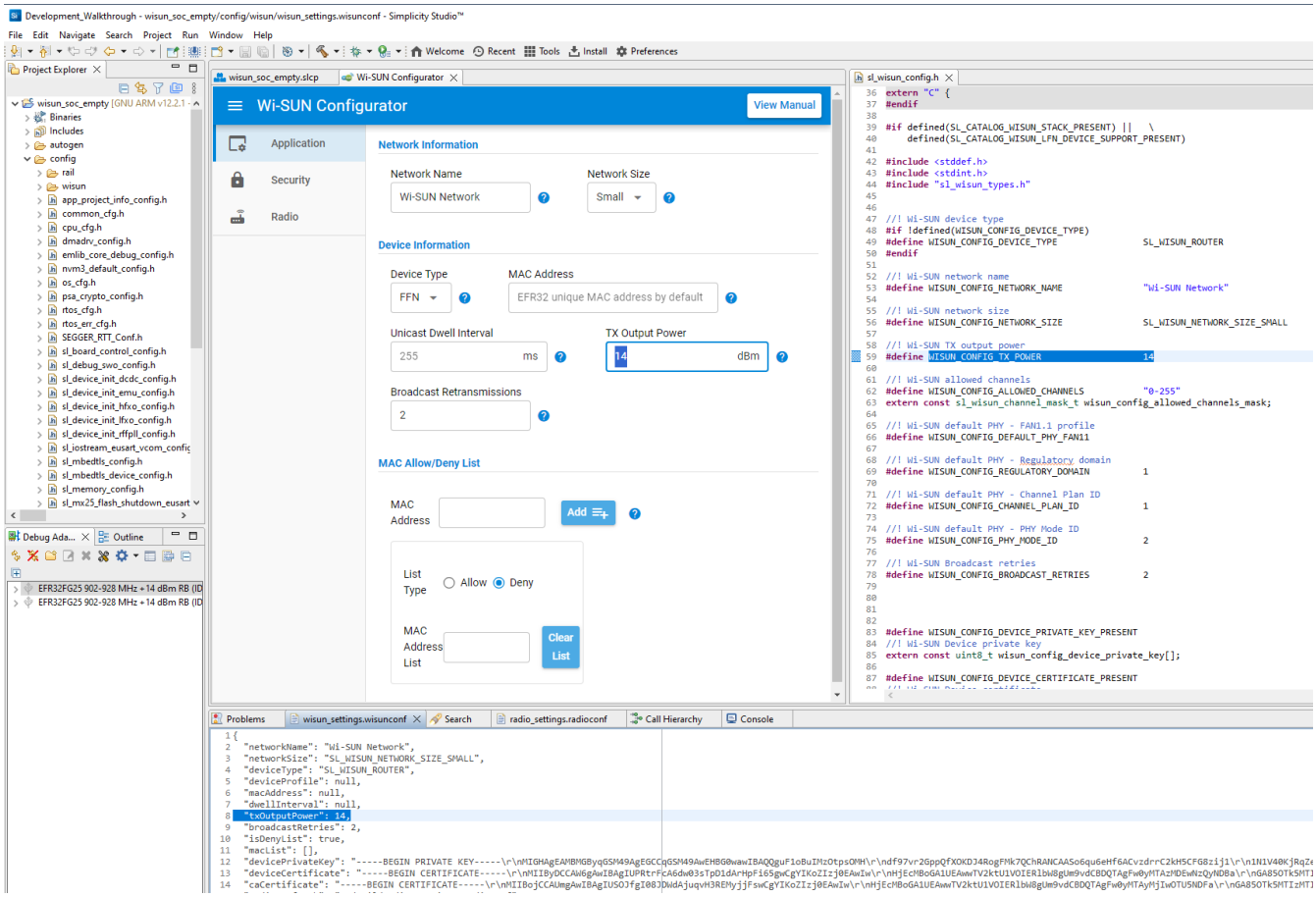
Copyright © 2025 Silicon Laboratories. All rights reserved.



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`

Copyright © 2025 Silicon Laboratories. All rights reserved.



- 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` :

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

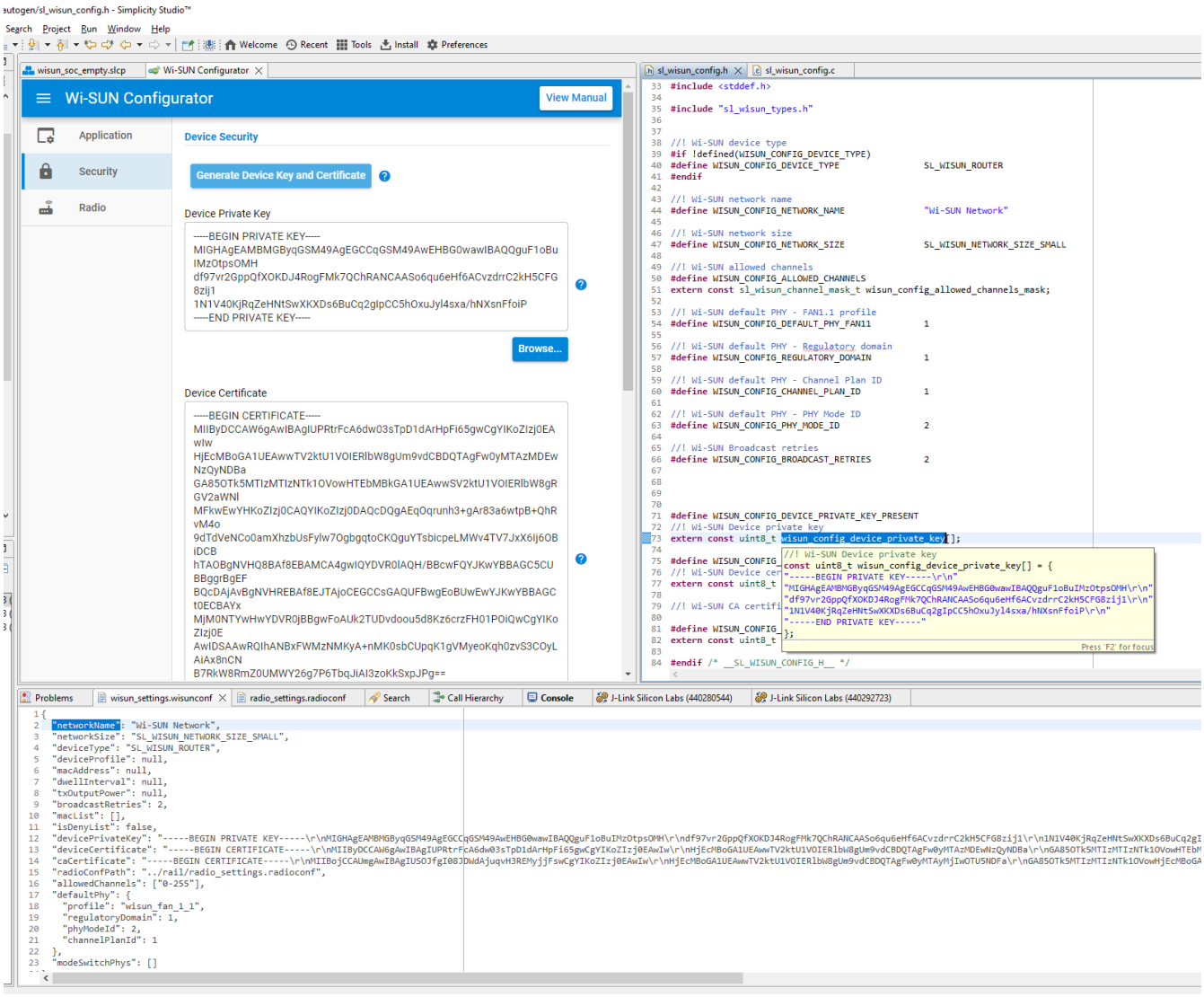
There are close relationships between:

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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.

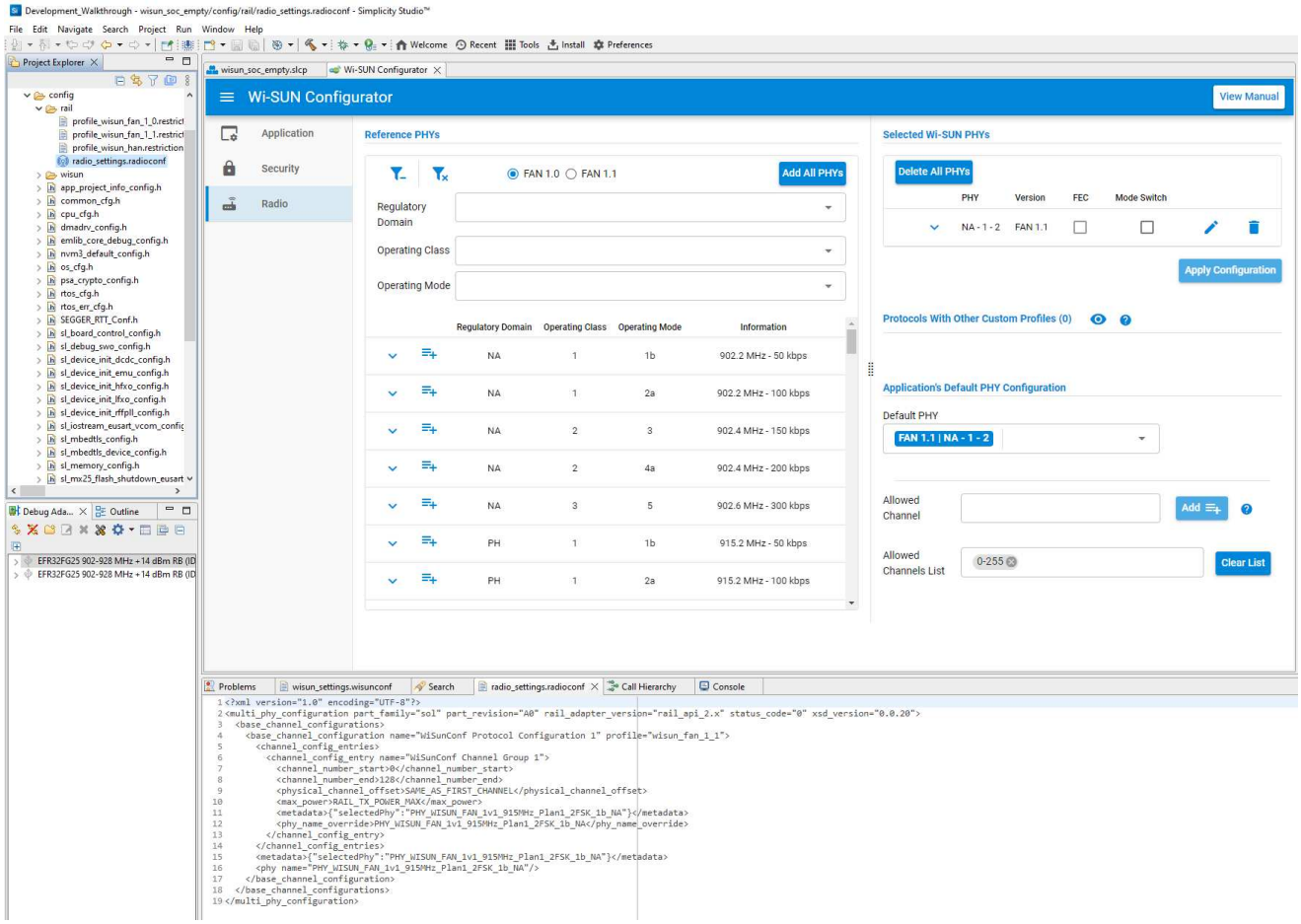
> TIP: As already stated, in a production application, these will need to be different per device. The CA certificate

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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.



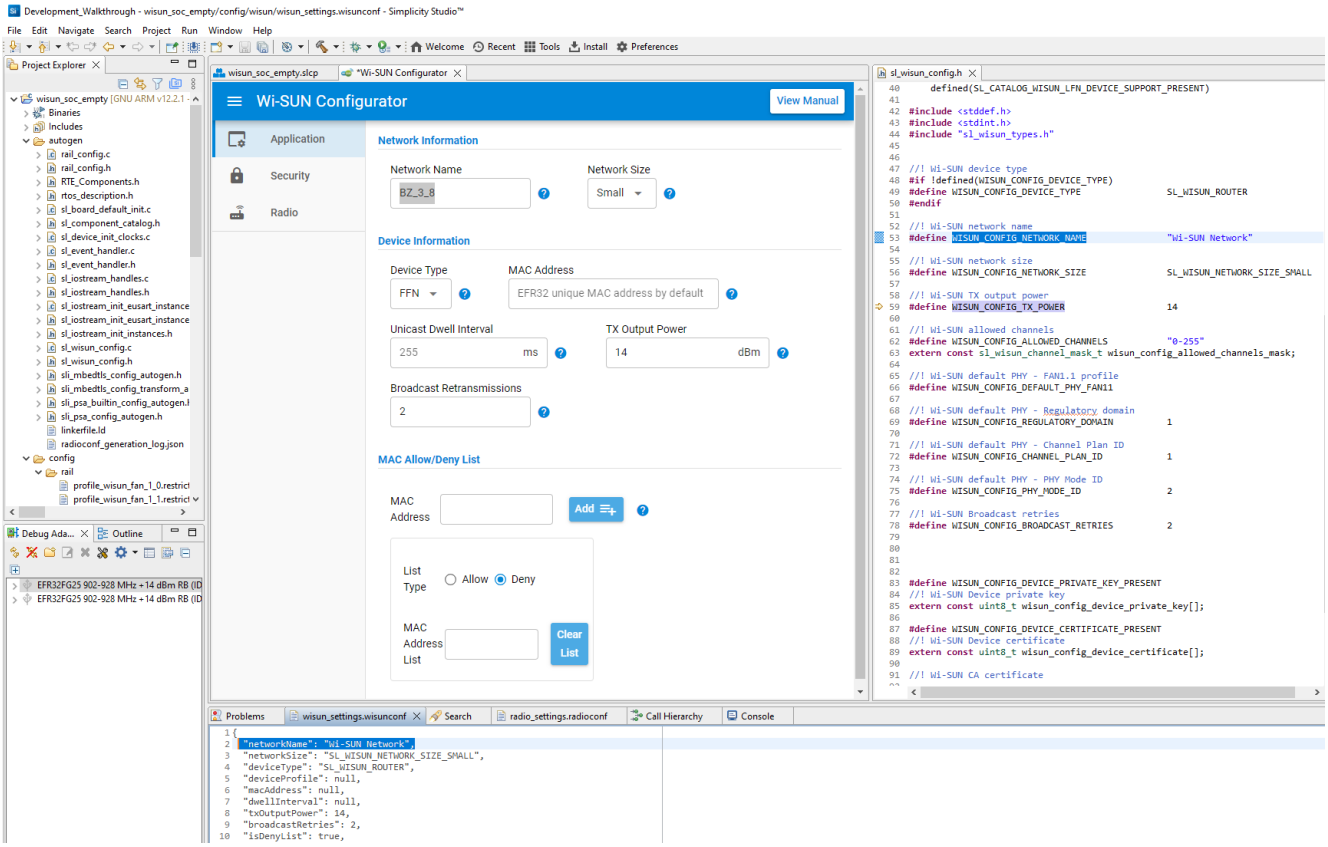
Observing `config/rail/radio_settings.radioconf`, you can check that `<phy_name_override>` matches the select PHY.

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.

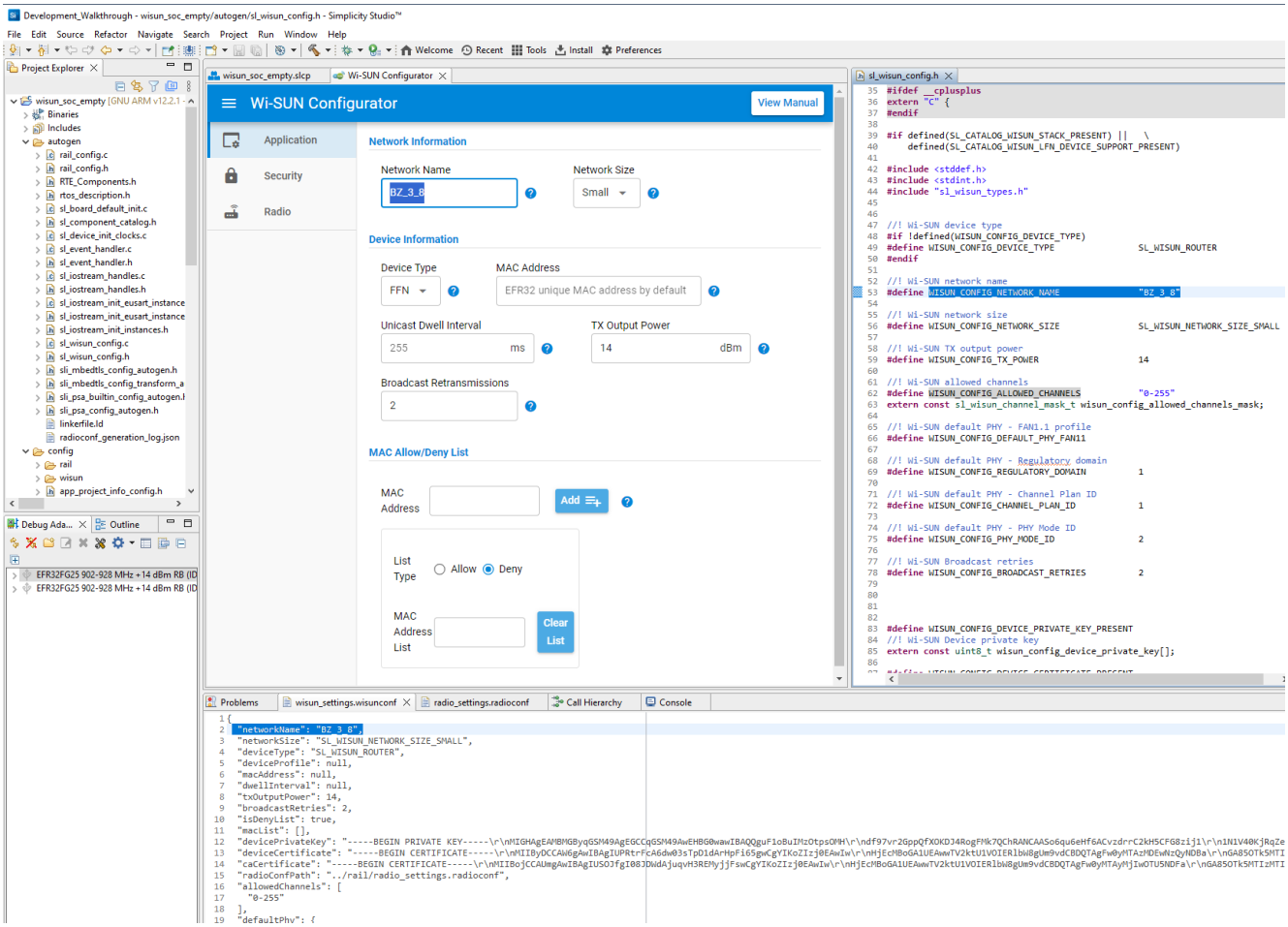
Copyright © 2025 Silicon Laboratories. All rights reserved.



- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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

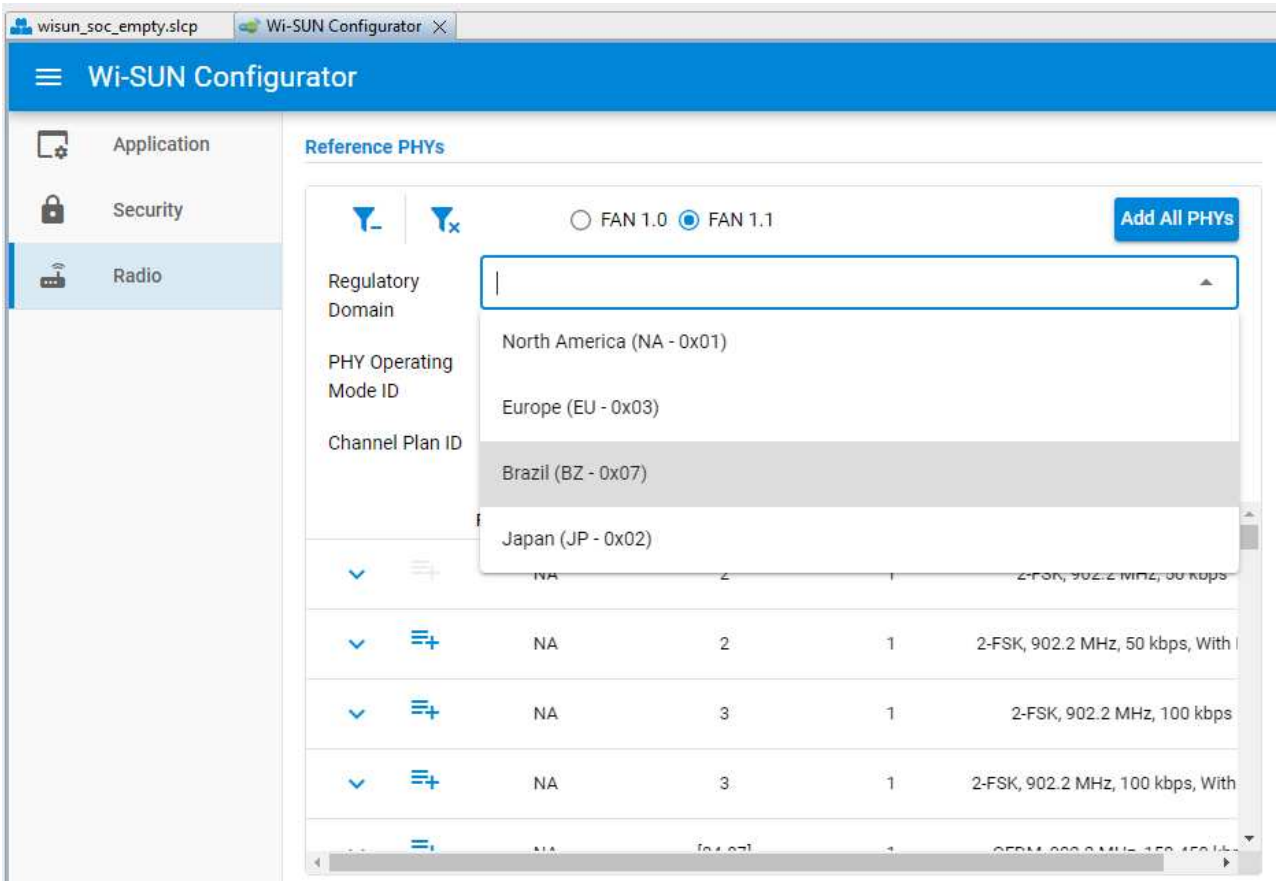
Now move to the **Radio** tab, since you won't change the security settings yet.

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.

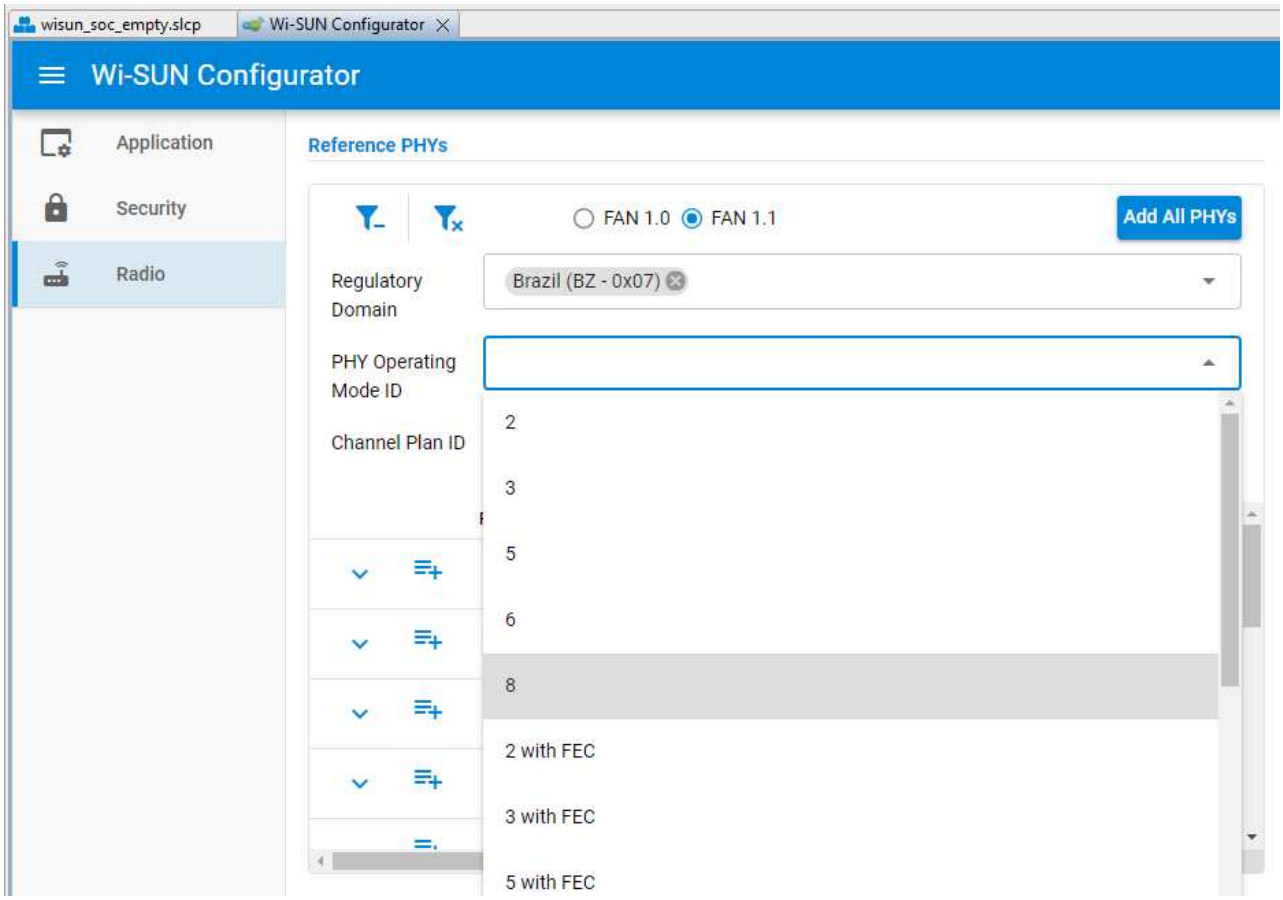
Copyright © 2025 Silicon Laboratories. All rights reserved.



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

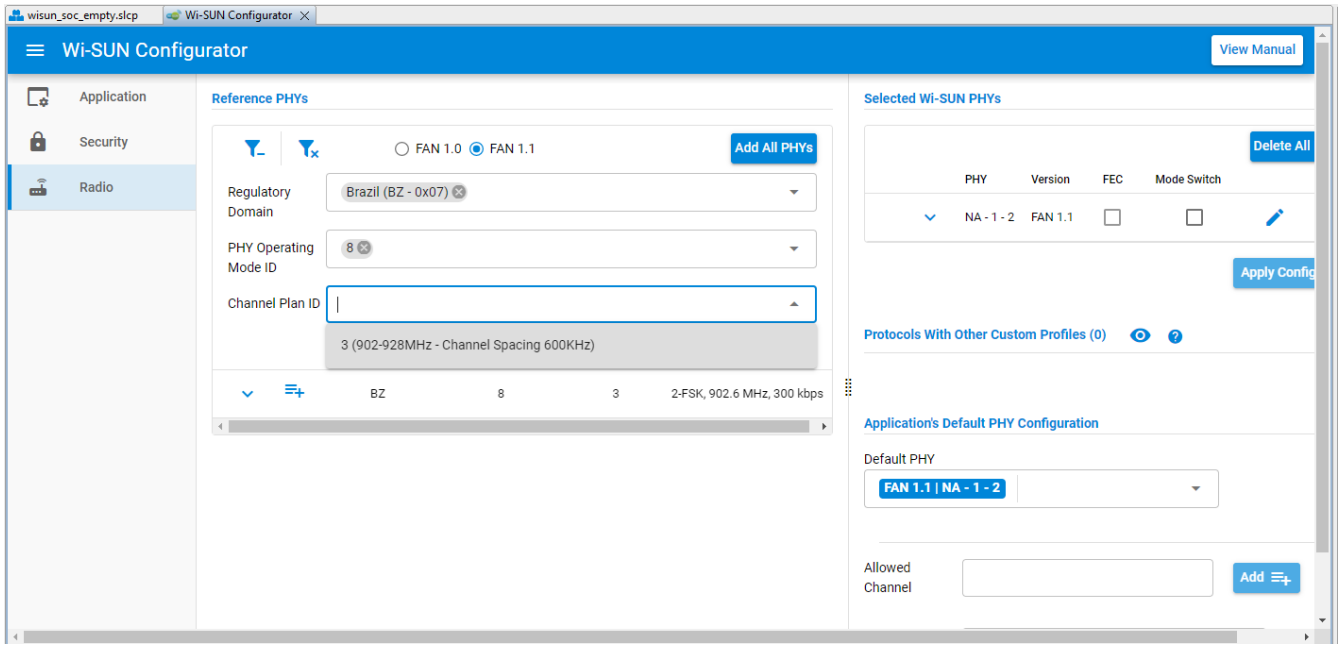
4. Select **phy_mode_id_8** in the **PHY Operating Mode ID** dropdown.

Copyright © 2025 Silicon Laboratories. All rights reserved.

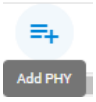


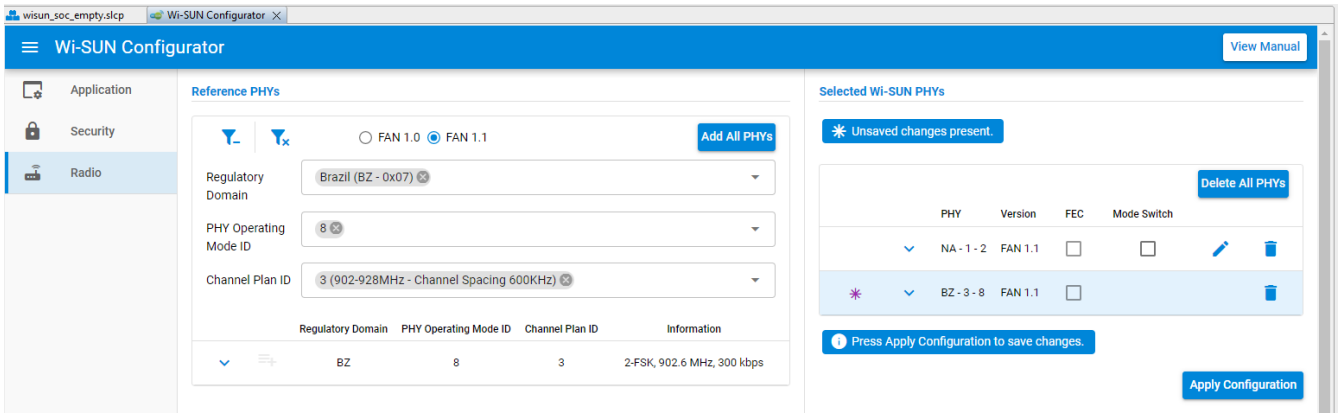
The list of possible PHYs at the bottom is further reduced to only those with '8' as their PHY Operating Mode ID (a single one).

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.

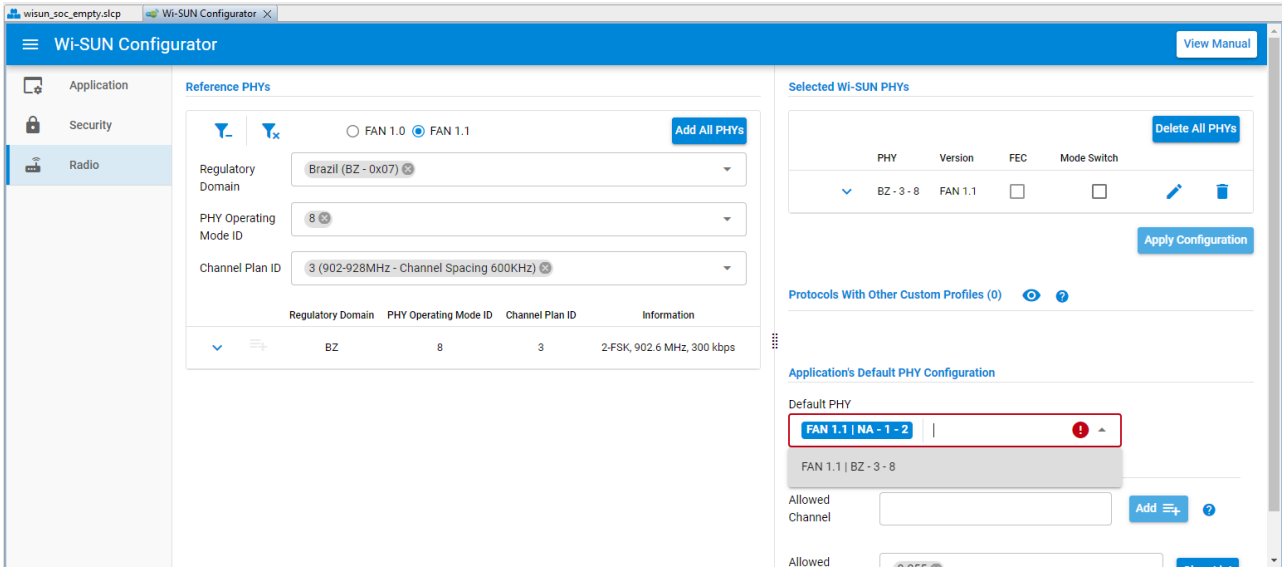


Copyright © 2025 Silicon Laboratories. All rights reserved.

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



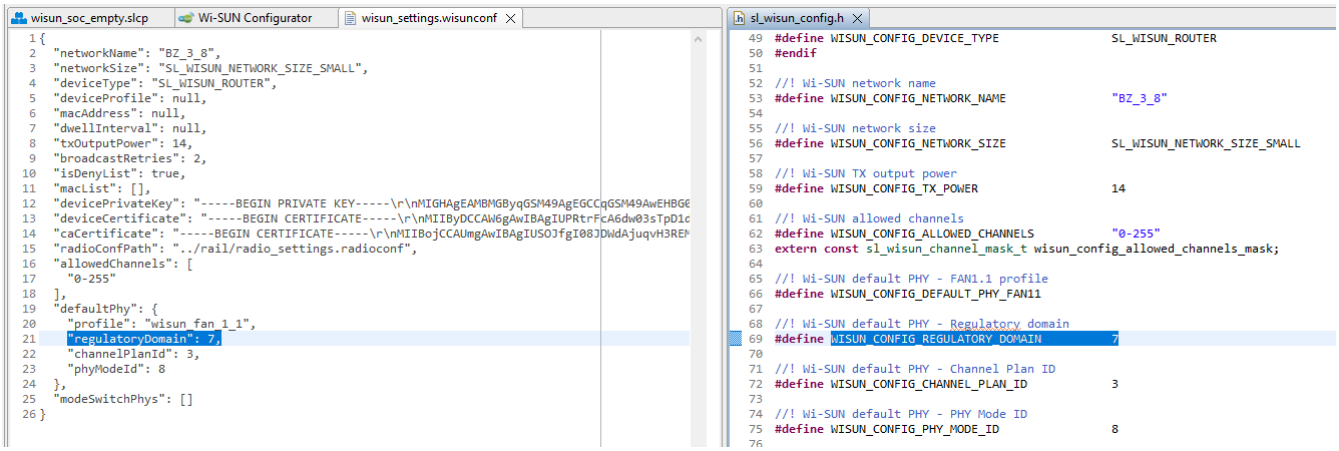
- The icon you just clicked turns light gray, indicating that this PHY is already in the list.
- A message is present in **Selected Wi-SUN PHYs** to indicate unsaved changes.
- A message informs you that you should save your new configuration by clicking the **Apply Configuration** button.
- Once saved, 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
$ 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.

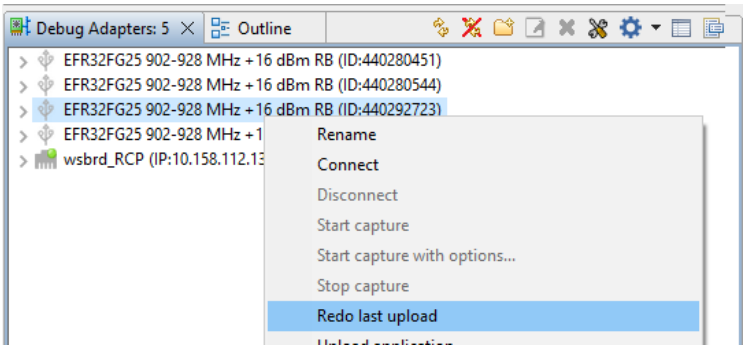
Take a minute to flash your new application to the Router to see if it displays the new network name and can connect to your new Wi-SUN network.

> 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



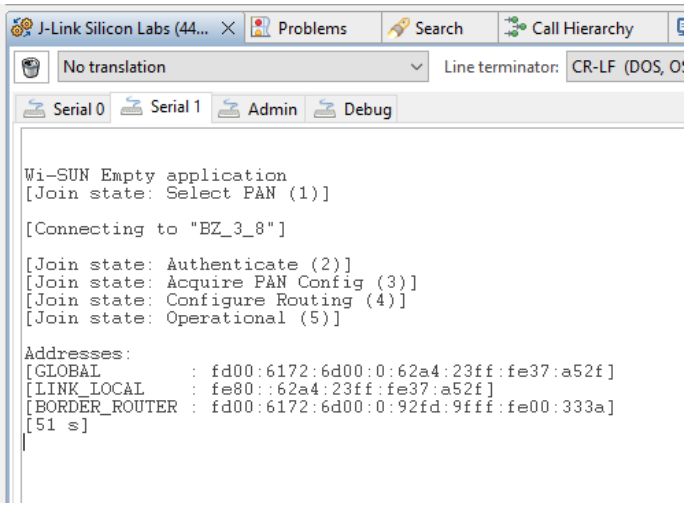
> TIP: Understand 'redo last upload' as 'reload the binary from the same location, with the same name'. If the binary content changed (your case), it will be updated.

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)

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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**.

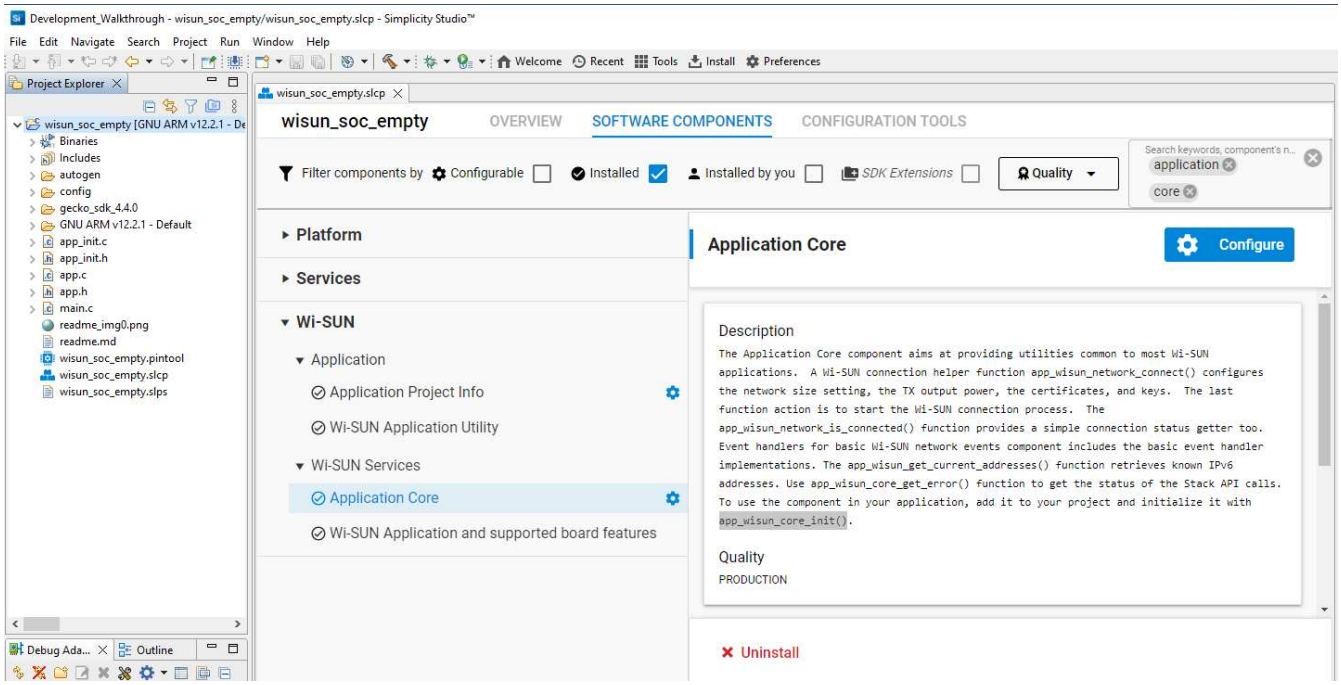
This is a lecture on how the **Wi-SUN Application Core component** uses the Wi-SUN Stack API functions to achieve successful connection. Key points to understand are:

- 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.

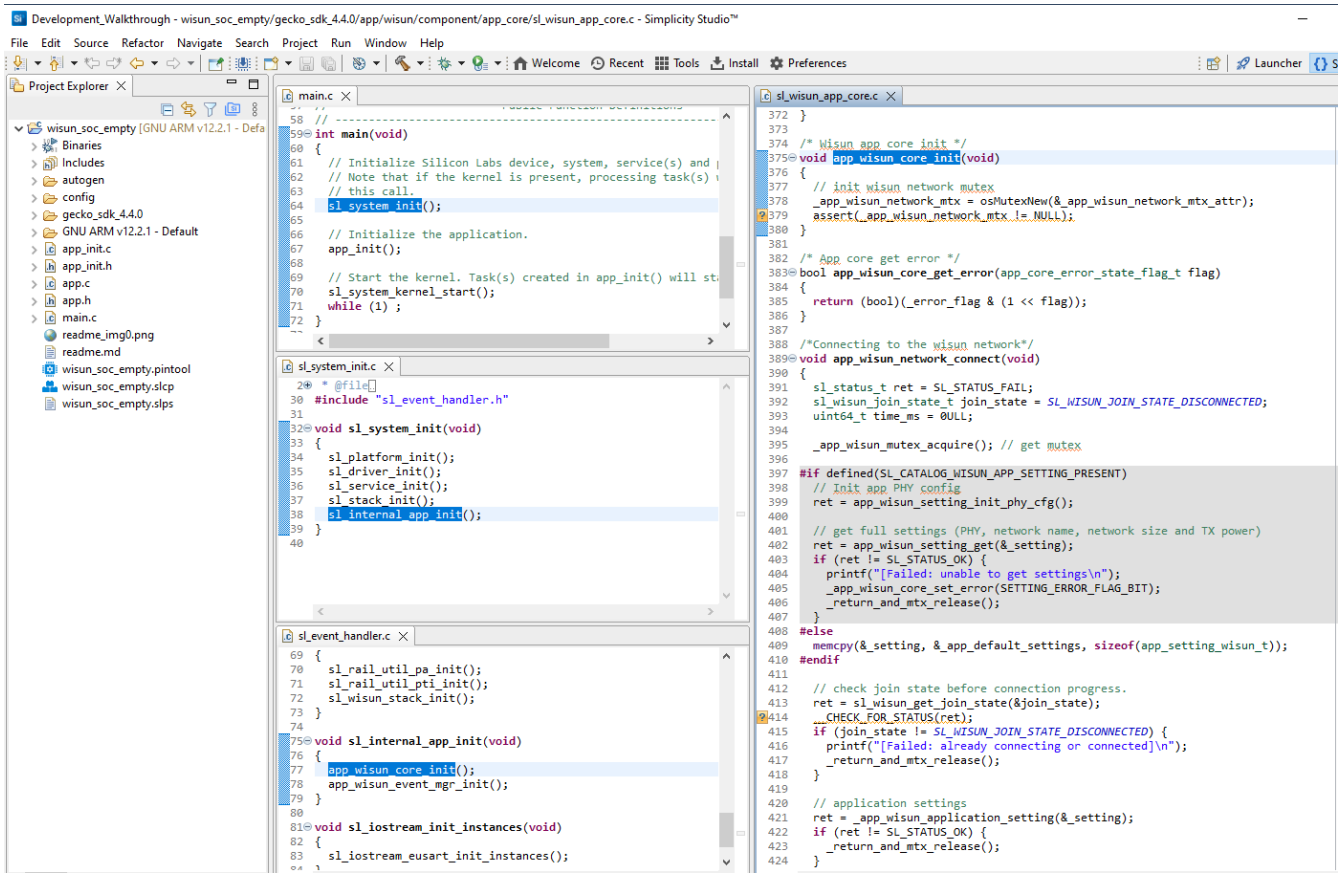


2. Check the content of `wisun_soc_empty.slcp` with a text editor.

Copyright © 2025 Silicon Laboratories. All rights reserved.

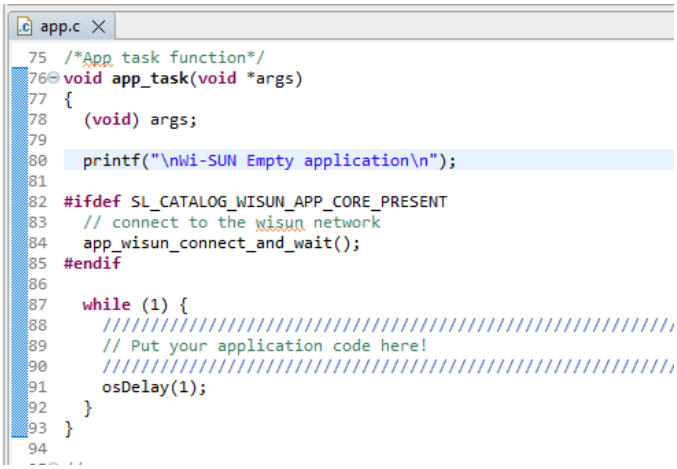
```
wisun_soc_empty.slcp
45 - {id: restrictions_profile_wisun_fan_1_0}
46 - {id: restrictions_profile_wisun_fan_1_1}
47 - {id: wisun_stack_debug}
48 - {id: silabs_core_sl_malloc}
49 - {id: EFR32FG25B222F1920IM56}
50 - {id: printf}
51 - {id: wisun_stack}
52 - {id: wisun_config}
53 - {id: sleeptimer}
54 - {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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



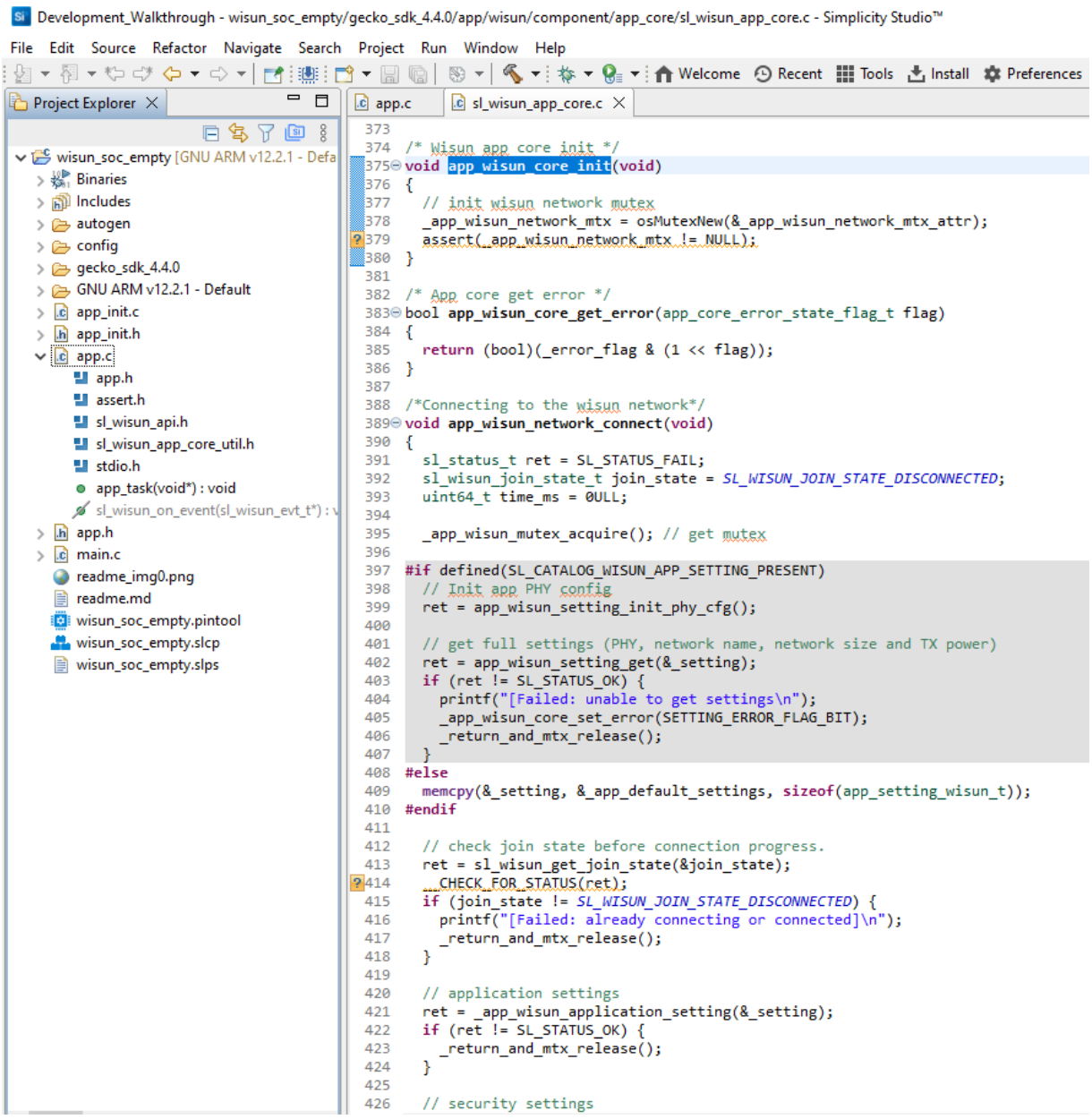
The Wi-SUN Application Core component uses the Wi-SUN Configurator settings to achieve connection to a Wi-SUN Network with only GUI actions. This is the simplest way to connect to any Wi-SUN Network.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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));
```

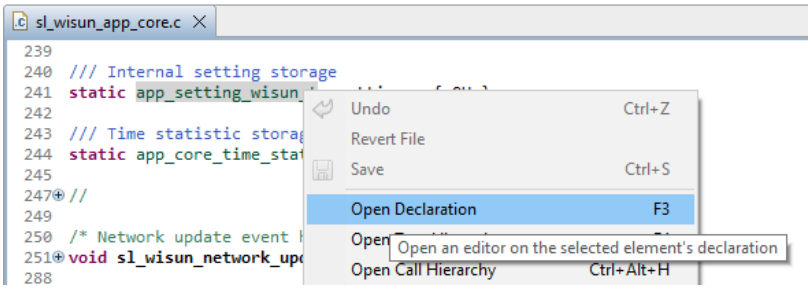
This line copies the `_app_default_settings` into `_setting`, a `app_setting_wisun_t` structure defined earlier as:

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

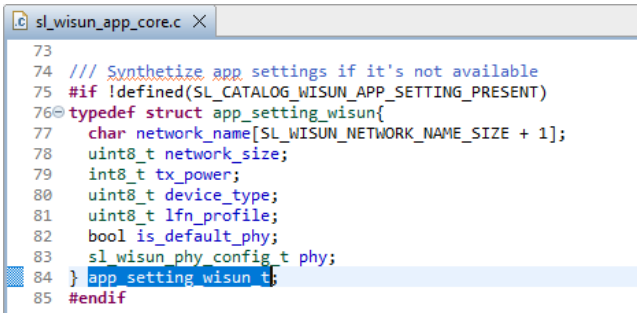
`app_setting_wisun_t` structure

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

Copyright © 2025 Silicon Laboratories. All rights reserved.



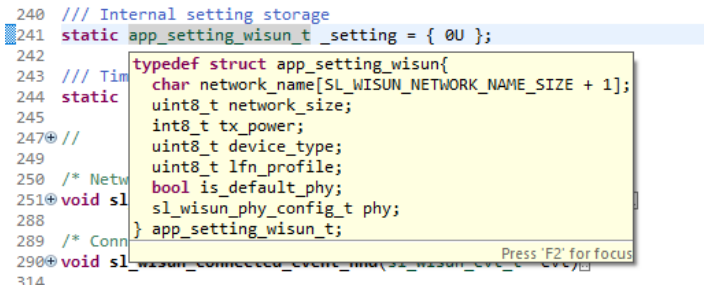
This brings you higher up in `/wisun_soc_empty/gecko_sdk_x.y.z/app/wisun/component/app_core/sl_wisun_app_core.c` to the declaration of the structure.



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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

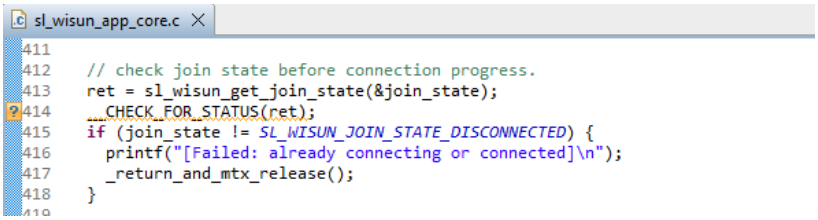
- `phy->fan11.reg_domain`
- `phy->fan11.chan_plan_id`
- `phy->fan11.phy_mode_id`

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

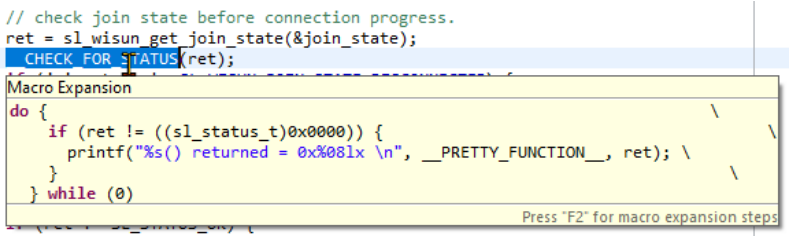
> TIP: When moving around files in Simplicity Studio, use **Alt+left_arrow** or **Alt+Right_arrow** to move between recent locations.

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).

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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.

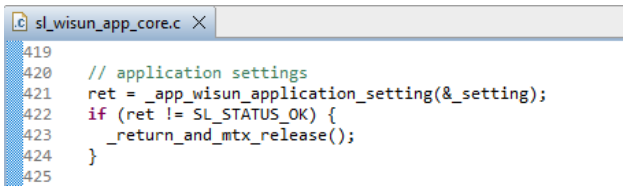
A typical pattern when using a mutex is:

- 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).

Copyright © 2025 Silicon Laboratories. All rights reserved.

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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

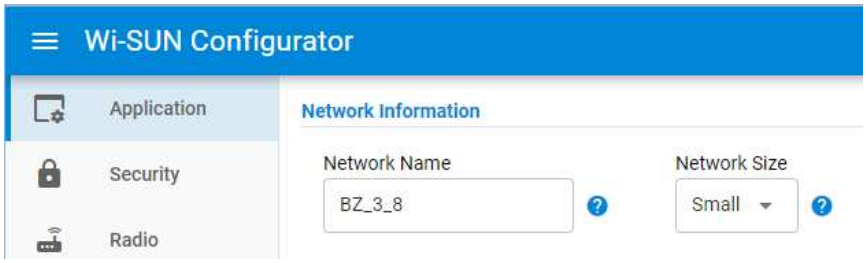
sl_wisun_connection_params_api.h

```
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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

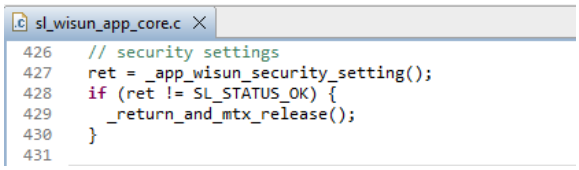
Copyright © 2025 Silicon Laboratories. All rights reserved.

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.



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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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 }
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

```
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()`.

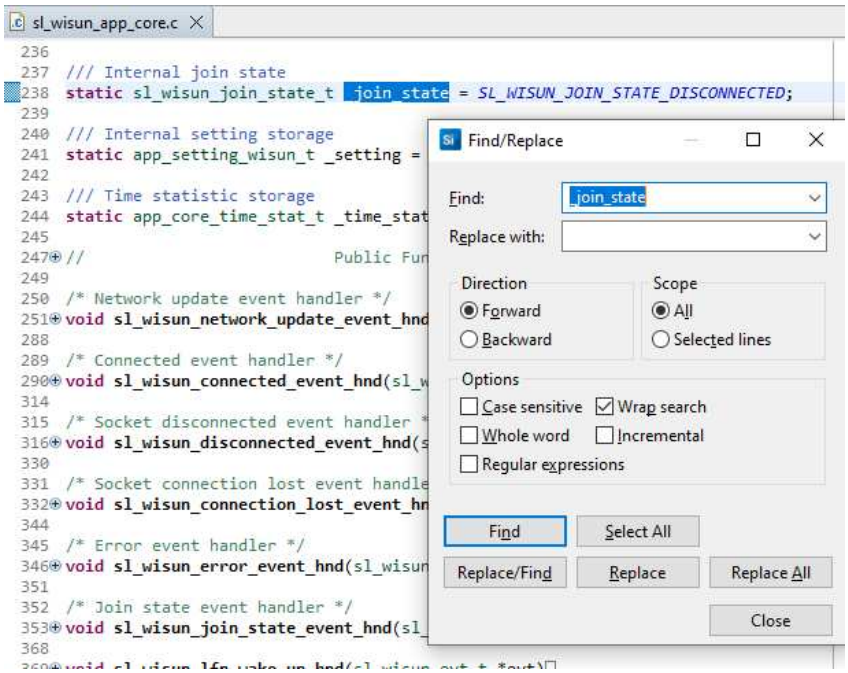
Copyright © 2025 Silicon Laboratories. All rights reserved.

```c
sl_wisun_join_state_t app_wisun_get_join_state(void)
{
  sl_wisun_join_state_t join_state = SL_WISUN_JOIN_STATE_DISCONNECTED;
  _app_wisun_mutex_acquire();
  join_state = _join_state;
  _app_wisun_mutex_release();
  return join_state;
}
```

`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`.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_types.h

```
248
249 /// Enumerations for join state
250 typedef enum {
251   /// Join state 0: Disconnected
252   SL_WISUN_JOIN_STATE_DISCONNECTED       = 0,
253   /// Join state 1: Select PAN
254   SL_WISUN_JOIN_STATE_SELECT_PAN         = 1,
255   /// Join state 2: Authenticate
256   SL_WISUN_JOIN_STATE_AUTHENTICATE       = 2,
257   /// Join state 3: Acquire PAN config
258   SL_WISUN_JOIN_STATE_ACQUIRE_PAN_CONFIG = 3,
259   /// Join state 4: Configure routing
260   SL_WISUN_JOIN_STATE_CONFIGURE_ROUTING  = 4,
261   /// Join state 5: Operational
262   SL_WISUN_JOIN_STATE_OPERATIONAL        = 5,
263   /// Join state 4: Preferred parent selection
264   SL_WISUN_JOIN_STATE_PARENT_SELECT      = 41,
265   /// Join state 4: DHCP address acquisition
266   SL_WISUN_JOIN_STATE_DHCP               = 42,
267   /// Join state 4: Address registration
268   SL_WISUN_JOIN_STATE_EARO               = 43,
269   /// Join state 4: DAO registration
270   SL_WISUN_JOIN_STATE_DAO                = 44
271 } sl_wisun_join_state_t;
272
```

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
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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```c
/// Wi-SUN Message API common header
SL_PACK_START(1)
typedef struct {
  /// Total length of the message in bytes, this field included
  uint16_t length;
  /// ID (request, confirmation, indication) of the message
  uint8_t id;
  /// Processing metadata for the message
  uint8_t info;
} SL_ATTRIBUTE_PACKED sl_wisun_msg_header_t;
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()` .

```c
/* Wi-SUN event callback */
void sl_wisun_on_event(sl_wisun_evt_t *evt)
{
  app_wisun_event_id_t idx = _decode_ind((sl_wisun_msg_ind_id_t) evt->header.id);

  app_wisun_event_mgr_mutex_lock();

  if (EVENT_IDX_NOTVALID == idx) {
    printf("[Unknown event: %d]\r\n", evt->header.id);
  } else {
    _wisun_events[idx].callback(evt);
    if (_wisun_events[idx].custom_callback != NULL) {
      _wisun_events[idx].custom_callback(evt);
    }
  }

  app_wisun_event_mgr_mutex_unlock();
}
```

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').

```c
/* Decode wisun event id to lookup index */
__STATIC_INLINE app_wisun_event_id_t _decode_ind(const sl_wisun_msg_ind_id_t ind)
{
  switch (ind) {
    case SL_WISUN_MSG_NETWORK_UPDATE_IND_ID:                return EVENT_IDX_NETWORK_UPDATE;
    case SL_WISUN_MSG_CONNECTED_IND_ID:                     return EVENT_IDX_CONNECTED;
    case SL_WISUN_MSG_SOCKET_DATA_IND_ID:                   return EVENT_IDX_SOCKET_DATA;
    case SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID:         return EVENT_IDX_SOCKET_DATA_AVAILABLE;
    case SL_WISUN_MSG_SOCKET_CONNECTED_IND_ID:              return EVENT_IDX_SOCKET_CONNECTED;
    case SL_WISUN_MSG_SOCKET_CONNECTION_AVAILABLE_IND_ID:   return EVENT_IDX_SOCKET_CONNECTION_AVAILABLE;
    case SL_WISUN_MSG_SOCKET_CLOSING_IND_ID:                return EVENT_IDX_SOCKET_CLOSING;
    case SL_WISUN_MSG_DISCONNECTED_IND_ID:                  return EVENT_IDX_DISCONNECTED;
    case SL_WISUN_MSG_CONNECTION_LOST_IND_ID:               return EVENT_IDX_CONNECTION_LOST;
    case SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID:              return EVENT_IDX_SOCKET_DATA_SENT;
    case SL_WISUN_MSG_ERROR_IND_ID:                         return EVENT_IDX_ERROR;
    case SL_WISUN_MSG_JOIN_STATE_IND_ID:                    return EVENT_IDX_JOIN_STATE;
    case SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID:           return EVENT_IDX_REGULATION_TX_LEVEL;
    case SL_WISUN_MSG_LFN_WAKE_UP_IND_ID:                   return EVENT_IDX_LFN_WAKE_UP;
    default:                                                return EVENT_IDX_NOTVALID;
  }
}
```

> 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()` :

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

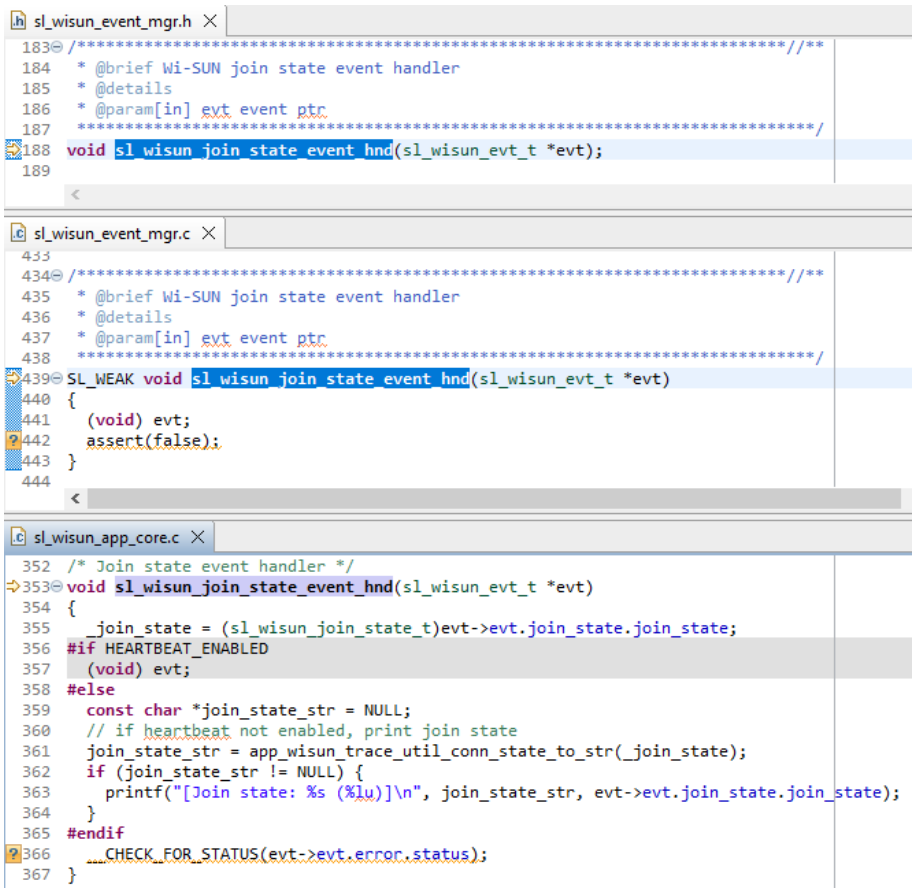
Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

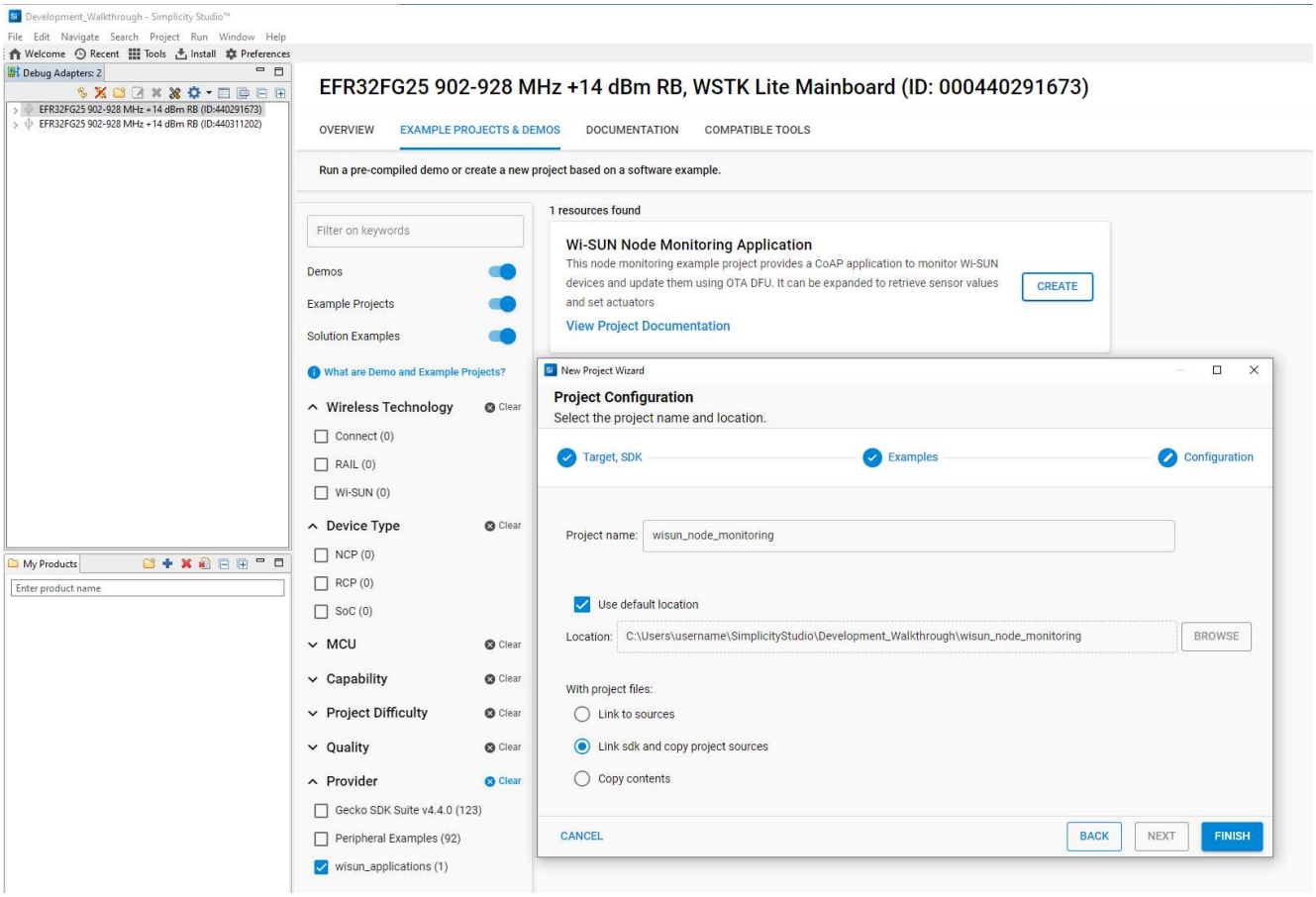
Add a Custom Application

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

    - 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

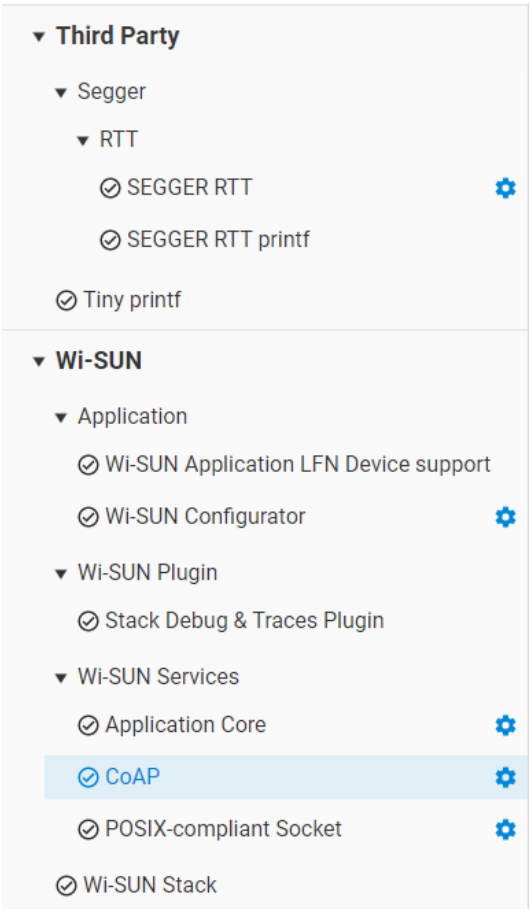
The following software components are required to compile the application with the added code:

- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
__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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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__)
```

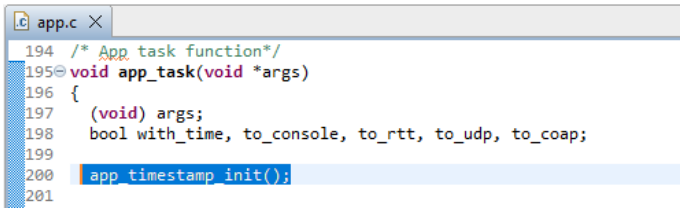
The above makes it easy in the application to:

- 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));
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.
  - If the previous join state was 'OPERATIONAL':
    - Store the time in `disconnection_time_sec`.

Copyright © 2025 Silicon Laboratories. All rights reserved.

    - 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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()`.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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;
}
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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;
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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;
}
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Add CoAP Resources

Add CoAP Resources to the Device Application

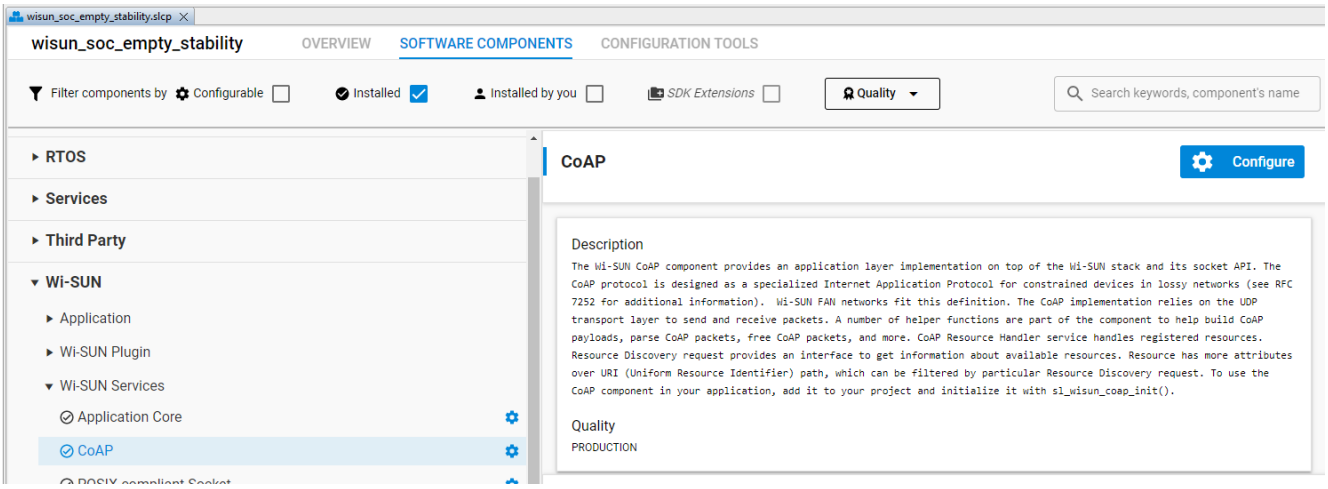
Using the Send status / Retrieve UDP notifications technique, the devices are automatically sending messages to the Border Router (by default, another destination is possible). In a large Wi-SUN network, this may not be convenient, or the `auto_send_sec` value would be set to a very high value, such 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
// <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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
/**
 * \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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
./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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
# 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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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
```

> On more recent linux kernels, use libcoap3 and the commands with the `-notls` suffix to keep it simple (without TLS authentication), considering that your Wi-SUN network already supports encryption.

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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

- `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)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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>]`

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
./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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

> On more recent linux kernels, use libcoap3 and the commands with the `-notls` suffix to keep it simple (without TLS authentication), considering that your Wi-SUN network already supports encryption.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

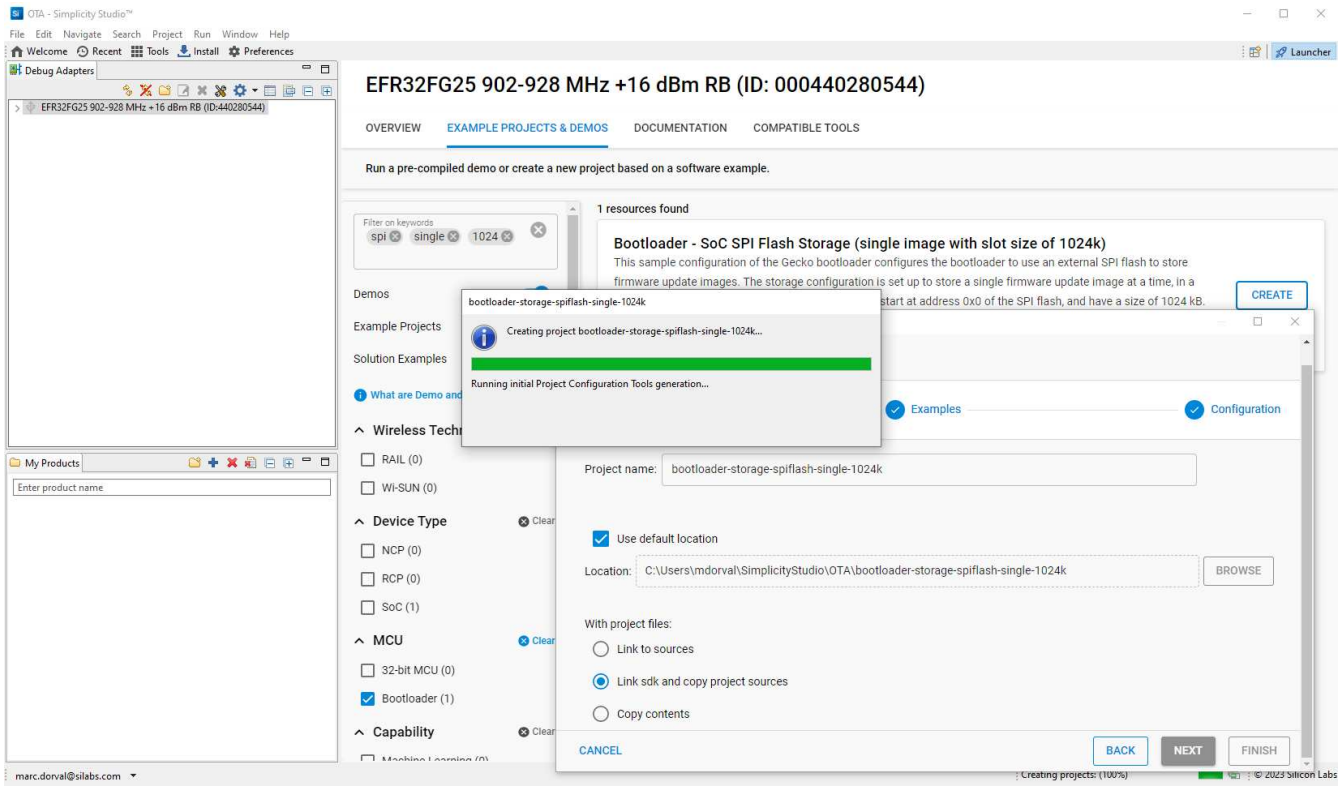
You need a Bootloader Application using SPI Flash for a single image of max 1024k bytes. This is to be manually flashed **once** to the part to allow OTA.

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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.

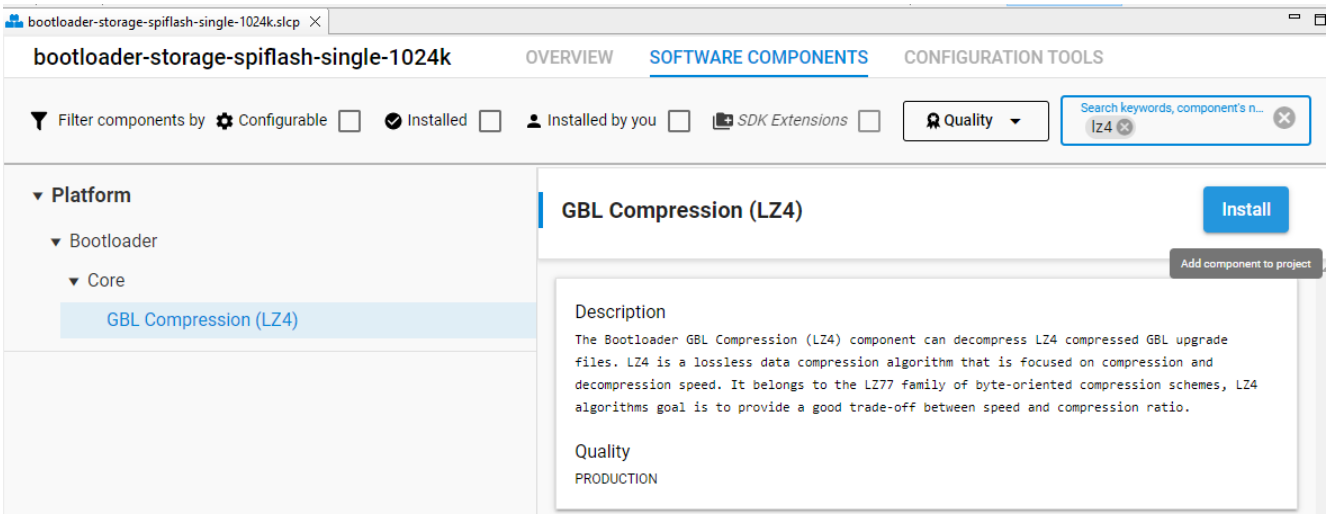
- The Base Address is defined as `BTL_STORAGE_BASE_ADDRESS`.
- The memory slots to store the images are defined as well.
- Only `SLOT0` is enabled, with a `SLOT0_SIZE` of `1048576` = `1024 * 1024` = 1024 kBytes, starting at `SLOT0_START 0`.

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.

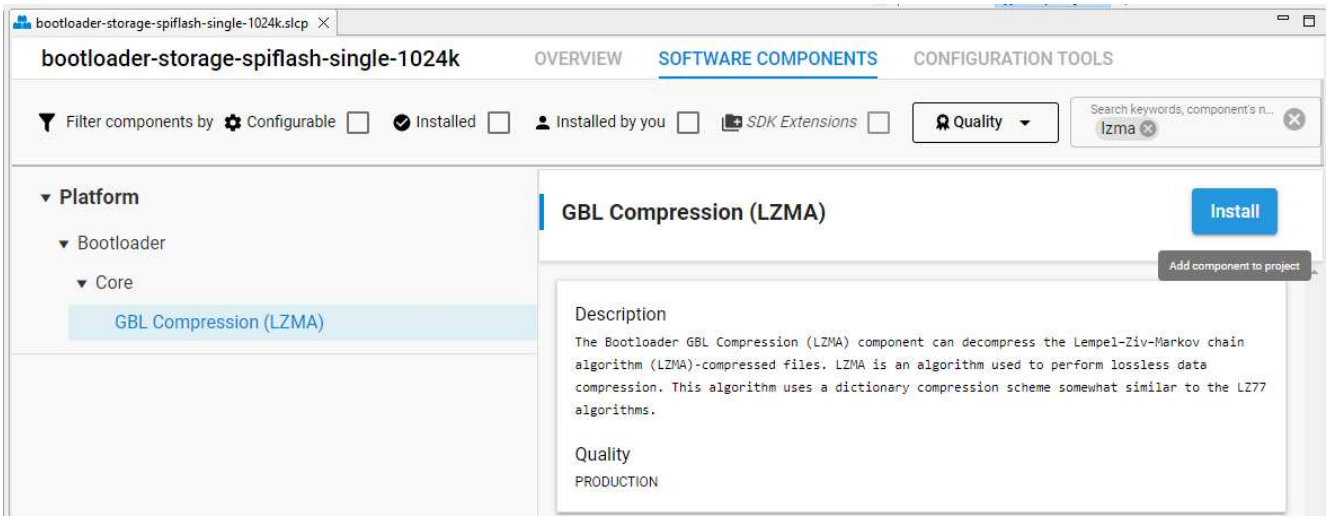
Copyright © 2025 Silicon Laboratories. All rights reserved.



Add LZMA Compression to Bootloader

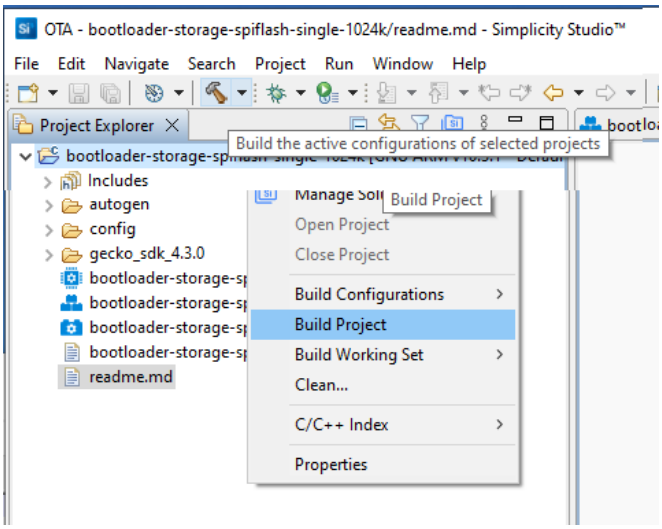
By default, the bootloader projects don't include the most efficient LZMA compression algorithm, so install it if you want to use it.

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

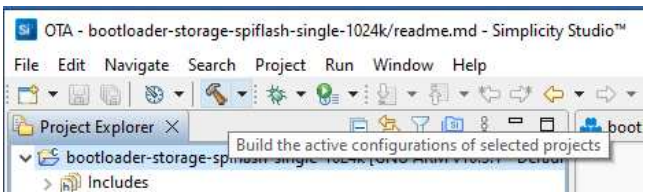


Build the Bootloader Project

Copyright © 2025 Silicon Laboratories. All rights reserved.



or



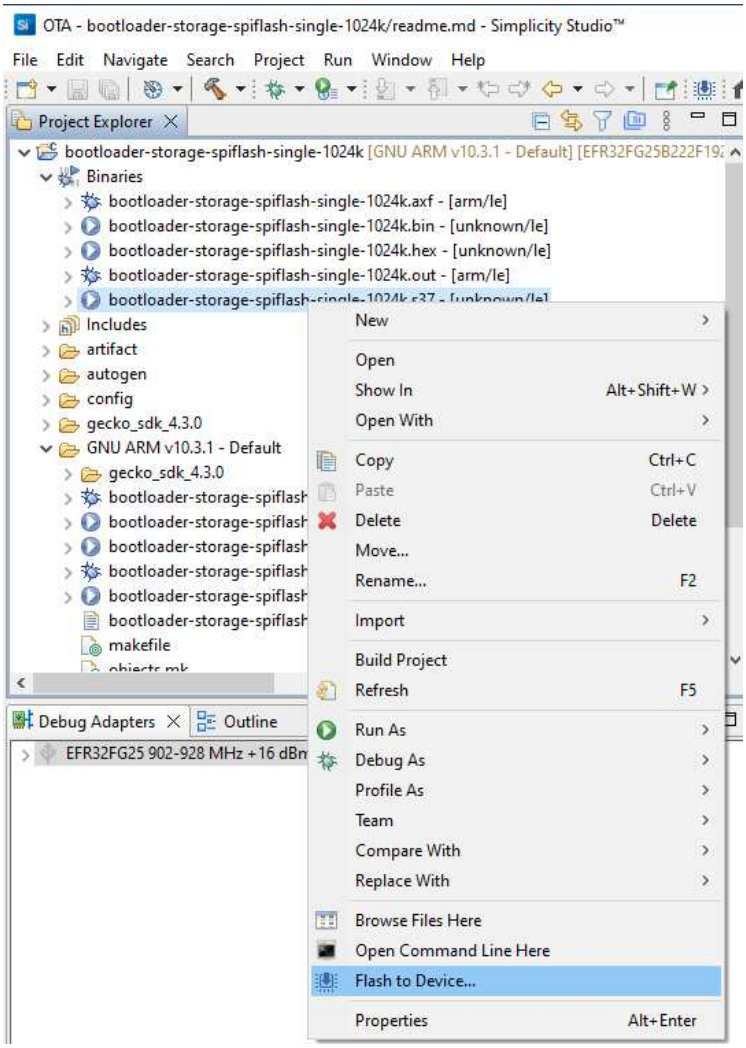
See Simplicity Studio 5 Users Guide for details.

Flash the Bootloader Binary to the Debug Adapter

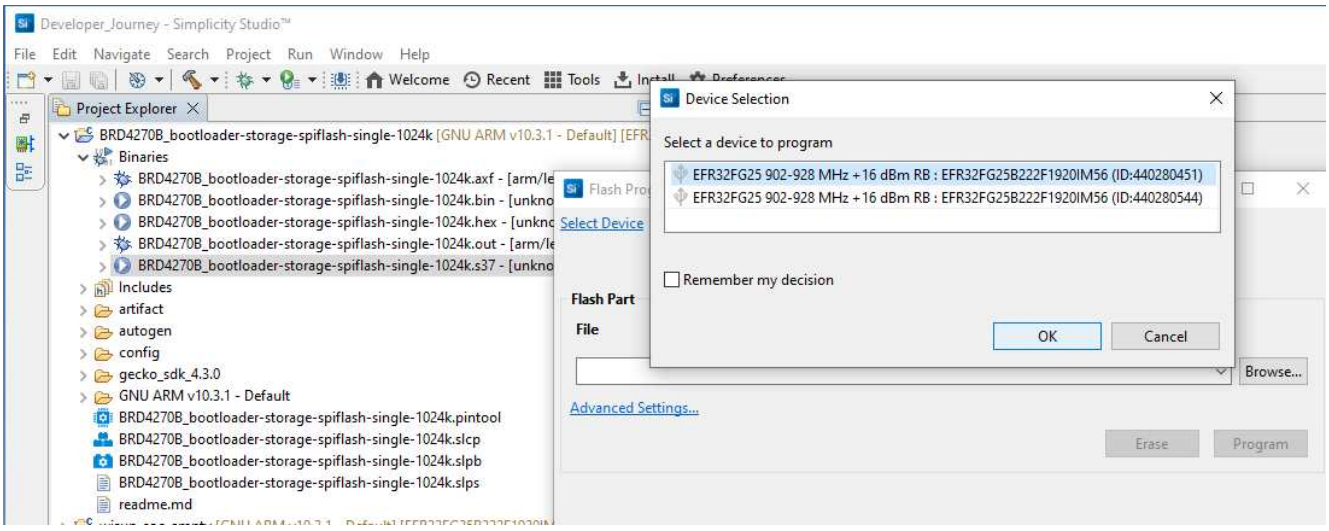
A new `Binaries` folder appeared in the project. This corresponds to the binaries present under `GNU ARM v10.3.1 - Default`.

- On Series 1 (xG12, for example), select the `<projectname>-combined.s37` binary (see Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) for details) or Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher.
- On Series 2 (xG25 or xG28, for example), select the `<projectname>.s37` binary.

Copyright © 2025 Silicon Laboratories. All rights reserved.

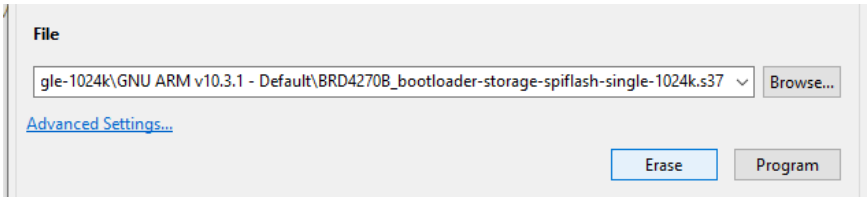


If you have several devices in your **Debug Adapters** window, you will be asked to select the device.

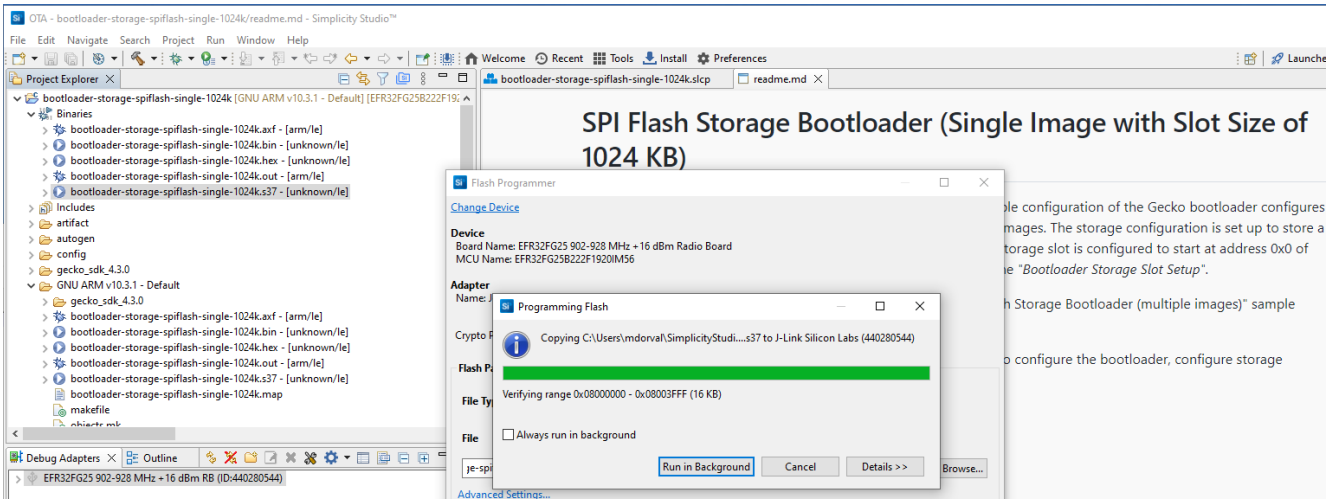


Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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

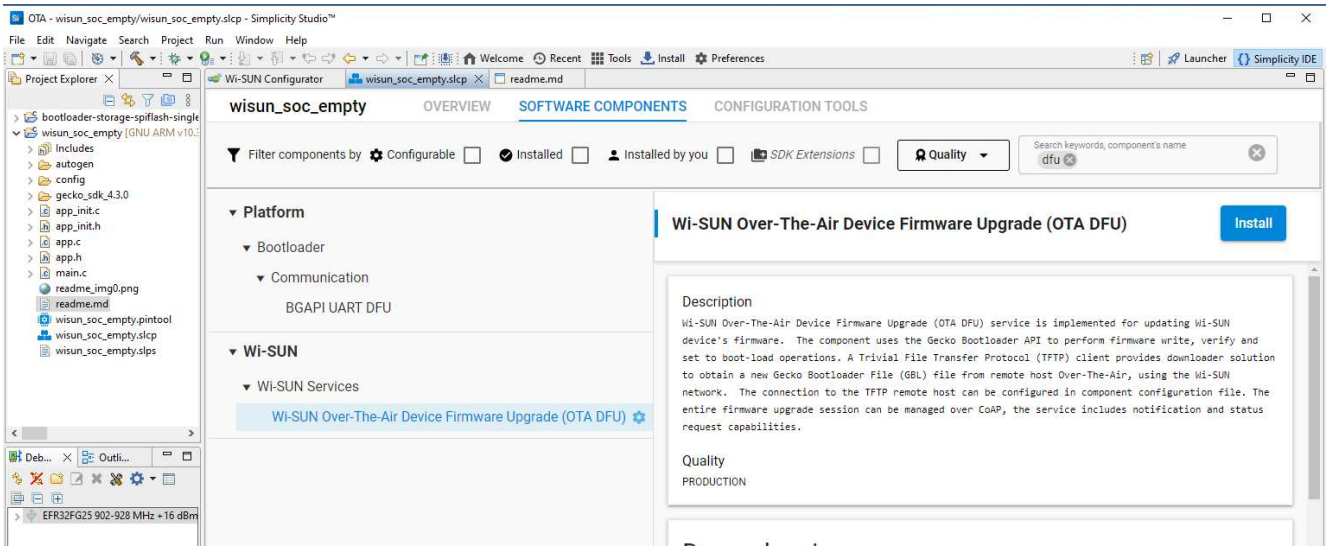
To perform OTA upgrades:

- 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

Copyright © 2025 Silicon Laboratories. All rights reserved.



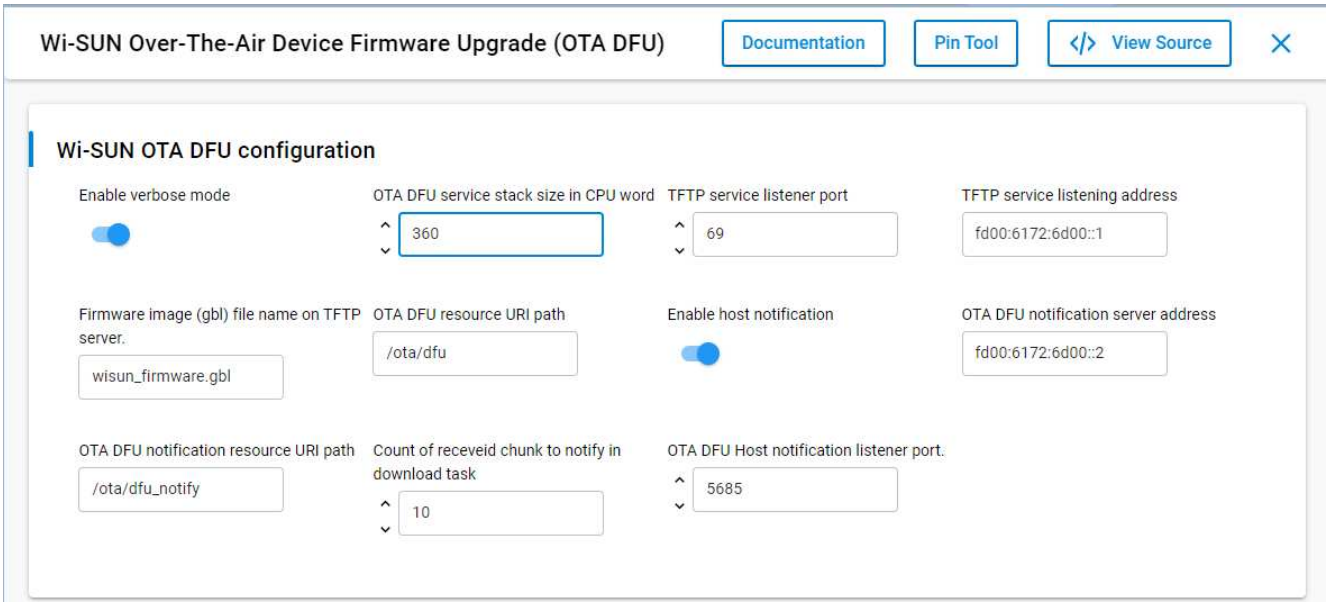
Check the OTA DFU Settings

> 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`.



Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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**.

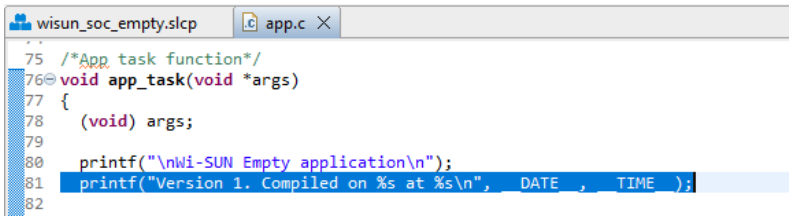
Set the Wi-SUN Configuration

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__);
```

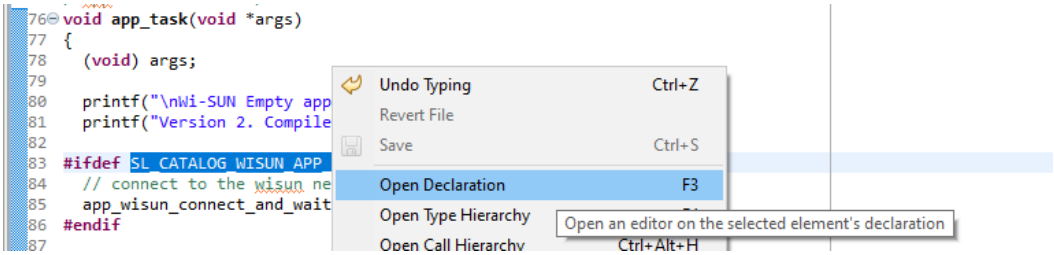


Copyright © 2025 Silicon Laboratories. All rights reserved.

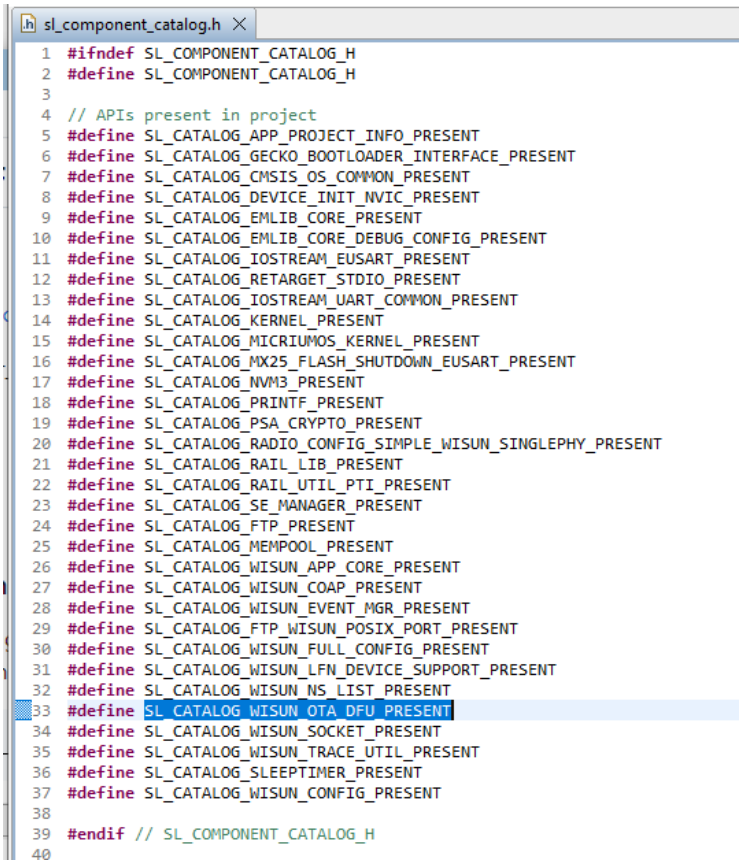
(Optional) Add a Startup Text Indicating OTA DFU Support

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.

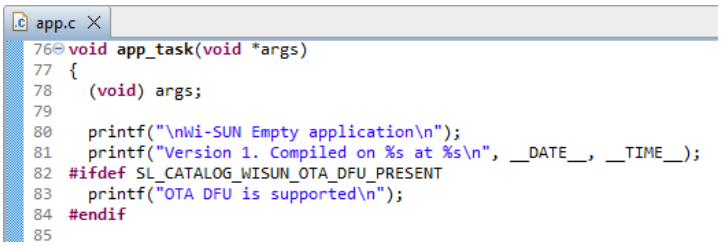


> TIP: You also see that `SL_CATALOG_WISUN_COAP_PRESENT` is declared on line 27, indicating that the CoAP component is also present. CoAP is a dependency for the OTA DFU component, so it has been installed automatically when you added OTA DFU (if not previously installed).

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
#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()`

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
#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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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);
  }
}
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

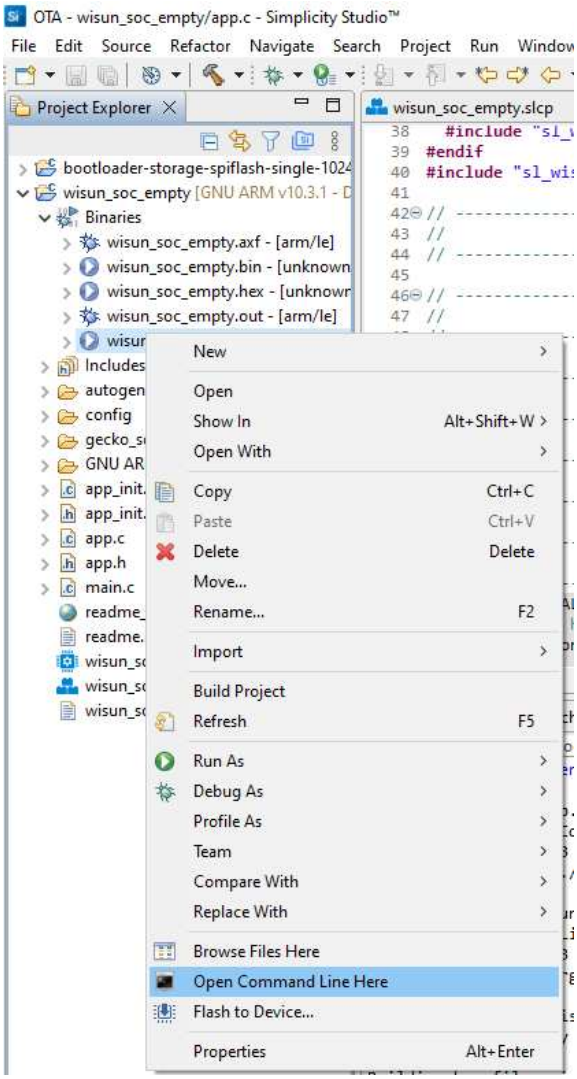
```
Building s37 file: wisun_soc_empty.s37
arm-none-eabi-objcopy -O srec "wisun_soc_empty.axf" "wisun_soc_empty.s37"

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).

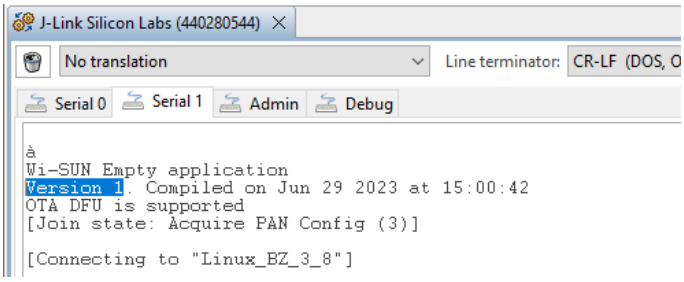
> NOTE: If you use the **Erase** Feature, you will need to re-flash the bootloader then flash the application.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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

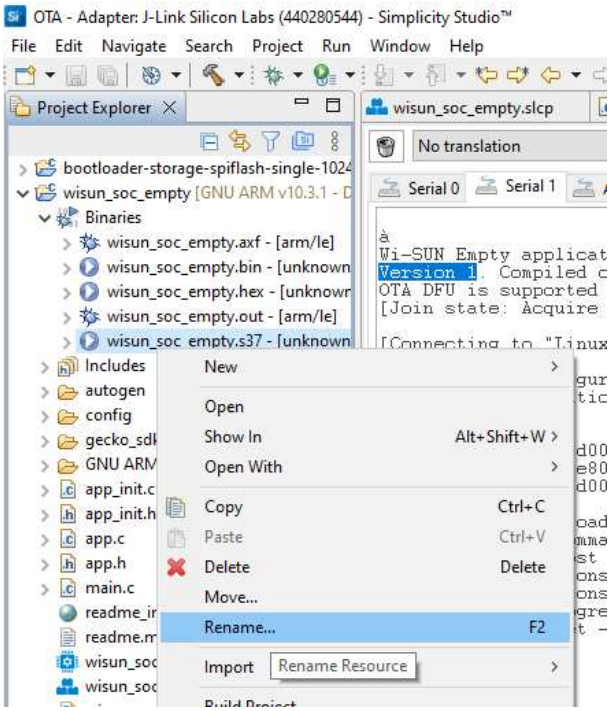
The 'Wi-SUN SoC Empty' example application is set for auto-connection to the Wi-SUN Network, so you see it starting to connect.

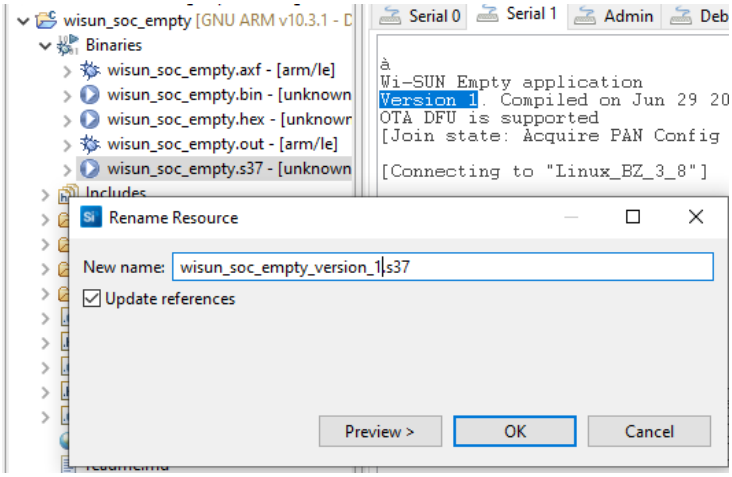


Rename the 'Version 1'Binary to Keep It

Copyright © 2025 Silicon Laboratories. All rights reserved.

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__);
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

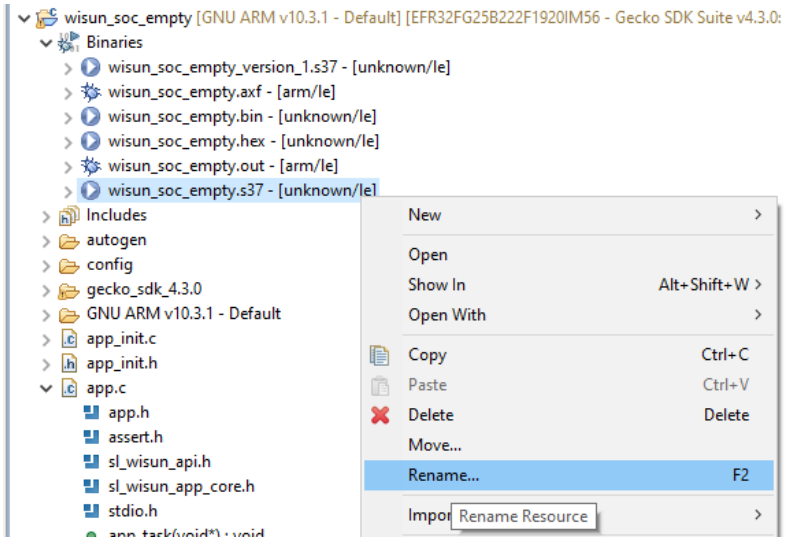
```
app.c
76 void app_task(void *args)
77 {
78   (void) args;
79
80   printf("\nWi-SUN Empty application\n");
81   printf("Version 2. Compiled on %s at %s\n", __DATE__, __TIME__);
82 #ifdef SL_CATALOG_WISUN_OTA_DFU_PRESENT
83   printf("OTA DFU is supported\n");
84 #endif
```

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,

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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
```

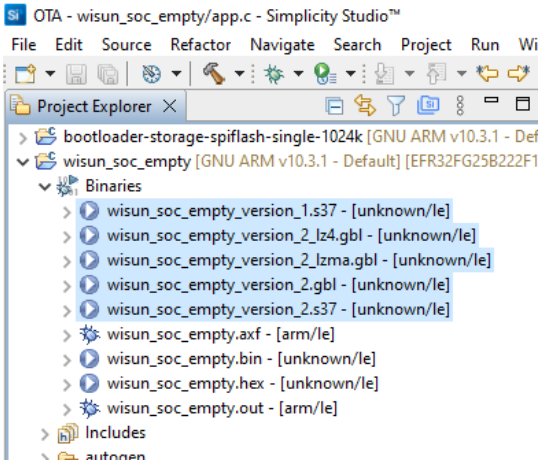
Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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

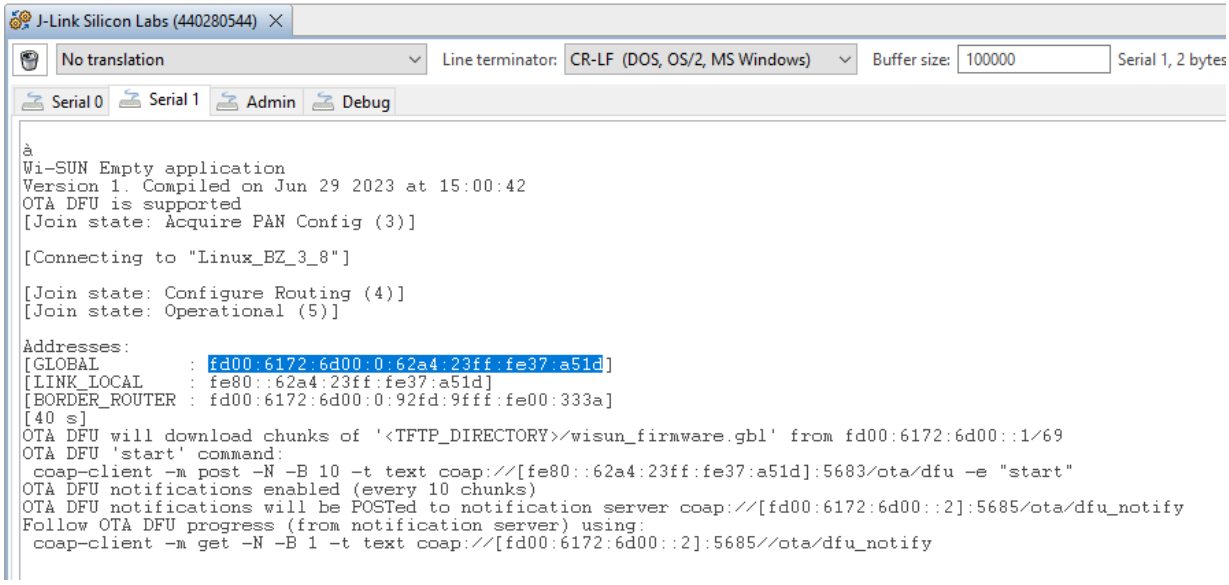
In `wsbrd_cli status`, the MAC address of the device should be visible:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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`

Copyright © 2025 Silicon Laboratories. All rights reserved.

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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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'.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
[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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
[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

Copyright © 2025 Silicon Laboratories. All rights reserved.

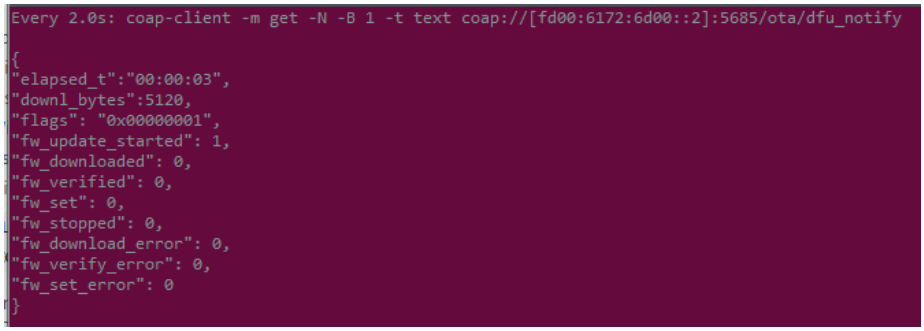
- The first two notifications have been sent to the notification server's IPV6

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
}
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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`

Copyright © 2025 Silicon Laboratories. All rights reserved.

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 install command	Command line: `coap-client -m post ... -e "install"`	OTA DFU Source file: SL_WISUN_OTA_DFU_PAYLOAD_STR_REBOOT_AND_INSTALL `<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`

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN Configurator

Wi-SUN Configurator

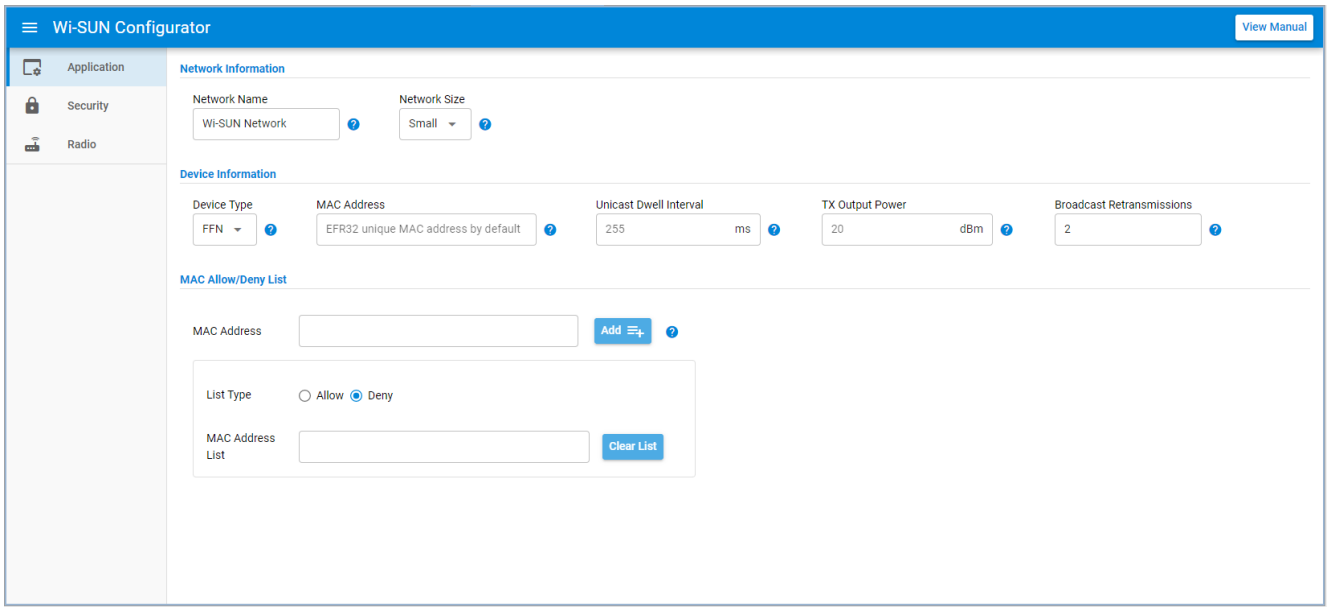
When creating a new Wi-SUN project, a Wi-SUN Configurator is added to it by default. The Wi-SUN Configurator provides an interface to the Wi-SUN application's main settings through three panels: Application, Security, and Radio. For some examples, the Wi-Sun Configurator only displays the Radio panel. 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

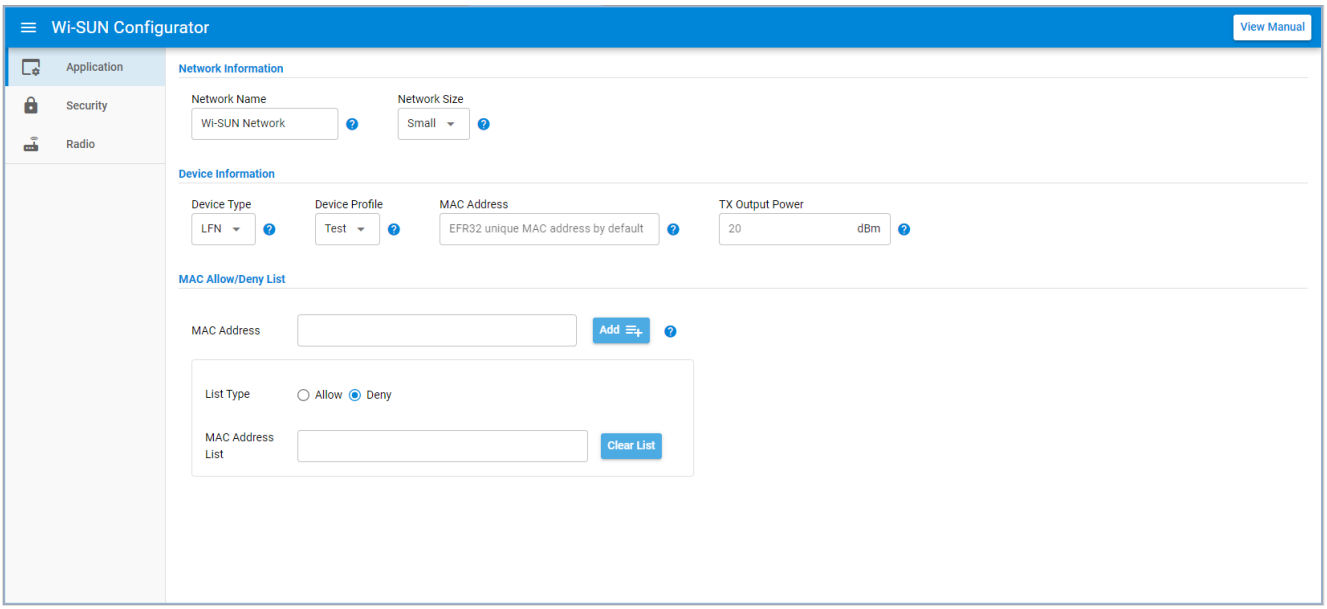
The Application panel exposes multiple Wi-SUN stack settings associated with the application, such as:

- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

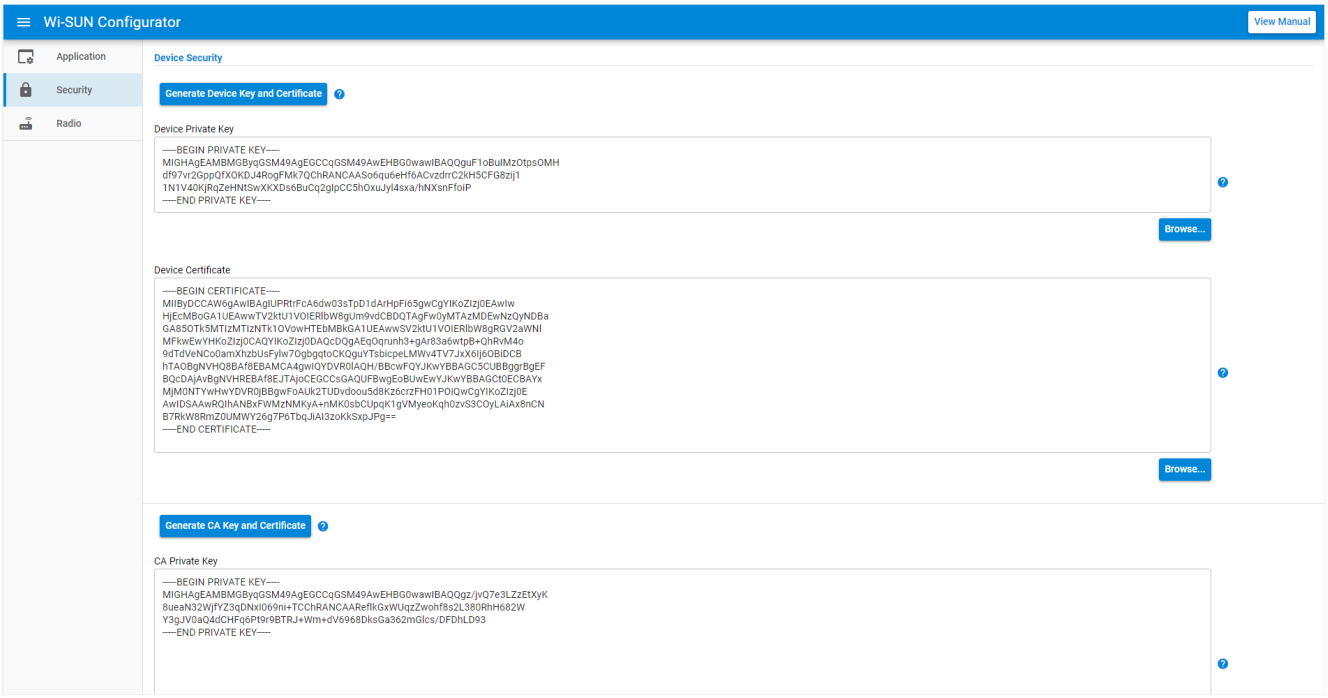
Security Panel

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.

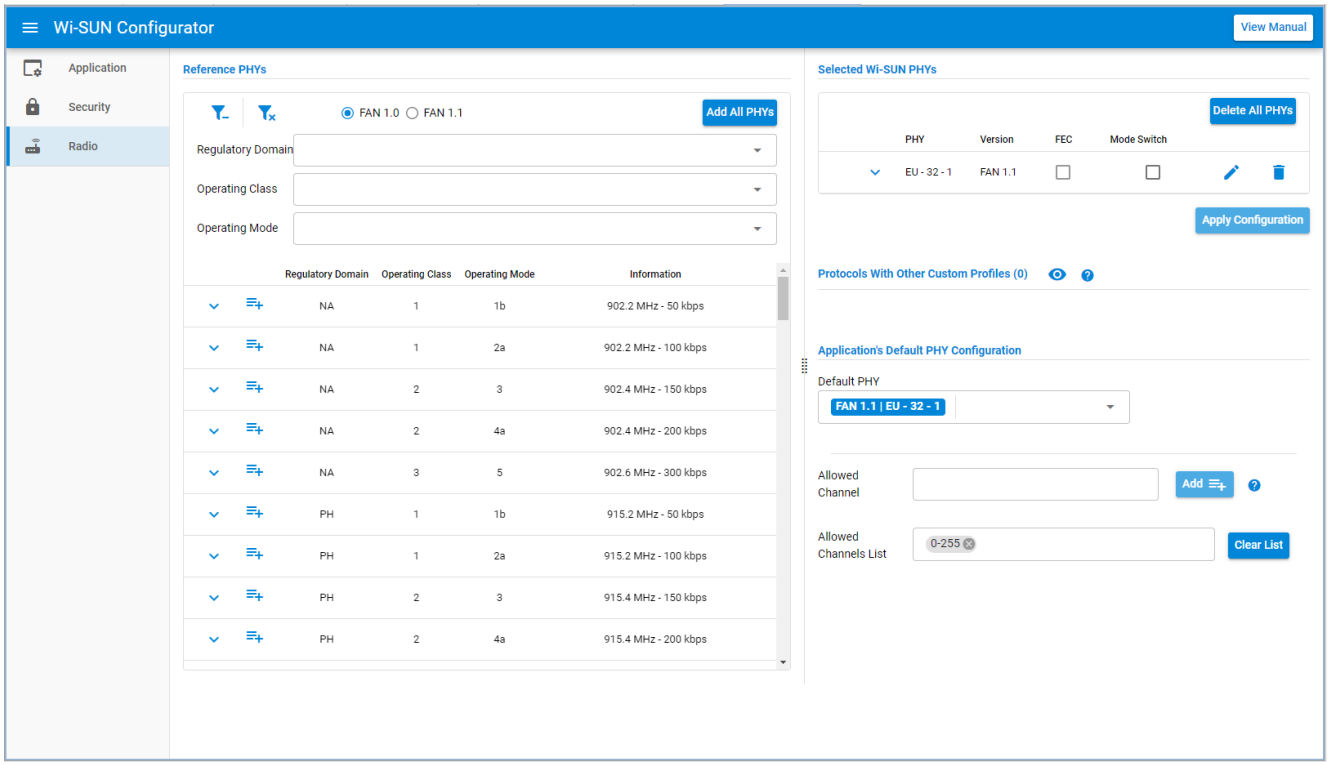
Copyright © 2025 Silicon Laboratories. All rights reserved.



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.



Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

FAN 1.0 Node Certification

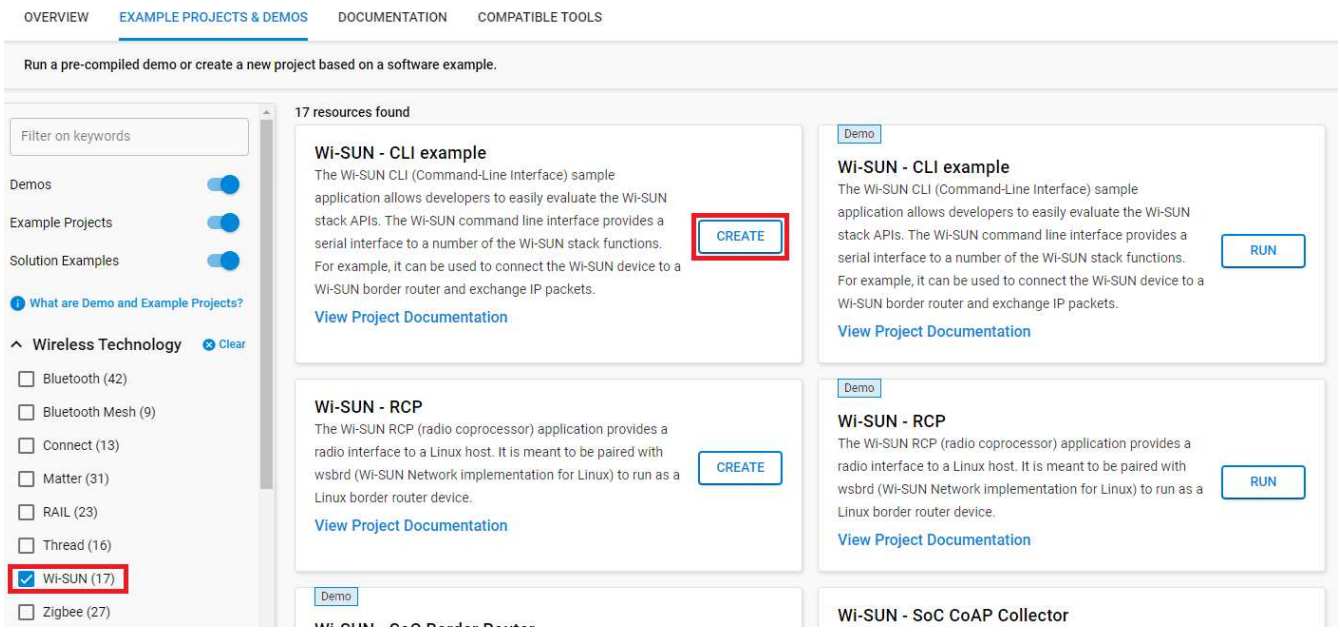
Wi-SUN FAN 1.0 Node Certification

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

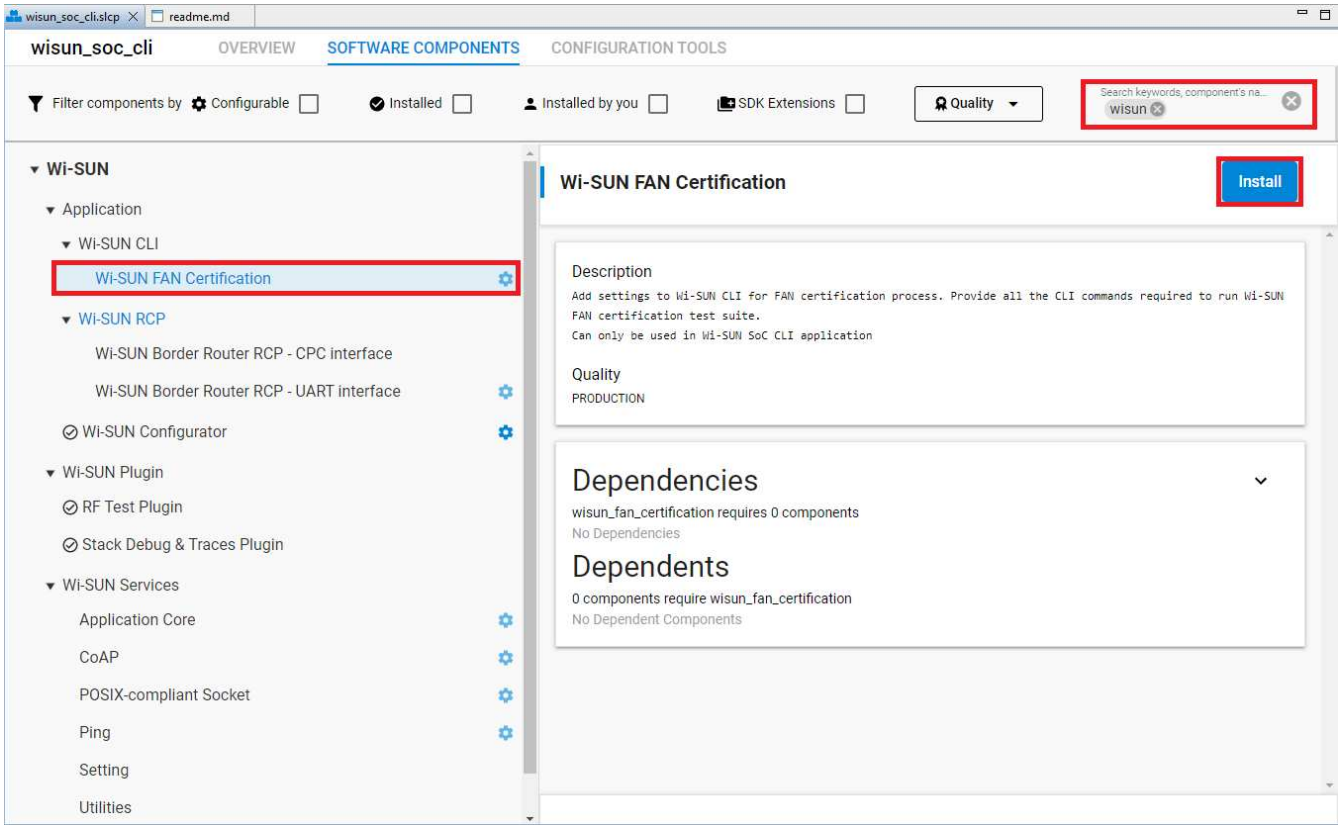
Follow these steps to add the **Wi-SUN FAN Certification** component to your application:

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
# Connect to the Wi-SUN network using FAN1.0 settings
> wisun join_fan10
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN Limited Function Nodes (LFN)

Wi-SUN Limited Function Nodes (LFN)

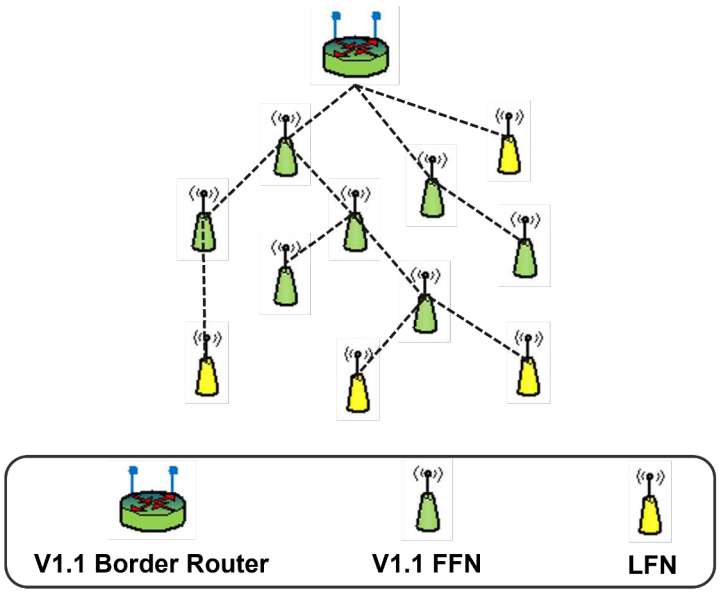
Introduction

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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).

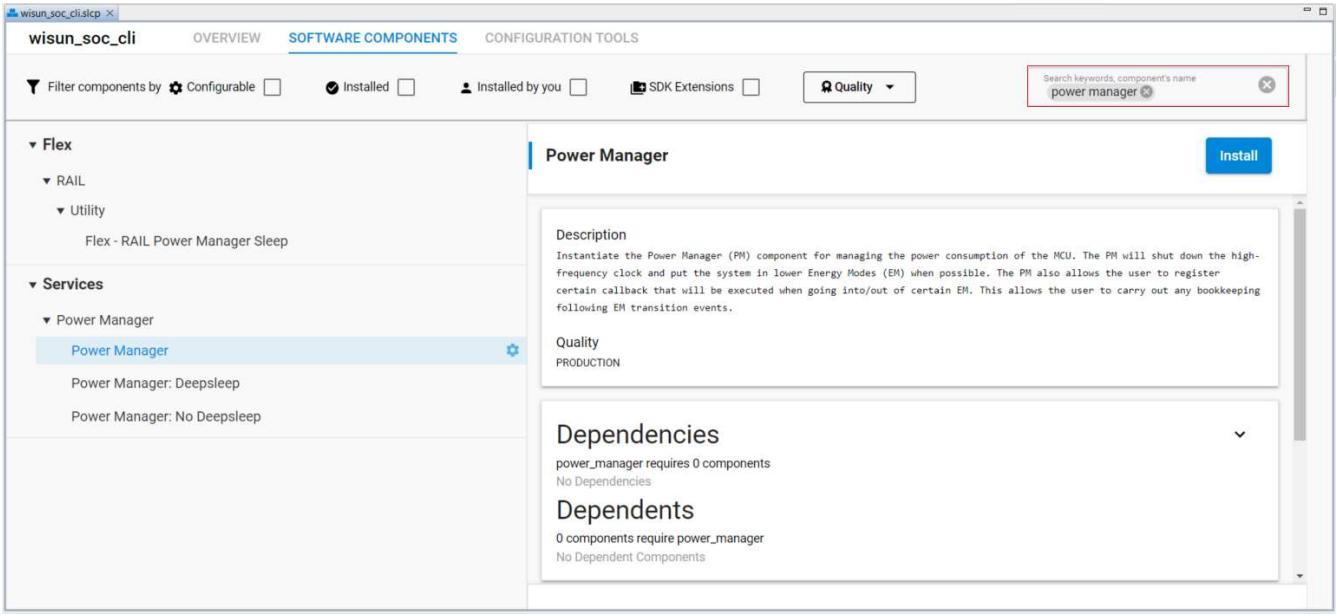
> Note: The EFR32FG28 version BRD4401B and the EFR32ZG28 version BRD4400B have hardware limitations that keep them from entering sleep mode EM2.

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.

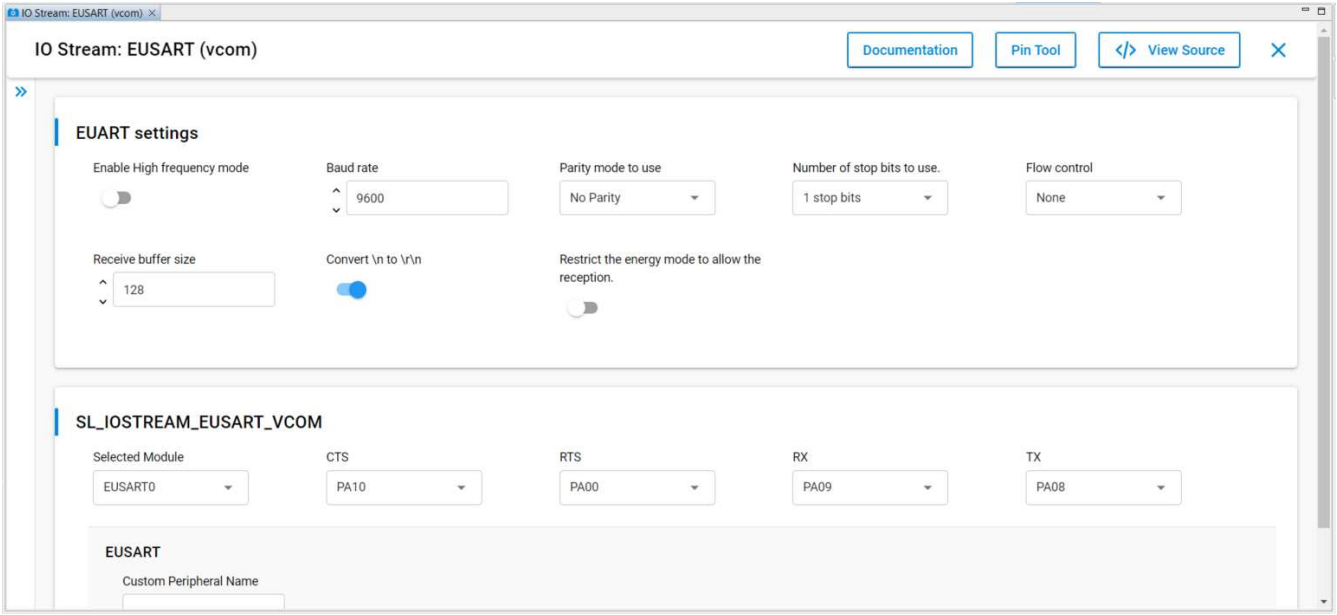
For the rest of the examples, this configuration has to be done manually on the software component perspective after creating the project. Follow the next steps to disable the UART high frequency clock and enable the power management capabilities:

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**.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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` .

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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
```

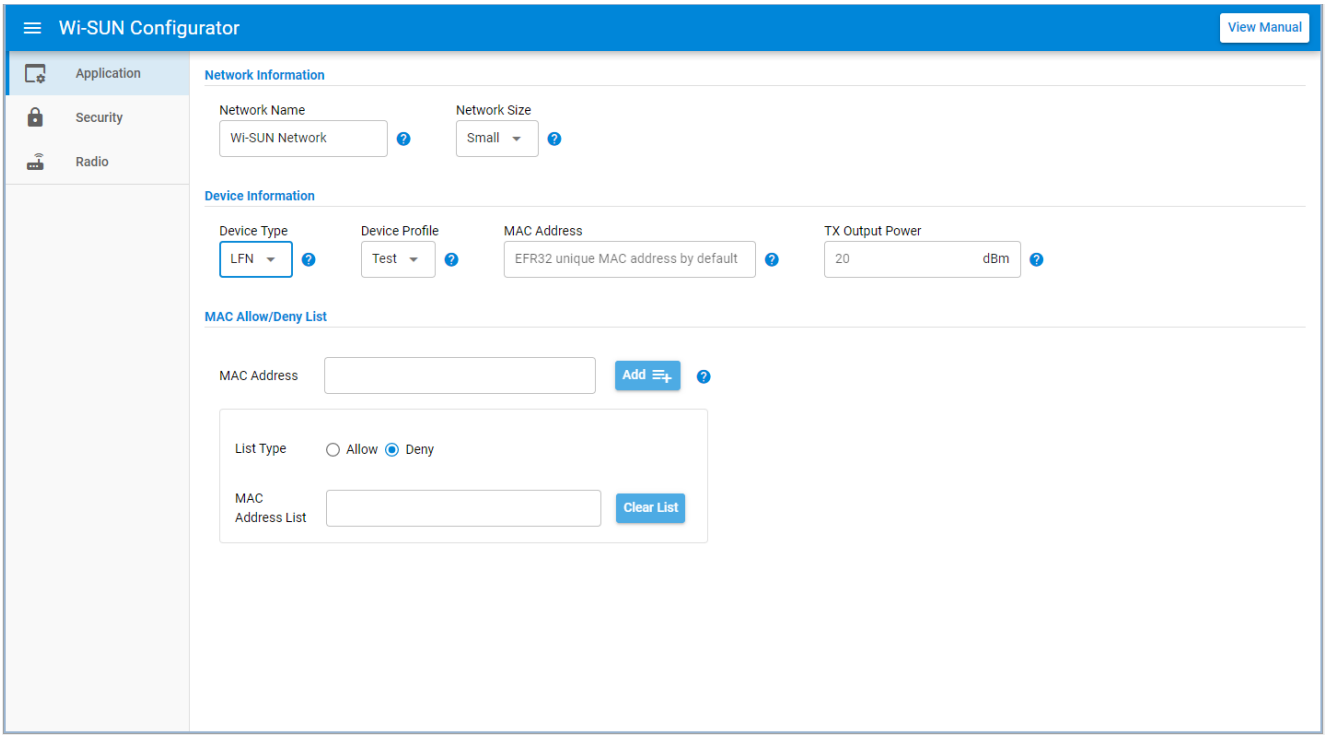
> Note: The profiles defined above focus on defining different time slots and intervals for the LFN parameters to optimize performance and battery life. These three predefined modes are just for reference, and it is up to the customer to fine tune the parameters to adapt the LFN behavior to the application 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**.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

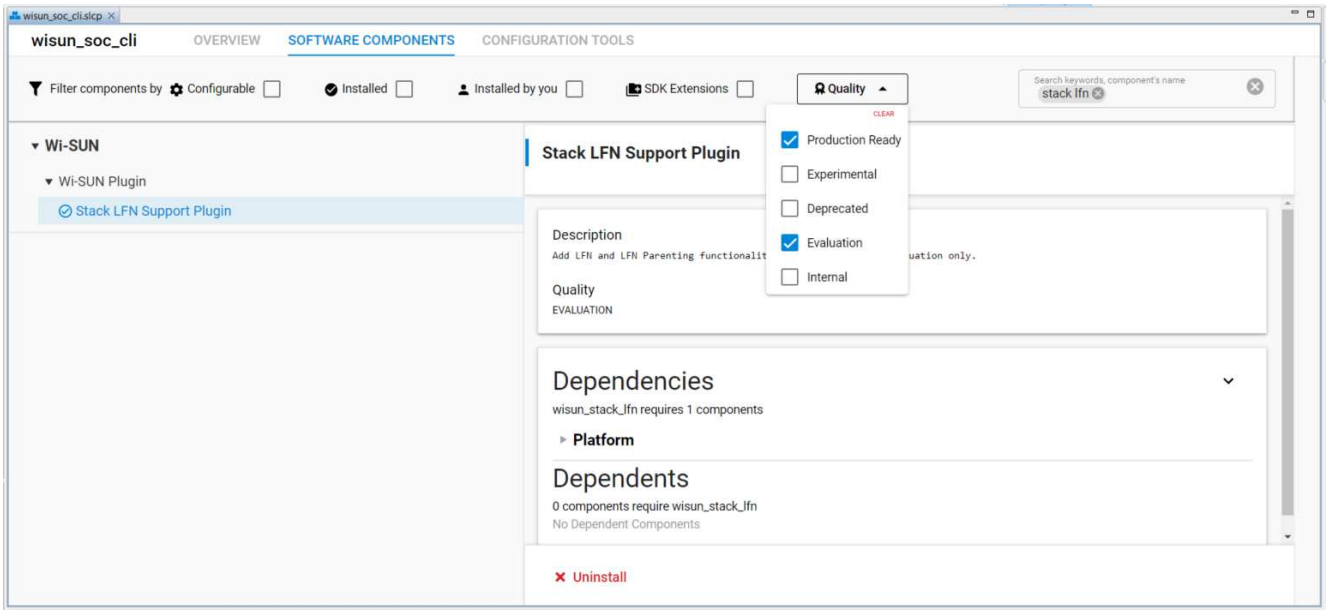
- Wi-SUN – SoC CoAP Collector
- Wi-SUN – SoC Collector
- Wi-SUN – SoC TCP Client

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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: Introduces bootloading for Silicon Labs networking devices. Discusses the Gecko Bootloader and bootloader file formats.
- Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher: Describes the high-level implementation of the Silicon Labs Gecko Bootloader for Series 3 microcontrollers, SoCs (System on Chips), and NCPs (Network Co-Processors), and provides information on different aspects of configuring the Gecko Bootloader.
- Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices): 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Bootloader Fundamentals

Bootloader Fundamentals

NOTE: This section replaces *UG103.06: Bootloader Fundamentals*. Further updates to this user guide will be provided here.

This guide introduces bootloading for Silicon Labs networking devices. It describes the concepts of standalone and application bootloaders and discusses their relative strengths and weaknesses. In addition, it looks at design and implementation details for each method. Finally, it describes the bootloader file format.

Silicon Labs' *Fundamentals* series covers topics that project managers, application designers, and developers should understand before beginning to work on an embedded networking solution using Silicon Labs chips, networking stacks such as EmberZNet PRO or Silicon Labs Bluetooth, and associated development tools. These guides can be used as a starting place for anyone needing an introduction to developing wireless networking applications, or who is new to the Silicon Labs development environment.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Overview

Introduction

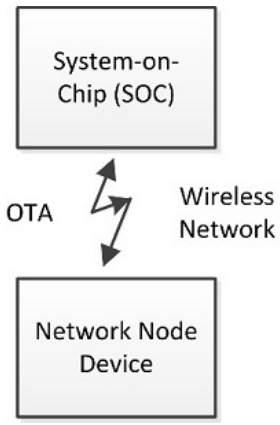
The bootloader is a program stored in reserved flash memory that can initialize a device, update firmware images, and possibly perform some integrity checks. Firmware image update occurs on demand, either by serial communication or over the air. Production-level programming is typically done during the product manufacturing process yet it is desirable to be able to reprogram the system after production is complete. More importantly, it is valuable to be able to update the device's firmware with new features and bug fixes after deployment. The firmware image update capability makes that possible.

Silicon Labs supports devices that do not use a bootloader, but this requires external hardware such as a Debug Adapter (Silicon Labs ISA3 or Wireless Starter Kit (WSTK)) or third-party SerialWire/JTAG programming device to update the firmware. Devices without a bootloader have no supported way of updating the firmware once they are deployed, which is why Silicon Labs strongly advocates implementing a bootloader.

In March of 2017, Silicon Labs introduced the Gecko Bootloader, 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 can be used with EFM32 and EFR32 Series 1 and later devices. The Gecko Bootloader was restructured into a component-based design and released with Gecko SDK Suite (GSDK) 4.0 in December of 2021. This new version is documented in Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) or Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher, along with other documents. Documentation for older versions is installed with their respective SDKs.

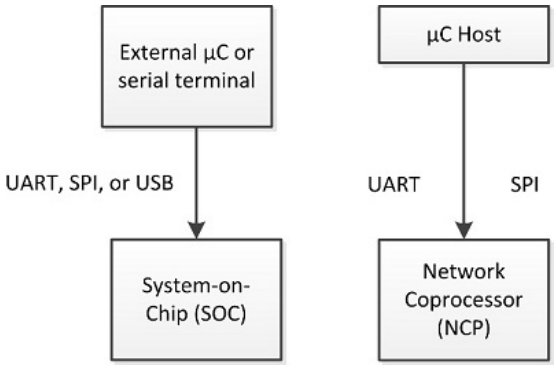
The Gecko Bootloader uses a customized update image file format. The update image file consumed by a Gecko Bootloader-generated application bootloader is a GBL (Gecko BootLoader) file, and is described in Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) or Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher.

Bootloading a firmware update image can be accomplished in two ways. The first is Over-The-Air (OTA), that is, through the wireless network, as shown in the following figure.



The second is through a hardwired link to the device. The following figure represents the serial bootloader use cases for SoCs (System on Chips) using either a UART (Universal Asynchronous Receiver/Transmitter), SPI (Serial Protocol Interface), or USB (Universal Serial Bus) interface, and for NCPs (Network Coprocessors) using either UART or SPI.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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). Application bootloaders are further divided into those that use external storage for the download update image, and those that use local storage. These bootloader types are discussed in the next two sections.

The firmware update situations described in this document assume that the source node (the device sending the firmware image to the target through a serial or OTA link) acquires the new firmware through some other means. For example, if a device on the local Zigbee network has an Ethernet gateway attached, this device could get or receive these firmware updates over the Internet. This necessary part of the firmware update process is system-dependent and beyond the scope of this document.

Standalone Bootloader

A standalone bootloader is a program that uses an external communication interface, such as UART or SPI, to get an application image. Standalone firmware update is a single-stage process that allows the application image to be placed into flash memory, overwriting the existing application image, without the participation of the application itself. Very little interaction occurs between the standalone bootloader and the application running in flash. In general, the only time that the application interacts with the bootloader is when it requests a reboot into the bootloader. Once the bootloader is running, it receives firmware update packets containing the (new) firmware image either by physical connections such as UART or SPI, or by the radio (over-the-air).

When the firmware update process is initiated, the new code overwrites the existing stack and application code. If any errors occur during this process, the application cannot be recovered, and the process must start over. For information about configuring the Gecko Bootloader as a standalone bootloader, see Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) or Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher.

Application Bootloader

An application bootloader begins the firmware update process after the running application has completely downloaded the update image file. The application bootloader expects that the image either lives in external memory accessible by the bootloader or in a portion of main flash memory (if the chip has sufficient memory to support this local storage model).

The application bootloader relies on the application to acquire the new firmware image. This image can be downloaded by the application in any way that is convenient (UART, over-the-air, etc.) but it must be stored into a region referred to as the download space. The download space is typically an external memory device such as an EEPROM or dataflash, but it can also be a section of the chip's internal flash when using a local storage variant of the application bootloader. Once the new image has been stored, the application bootloader is then called to validate the new image and copy it from the download space to flash.

Since the application bootloader does not participate in acquiring the image, and the entire image is downloaded before the firmware update process is started, download errors do not adversely affect the running image. The download process can be restarted or paused to acquire the image over time. The integrity of the downloaded update image can be verified before initiating the firmware update process, to prevent a corrupt or non-functional image from being applied.

The Gecko Bootloader can be configured to accept a list of multiple upgrade images to attempt to verify and apply. This allows the Gecko Bootloader to store what is in effect a backup copy of the update image, which it can access if the first

Copyright © 2025 Silicon Laboratories. All rights reserved.

image is corrupt.

Note that the EmberZNet NCP platform does not utilize an application bootloader because the application code resides on the host rather than on the NCP directly. Instead a device acting as a serial coprocessor would utilize a standalone bootloader designed to accept code over the same serial interface as the expected NCP firmware uses. However, the host application (residing on a separate MCU from the NCP) can utilize whatever bootloading scheme is appropriate.

For more information on application bootloaders, see Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) or Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher.

Copyright © 2025 Silicon Laboratories. All rights reserved.

About the Gecko Bootloader

About the Gecko Bootloader

The Silicon Labs Gecko Bootloader is a configurable code library that can be used with all the newer Silicon Labs Gecko MCUs and wireless MCUs. It uses a specially-formatted update image file called a GBL file. The Gecko Bootloader has a two-stage design on Series 1 devices, where a minimal first stage bootloader is used to update the main bootloader. On Series 2 devices, the first stage bootloader is replaced by a Secure Engine and the Gecko Bootloader consists only of the main bootloader. The Secure Engine may be hardware-based, or virtual (software). If hardware-based, the implementation may be either with or without Secure Vault. Throughout this document, the following conventions will be used.

- **HSE**: Hardware Secure Engine, either with or without Secure Vault if not specified
- **VSE**: Virtual Secure Engine
- **SE**: Secure Engine (either HSE or VSE, in general)

Having a first stage bootloader or SE allows for field updates of the main bootloader, including adding new capabilities, changing communication protocols, adding new security features and fixes, and so on. The Gecko Bootloader consists of three component parts:

Core: The bootloader core contains the main function of both bootloader stages. It also contains functionality to write to the internal main flash, to perform a bootloader update, and to reset into the application flagging applicable reset reasons.

Driver: Different bootloading applications require different hardware drivers for use by the other components of the bootloader.

Component/Plugin: All parts of the main bootloader that are either optional or selectable for different configurations are implemented as components (in GSDK 4.0 and higher) or previously in plugins. Each component/plugin has a generic header file, and one or more implementations. The current release contains components for functionality like UART and SPI communication protocols, SPI flash storage, internal flash storage, and different cryptographic operations.

Features

Gecko Bootloader features include:

- Field-updatable
- Secure boot
- Signed GBL firmware update image file
- Encrypted GBL firmware update image file

These features are summarized in the following sections and described in more detail in Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) or Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher. Protocol-specific information about using the Gecko Bootloader may be found in the following documents:

- Using the Gecko Bootloader with EmberZNet
- *UG235.06: Bootloading and OTA with Silicon Labs Connect SDK v2.x*
- *UG435.06: Bootloading and OTA with Silicon Labs Connect SDK v3.x*
- Using the Gecko Bootloader with Silicon Labs Bluetooth Applications

Field-Updatable

Series 1

On EFM32 and EFR32 Series 1 devices, field update capability for the Gecko bootloader is provided by a two-stage design where the bootloader has a first stage and a main stage. The minimal first stage of the bootloader is not field updatable, and can only update the main bootloader by reading from and writing to fixed addresses in internal flash memory. To perform a main bootloader update, the running main bootloader verifies the integrity and authenticity of the bootloader update image, writes it to internal flash, and issues a reboot into the first stage bootloader. The first stage bootloader

Copyright © 2025 Silicon Laboratories. All rights reserved.

verifies the integrity of the main bootloader update image before copying it to the main bootloader location, completing the update

Series 2

On Series 2 devices, field update capability for the Gecko bootloader is provided by the SE. To perform a main bootloader update, the running main bootloader verifies the integrity and authenticity of the bootloader update image, writes it to internal flash, and requests that the SE installs the update. The SE optionally verifies the authenticity of the main bootloader update image before copying it to the main bootloader location, completing the update. The same mechanism can be used to update the SE itself.

Secure Boot

Secure boot is designed to prevent an untrusted application from running on the device. When Secure Boot is enabled, the bootloader enforces cryptographic signature verification of the application image on every boot using asymmetric cryptography. The signature algorithm used is ECDSA-P256-SHA256. The public key is written to the device during manufacturing, while the private key is kept secret. This ensures that the application was created and signed by a trusted party.

Signed GBL Update Image File

The Gecko Bootloader supports enforcing cryptographic signature verification of the update image file in addition to Secure Boot. This allows the bootloader and application to verify that the application or bootloader update comes from a trusted source before starting the update process. The signature algorithm used is ECDSA-P256-SHA256. The public key is the same key as for secure boot, written to the device during manufacturing, while the private key is never distributed. This ensures that the GBL file was created and signed by a trusted party.

Encrypted GBL Update File

The GBL update file can also be encrypted, to prevent eavesdroppers from getting hold of the plaintext firmware image. The encryption algorithm used is AES-CTR-128, and the encryption key is written to the device during manufacturing.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Memory Space for Bootloading

Memory Space for Bootloading

The first stage of the Gecko Bootloader on Series 1 devices takes up a single flash page. On devices with 2 kB flash pages, like EFR32MG1, this means that the first stage takes 2 kB.

The size of the main bootloader is dependent on the functionality required. With a typical bootloader configuration, the main bootloader for Series 1 devices takes up 14 kB of flash, bringing the total bootloader size to 16 kB.

Silicon Labs recommends reserving 16 kB for the bootloader for Series 1 and EFR32xG21 devices and 24 kB for EFR32xG22 devices.

On EFR32xG1 devices (Mighty Gecko, Flex Gecko, and Blue Gecko families), the bootloader resides in main flash.

- First stage bootloader @ 0x0
- Main bootloader @ 0x800
- Application @ 0x4000

On EFR32xG12 and later Series 1 devices, the bootloader resides in the bootloader area in the Information Block.

- Application @ 0x0
- First stage bootloader @ 0x0FE10000
- Main bootloader @ 0x0FE10800

On EFR32xG21, the main bootloader resides in main flash:

- Main bootloader @ 0x0
- Application @ 0x4000

On EFR32xG22, the main bootloader resides in main flash:

- Main bootloader @ 0x0
- Application @ 0x6000

On EFR32xG23, the main bootloader resides in main flash:

- Main bootloader @ 0x08000000
- Application @ 0x08006000

On EFR32xG24, the main bootloader resides in main flash:

- Main bootloader @ 0x08000000
- Application @ 0x08006000

Copyright © 2025 Silicon Laboratories. All rights reserved.

Design Decisions

Design Decisions

The decision of what bootloader type to deploy depends on many factors. Note that the platform type and available flash memory may limit bootloader choices.

Some questions related to this are:

- Where does the device get the new update image? Is this over-the-air via the networking protocol? Using a separate interface connected to the Internet?
- Will the device have an external memory chip to store a new update image? If not, is there enough internal flash memory to store both a current and a newly downloaded copy of the largest expected application image?
- If the device receives the new image over-the-air, will it be multiple hops away from the server holding the download image?
- What kind of image security is needed?
- What communications driver will be used (in the single protocol case)?
- Does the use case require more than one protocol?

The configurable design of the Gecko Bootloader platform means that developers can create bootloaders to fit almost any design choice. See Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices) or Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader User's Guide for Series 3 and Higher

Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher

This guide describes the high-level implementation of the Silicon Labs Gecko Bootloader for Series 3 microcontrollers, SoCs (System on Chips), and NCPs (Network Co-Processors), and provides information on different aspects of configuring the Gecko Bootloader.

The Silicon Labs Gecko Bootloader is a common bootloader for all the newer MCUs and wireless MCUs from Silicon Labs. This guide is specific to Series 3 devices and above. The Gecko Bootloader can be configured to perform a variety of functions, from device initialization to firmware upgrades. Key features of the bootloader are:

- Usable across Silicon Labs Gecko microcontroller and wireless microcontroller families
- In-field upgradeable
- Configurable
- Enhanced security features, including:
- **Secure Boot**: When Secure Boot is enabled, the bootloader enforces cryptographic signature verification of the application image on every boot, using asymmetric cryptography. This ensures that the application was created and signed by a trusted party.
- **Signed upgrade image file**: The Gecko Bootloader supports enforcing cryptographic signature verification of the upgrade image file. This allows the bootloader and application to verify that the application or bootloader upgrade comes from a trusted source before starting the upgrade process, ensuring that the image file was created and signed by a trusted party.
- **Encrypted upgrade image**: The image file can also be encrypted to prevent eavesdroppers from acquiring the plaintext firmware image.

For Series 3 devices and above, Gecko Bootloader uses a proprietary format for its upgrade images, called GBL V4 (Gecko Bootloader file). These files have the file extension ".gbl". See the *Gecko Bootloader File Format* page for more details. Series 3 and above devices do not support GBL versions lesser than v4.

On Series 3 devices, the Gecko bootloader consist of the main stage bootloader. The main bootloader is upgradable through the Secure Engine.

The Secure Engine provides functionality to install an image to code region 0 in the external flash.

To perform a bootloader upgrade, the running bootloader verifies the integrity and authenticity of the bootloader upgrade image file. The bootloader then writes the upgrade image to the upgrade location in external flash and requests that the Secure Engine installs it. On some devices, the Secure Engine is also capable of verifying the authenticity of the main bootloader update image against a root of trust. The Secure Engine itself is upgradable using the same mechanism. See the *Gecko Bootloader Operation - Secure Engine Upgrade* page for more details.

The bootloader consists of a common core, drivers, and a set of components that give the bootloader specific capabilities. The common bootloader core is provided as a full-source delivery. The common bootloader core contains functionality to parse GBL v4 files and flash their contents to the device.

The Gecko Bootloader can be configured to perform firmware upgrades in standalone mode (also called a standalone bootloader) or in application mode (also called an application bootloader), depending on the component configuration. Components can be installed in and configured through the Simplicity Studio IDE.

A standalone bootloader uses a communications channel to get a firmware upgrade image. NCP (network co-processor) devices always use standalone bootloaders. Standalone bootloaders perform firmware image upgrades in a single-stage process that allows the application image to be placed into flash memory, overwriting the existing application image, without the participation of the application itself. In general, the only time that the application interacts with a standalone bootloader is when it requests to reboot into the bootloader. Once the bootloader is running, it receives packets containing

Copyright © 2025 Silicon Laboratories. All rights reserved.

the firmware upgrade image by a physical connection such as UART or SPI. To function as a standalone bootloader with a physical connection, a component providing a communication interface such as UART or SPI must be configured.

An application bootloader relies on the application to acquire the firmware upgrade image. The application bootloader performs a firmware image upgrade by writing the firmware upgrade image to a region of flash memory referred to as the download space. The application transfers the firmware upgrade image to the download space in any way that is convenient (UART, over-the-air, Ethernet, USB, and so on). The download space is data flash or a section of the device's flash. The Gecko Bootloader can partition the download space into multiple storage slots and store multiple firmware upgrade images simultaneously. To function as an application bootloader, a component providing a bootloader storage implementation must be configured.

Silicon Labs provides example bootloaders that come with a preconfigured set of installed components for configuration in either standalone or application mode. See the *Configuring the Gecko Bootloader* page. The following sections provide an overview of the Gecko Bootloader common core, drivers, and components. For details, including details on error codes and conditions, see the Gecko Bootloader API Reference, shipped with the SDK in the platform/bootloader/documentation folder.

Core

The bootloader core contains the bootloader's main functions. It also contains functionality to write to the external flash, an image parser to parse and act upon the contents of GBL v4 upgrade files, and functionality to boot the application in main flash.

A version of the GBL v4 image parser without support for encrypted upgrade images is also available. This version can be used in flash space constrained bootloader applications where encryption of the upgrade image is not required.

Shared Memory

To exchange information between the bootloader and application, a section of SRAM is used. The contents of SRAM are preserved through a software reset, making SRAM suitable as a communication channel between bootloader and application.

The shared memory has a size of 4 bytes, and is located at the first address of SRAM, 0x20000000. It is used to store a single word containing the reason for a reset. The structure of the reset cause word is defined in the Reset Information part of the Application Interface, in the file **btl_reset_info.h**, as 16 bits containing the reason, and 16 bits of signature indicating if the word is valid or not. If the signature reads 0xF00F, the reset reason is valid.

All 16-bit reset reasons used by Silicon Labs have the most significant bit set to zero. If custom reset reasons are desired, it is recommended to set the most significant bit to avoid conflicting definitions.

Drivers

Different applications for firmware upgrade require different hardware drivers for use by the other components of the bootloader.

Driver modules include:

- Delay: Simple delay routines for use with components that require small delays or timeouts.
- SPI Slave: Flexible SPI Slave driver implementation for use in communication components implementing SPI protocols. This driver supports both blocking and non-blocking operation, with DMA (Direct Memory Access) backing the background transfers to support non- blocking operation.
- UART: Flexible serial UART driver implementation for use in communication components implementing UART protocols. This driver supports both blocking and non-blocking operations, with DMA backing the background transfers to support non-blocking operation. Additionally, support for hardware flow control (RTS/CTS) is included.

Components

Copyright © 2025 Silicon Laboratories. All rights reserved.

All parts of the bootloader that are either optional or that may be exchanged for different configurations are implemented as components. Each component may have a configuration header file, and one or more implementations. Components include:

- Communication
- UART: XMODEM
- UART: BGAPI
- Compression
- Debug
- GPIO Activation
- Security
- Storage

Communication

Communication components provide an interface for implementing communication with a host device, such as a computer or a micro- controller. Several components implement the communication interface, using different transports and protocols.

- BGAPI UART DFU: By enabling the BGAPI communication component, the bootloader communication interface implements the UART DFU protocol using BGAPI commands.
- UART XMODEM: By enabling the UART XMODEM communication component, the bootloader communication interface implements the XMODEM-CRC protocol over UART. This component makes the bootloader compatible with the legacy serial-uart-bootloader that was previously released with the EmberZNet wireless stack. See *AN760: Using the Ember Standalone Bootloader* for more information about legacy Ember standalone bootloaders.

Compression

Compression components provide capability for the bootloader GBL file parser to handle compressed GBL v4 upgrade images. Each compression component provides support for one (de)compression algorithm. At the time of writing, decompression of data compressed with the LZ4 and LZMA algorithms is supported, through the *GBL Compression (LZ4)* and *GBL Compression (LZMA)* components.

Debug

This component provides the bootloader with support for debugging output. If the component is configured to enable debug prints, short debug messages will be printed over Serial Wire Output (SWO), which can be accessed in multiple ways, including using Simplicity Commander, and by connecting to port 4900 of the Wireless Starter Kit TCP/IP interface.

To turn on debug prints, enable the Debug component and select **Debug prints**. Select **Debug asserts** to enable assertions in the source code.

GPIO Activation

This component provides functionality to enter firmware upgrade mode automatically after reset if a GPIO pin is active during boot. The GPIO pin location and polarity are configurable.

- GPIO: By enabling the GPIO activation component, the firmware upgrade mode can be activated by push buttons.

Security

Security components provide implementations of cryptographic operations as well as functionality to compute checksums and to read cryptographic keys from manufacturing tokens.

Modules include:

- AES: AES decryption functionality
- CRC16: CRC16 functionality
- CRC32: CRC32 functionality
- ECDSA: ECDSA signature verification functionality
- SHA-256: SHA-256 digest functionality

Storage

Copyright © 2025 Silicon Laboratories. All rights reserved.

These components provide the bootloader with multiple storage options for SoCs. All storage implementations must provide an API to access image files to be upgraded. This API is based on the concept of dividing the download space into storage slots, where each slot has a predefined size and location in memory and can be used to store a single upgrade image. Some storage implementations also support a raw storage API to access the underlying storage medium. This can be used by applications to store other data in parts of the storage medium that are not used for storing firmware upgrade images. Implementations include:

- **External Flash**: The external flash storage implementation uses the external flash of the device for upgrade image storage. Note that this storage area is only a download space and is separate from the portion of external flash used to hold the active application code.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader File Format v4

Gecko Bootloader File Format v4

The Gecko Bootloader supports GBL File format v4 for Series 3 device. This format has significant differences as compared to version 3, The file formats described in this section are generated by Simplicity Commander commands.

File Structures

The GBL v4 file format is composed of nested TLV (Type-Length-Value) elements. The format of the TLV is as follows:

Field	Length (Bytes)	Description
Type	4	Type indicates the content of the Value field
Length	4	Length of the Value field in bytes
Value	N	Length size field containing either more TLVs or data

There are two types of TLVs, data and container. Data TLVs value field contains data whose format is specified in the type field and the container TLVs value filed contains more TLVs, which in turn again may contain data or container TLVs.

TLV Descriptions

This section describes each of the TLVs.

GBLV4

This mandatory TLV is used to identify the GBLv4 file when parsing and tell the length of the file.

Type	Length	Value
0x84A617EB	Variable	Contains all the TLVs for the file

MANIFEST

MANIFEST TLV is present inside the GBLV4 TLV. It has a collection, TLVs, that describes the metadata of the GBLv4 file that is used during the update process.

Type	Length	Value
0xAA01012A	Variable	Contains metadata TLVs for applying the file.

MANIFEST_INFO

The MANIFEST_INFO TVL contains the `ManifestInfo_t` structure that describes the version dependency and the features of the GBL file.

Type	Length	Value
0x2B03032B	Variable	manifest_info structure

The `ManifestInfo_t` structure is as follows:

Copyright © 2025 Silicon Laboratories. All rights reserved.

Data Type	Field	Purpose
uint32_t	version	Minimum version of the GLBv4 parser required, used for future extensions.
uint32_t	features	Features requested from the parser (encryption, compression etc.)

BUNDLE_VERSION

This is the child TVL of MANIFEST TLV, required to identify the bundle/file and is used for versioning and rejection of downgrades.

Type	Length	Value
0x2B04042B	Fixed	bundle_version structure

The `BundleVersion_t` structure is as follows:

Data Type	Field	Purpose
Uint8_t	productId	128 bit unique product ID
uint32_t	bundleVersion	Reject if smaller than this exists
Uint32_t	minVersion	The minimum previous version to which this update file can be applied

CONTENT_HASH

An optional TLV that provides integrity checking on the remainder of the file. This TLV is applicable to batch updates only.

Type	Length	Value
0x0x2B05052BUL	Fixed	content_hash structure

The `content_hash` structure is as follows:

Data Type	Field	Purpose
uint32_t	hashType	The hash function used to validate (1=SHA, etc.)
uint8_t	sha[]	The hash of all contents following the manifest

MANIFEST_CERTIFICATE

Type	Length	Value
0x2B01012B	Fixed	ApplicationCertificate_t

The `ApplicationCertificate_t` structure is as follows:

Data Type	Field	Purpose
uint8_t	structVersion	Vesion of the certificate structure
uint8_t	flags	Reserved flags
uint8_t	key	Public key used to verify GBL file
uint32_t	version	Version number of this certificate
uint8_t	signature	The signature of the certificate itself (not the GBL file)

MANIFEST_SIGNATURE

Type	Length	Value
0x2B02022BUL	Variable	Manifest signature structure

Copyright © 2025 Silicon Laboratories. All rights reserved.

The `Manifest_signature` structure is as follows:

Data Type	Field	Purpose
uint32_t	Signature function	The function used for signing
uint8_t[]	Signature	The signature for authentication

UPDATE_PROCESS

This is a collection of TLVs that describes the update process. Parsing this TLV does not happen until the MANIFEST_TLV is authenticated. Each TLV contained in this is parsed and processed one at a time in the order they are listed.

Type	Length	Value
0xAB06062BUL	Variable	TLVs describing the steps to update the system

UPDATE_SE

This is an optional TLV that is used to update the SE.

Type	Length	Value
0x2C02022CUL	Fixed	UpdateSe_t structure

The `UpdateSe_t` structure is as follows:

Data Type	Field	Purpose
uint32_t	version	Version number of the SE update image.
uint32_t	tlvPosition	The absolute position of the SE update image Blob TLV

UPDATE_MEMORY_SECTION

The memory section's content is present in the file and needs to be updated.

Type	Length	Value
0x2C03032CUL	Fixed	UpdateMemorySection_t structure

The `UpdateMemorySection_t` structure is as follows:

Data Type	Field	Purpose
uint8_t	targetMemory	Which memory this definition is for (0=main memory, 0x1-0x7f = reserved, 0x80-0xff=defined by the user)
uint8_t	plainImageSize	Represents the size of the plain image in 3 bytes
uint32_t	targetAddr	The target memory address for the update
uint32_t	type	Bitfield representing the type of application
uint32_t	version	Version number for this application (customer-defined)
uint32_t	capabilities	Bitfield representing the capabilities of this application
uint32_t	memorySectionPos	The absolute position of the MEMORY_SECTION_TLV in the file
HashVaule_t	memSecHash	A hash value for verifying the integrity of the memory section

MANIFEST_FINISH

TLV to indicate the end of an update.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Length	Value
0x2C04042CUL	Fixed	Empty

MEMORY_SECTION

This TLV contains TLVs for a memory section update.

Type	Length	Value
0xBA01013AUL	Variable	TLVs for the region

MEMORY_SECTION_INFO

This TLV contains information about how the blob data is read and where the data is to be written to. This is a common wrapper for all data as it allows the data to be compressed, verified and processed in a unified manner.

The data can be divided into blocks, with each block having its own hash. This can then be compared with the hashes in the memory section info to authenticate the block. This is useful during streaming updates to avoid writing unauthenticated data to flash.

Type	Length	Value
0x3B01013BUL	Fixed	MemSectionPartialInfo_t

The `MemSectionPartialInfo_t` structure is as follows:

Data Type	Field	Purpose
uint8_t	compressionScheme	0 = No compression 1 = LZ4 compression 2 = LMZA compression
uint8_t	encryptionScheme	0 = No encryption 1 = AES CCM
uint8_t	secureBootScheme	Reserved for future use.
uint8_t	reserved	
uint16_t	signBlockSize	Block size if DFU authentication is done for each block.
uint16_t	numBlocks	Number of blocks inside the BLOB.
uint8_t[]	nonce	Nonce for encryption
uint8_t[]	finalImageHash	Reserved for future use.
uint8_t[]	secureBootSign	Reserved for future use.

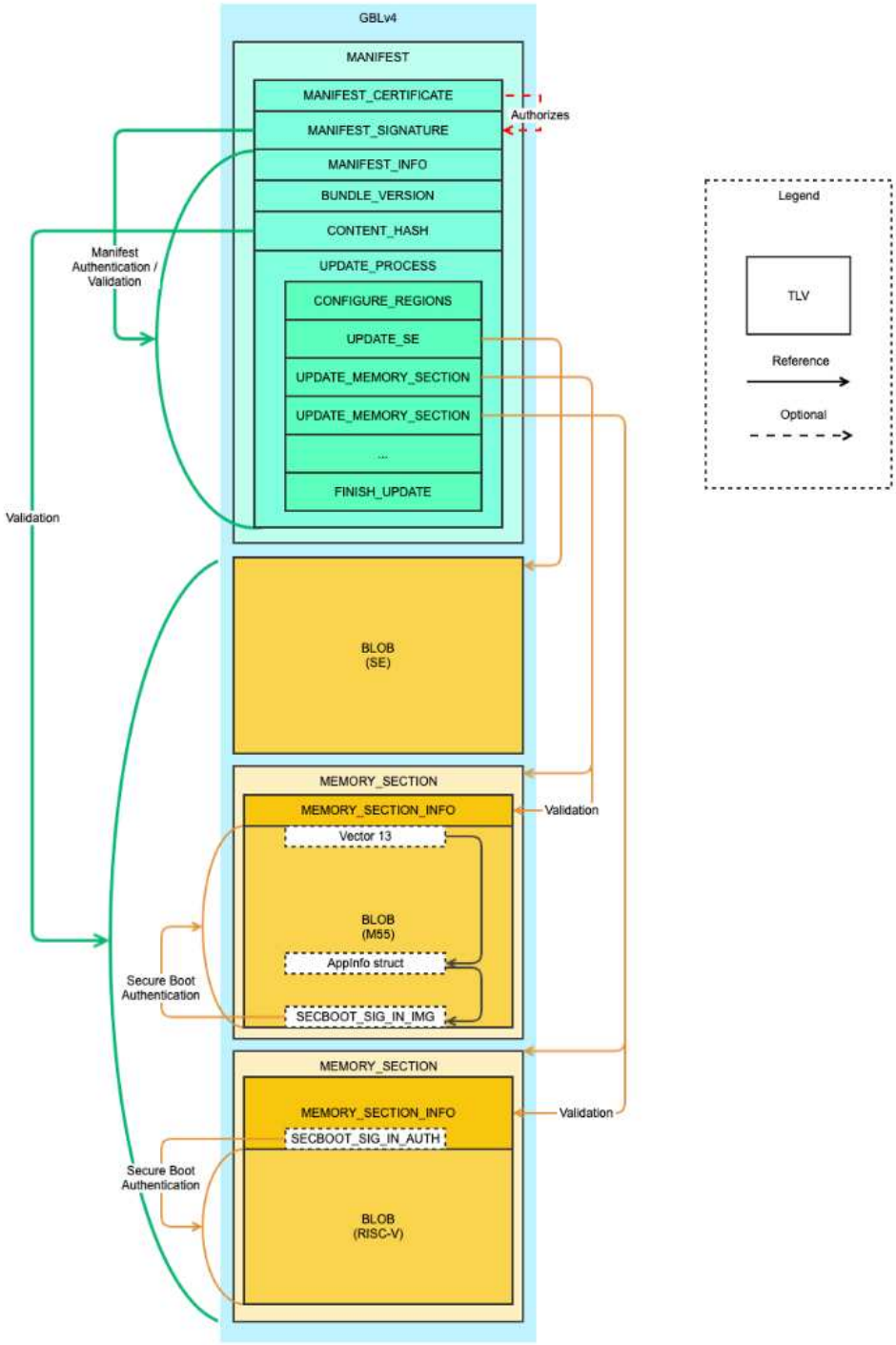
BLOB

This TLV contains variable length binary data.

Type	Length	Value
0x3B02023BUL	Variable	Binary Data

The figure below shows the structure of a GBL v4 file.

Copyright © 2025 Silicon Laboratories. All rights reserved.



Copyright © 2025 Silicon Laboratories. All rights reserved.

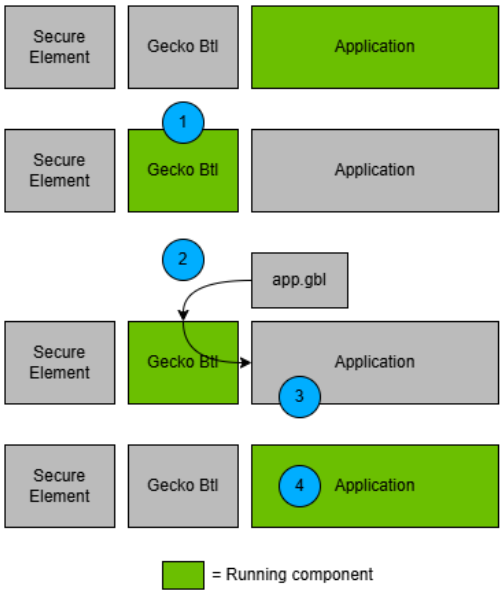
Gecko Bootloader Operation Application Upgrade

Gecko Bootloader Operation - Application Upgrade

This section summarizes Gecko Bootloader operation for updating application firmware, first if the Gecko Bootloader is configured in standalone mode and then if it is configured in application mode. The *Gecko Bootloader Operation - Bootloader Upgrade* provides the same information for updating the bootloader firmware.

Standalone Bootloader Operation

Standalone bootloader operation is illustrated in the following figure:



1. The device reboots into the bootloader.
2. A GBL file containing an application image is transmitted from the host to the device. If image encryption is enabled in the bootloader and the image is encrypted, decryption is performed during the process of receiving and parsing the GBL file.
3. The bootloader applies the application upgrade from the GBL upgrade file on-the-fly. If image authentication is enabled in the bootloader and the GBL file contains a signature, the authenticity of the image is verified before completing the process.
4. The device boots into the application. Application upgrade is complete.

Rebooting Into the Bootloader

The Gecko Bootloader supports multiple mechanisms for triggering the bootloader. If the **GPIO activation** component is installed, the host device can keep this pin low/high (depending on configuration) through reset to make the device enter the bootloader. The bootloader can also be entered through software. The `bootloader_rebootAndInstall` API first signals to the bootloader that it should enter firmware upgrade mode by writing a command to the shared memory location at the bottom of SRAM, and then performs a software reset. If the bootloader finds the correct command in shared memory upon boot, it will enter firmware upgrade mode instead of booting the existing application.

Downloading and Applying a GBL Upgrade File

When the bootloader enters firmware upgrade mode, it enters a receive loop waiting for data from the host device. The specifics of the receive loop depend on the protocol. Received packets are passed to the image parser, a state machine that parses the data and returns a callback containing any data that should be acted upon. The bootloader core implements this callback and flashes the data to external flash at the specified address. If GBL file authentication or encryption is

Copyright © 2025 Silicon Laboratories. All rights reserved.

enabled, the image parser will enforce this, and abort the image upgrade if the authentication fails The bootloader prevents a newly uploaded image from being bootable by holding back parts of the application vector table until the GBL file hash and GBL signature (if required) have been verified.

Booting Into the Application

When an application upgrade is completed, the bootloader triggers a reboot with a message in shared memory at the bottom of SRAM signaling that an application upgrade has been successfully completed. The application can use this reset information to learn that an application upgrade was just performed.

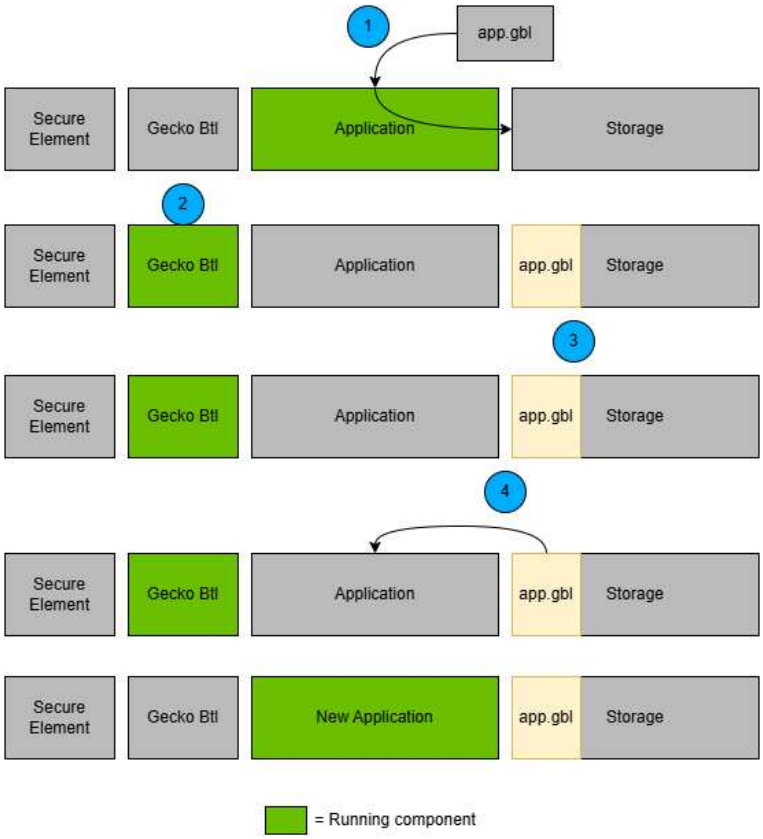
Before jumping to the main application, the bootloader verifies that the application is ready to run. This includes verifying if the Program counter and Stack Pointer are valid and also if the lock bit is set. If secure boot is enabled, the bootloader expects a signed application and attempts to validate the signature of the application. In scenarios where secure boot is not enabled, the bootloader attempts to validate if the Application properties pointer points to valid app properties structure in the flash. If valid app properties struct is found, the bootloader proceeds based on the signature type indicated by the application properties struct or else the bootloader assumes that the Application properties pointer points to the Reset Handler of the application (an app without application properties) and proceeds to boot into the application. In case the verification of the application fails at any stage, the bootloader enters the bootload mode instead of booting into the application.

Error Handling

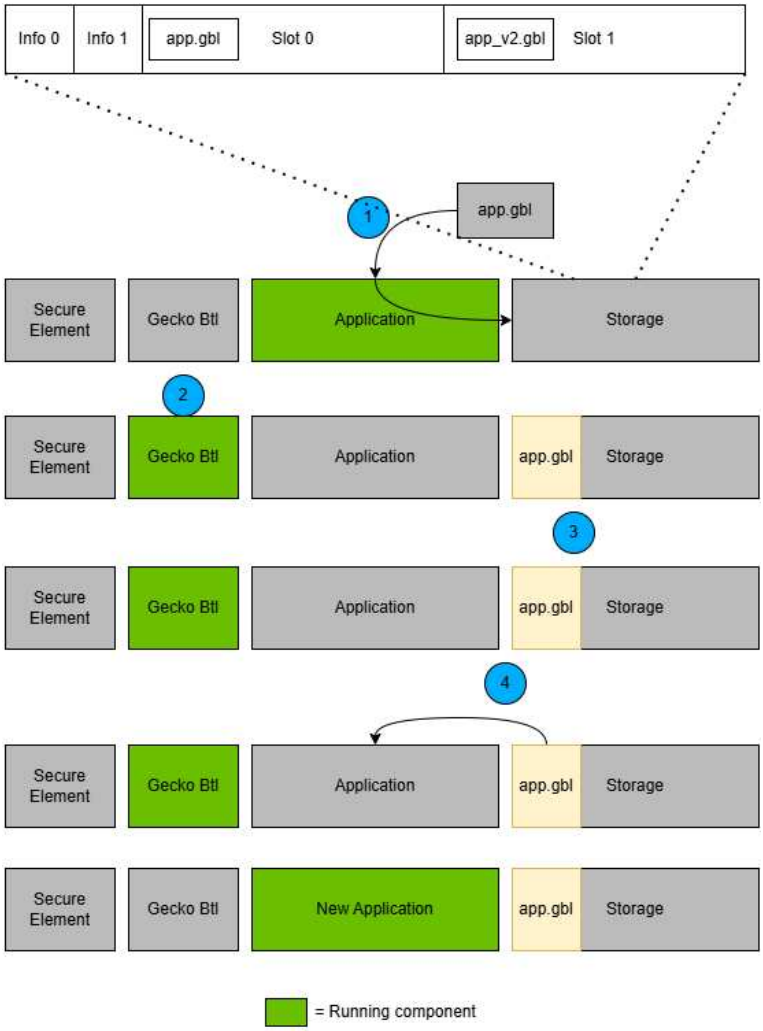
If the application upgrade is interrupted at any time, the device will be without a working application. The bootloader then resets the device, and re-enters firmware upgrade mode. The host device can easily restart the application upgrade process, to try loading the upgrade image again.

Application Bootloader Operation

The following figure illustrates the application bootloader operation both for a single image/single storage slot, and multiple images/ multiple storage slots.



Copyright © 2025 Silicon Laboratories. All rights reserved.



1. A GBL file is downloaded onto the storage medium of the device, as described below, and the presence of an upgrade image is indicated.
2. The device reboots into the bootloader, and the bootloader enters firmware upgrade mode.
3. The device applies the application upgrade.
4. The device boots into the application. Application upgrade is complete.

Downloading and Storing a GBL Upgrade Image File

To prepare for receiving an upgrade image, the application finds an available storage slot or erases an existing one using `bootloader_eraseStorageSlot`. If the bootloader only supports a single storage slot, a value of 0 should be used for the slot ID.

The application then receives a GBL file using an applicable protocol, such as Ethernet, USB, Zigbee, OpenThread or Bluetooth, and stores it in the slot by calling `bootloader_writeStorage`.

When the download is complete, the application can optionally verify the integrity of the GBL file by calling `bootloader_verifyImage`. This is also done by the bootloader before applying the image but can be done from the application to avoid rebooting into the bootloader if the received image was corrupt.

If multiple storage slots are supported, the application should write a bootload list by calling `bootloader_setImageToBootload`. The list is written to the two bootload info pages as shown in the figure above. The bootload list is a prioritized list of slots indicating the order the bootloader should use when attempting to perform a firmware upgrade. The bootloader attempts to verify the images in these storage slots in sequence and applies the first image to pass verification. If only a single storage slot is supported, the bootloader treats the entire download space as a single storage slot.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Rebooting and Applying a GBL Upgrade File

The bootloader can be entered through software. The `bootloader_rebootAndInstall` API signals to the bootloader that it should enter firmware upgrade mode by writing a command to the shared memory location at the bottom of SRAM, and then performs a software reset. If the bootloader finds the correct command in shared memory upon boot, it enters firmware upgrade mode instead of booting the existing application.

The bootloader iterates over the list of storage slots marked for bootload and attempts to verify the image stored in each. Once it finds a valid GBL upgrade file, firmware upgrade is attempted from this GBL file. If the upgrade fails, the bootloader moves to the next image in the list. If no images pass verification, the bootloader reboots back into the existing application with a message in the shared memory location in SRAM indicating that no good upgrade images were found.

Booting Into the Application

When an application upgrade is completed, the bootloader triggers a reboot with a message in shared memory at the bottom of SRAM signaling that an application upgrade has been successfully completed. The application can use this reset information to learn that an application upgrade was just performed.

Before jumping to the main application, the bootloader verifies that the application is ready to run. This includes verifying if the Program counter and Stack Pointer are valid as well as if the lock bit is set. If secure boot is enabled, the bootloader expects a signed application and attempts to validate the signature of the application. In scenarios where secure boot is not enabled, the bootloader attempts to validate if the Application properties pointer points to valid app properties structure in the flash. If valid app properties struct is found, the bootloader proceeds based on the signature type indicated by the application properties struct or else the bootloader assumes that the Application properties pointer points to the Reset Handler of the application (an app without application properties) and proceeds to boot into the application. In case the verification of the application fails at any stage, the bootloader enters the bootload mode instead of booting into the application.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader Operation Bootloader Upgrade

Gecko Bootloader Operation - Bootloader Upgrade

Bootloader upgrade functionality is provided by the Secure Engine on Series 3 devices. The Secure Engine itself is also upgradable. For more details, see the *Gecko Bootloader Operation - Secure Engine Upgrade* page.

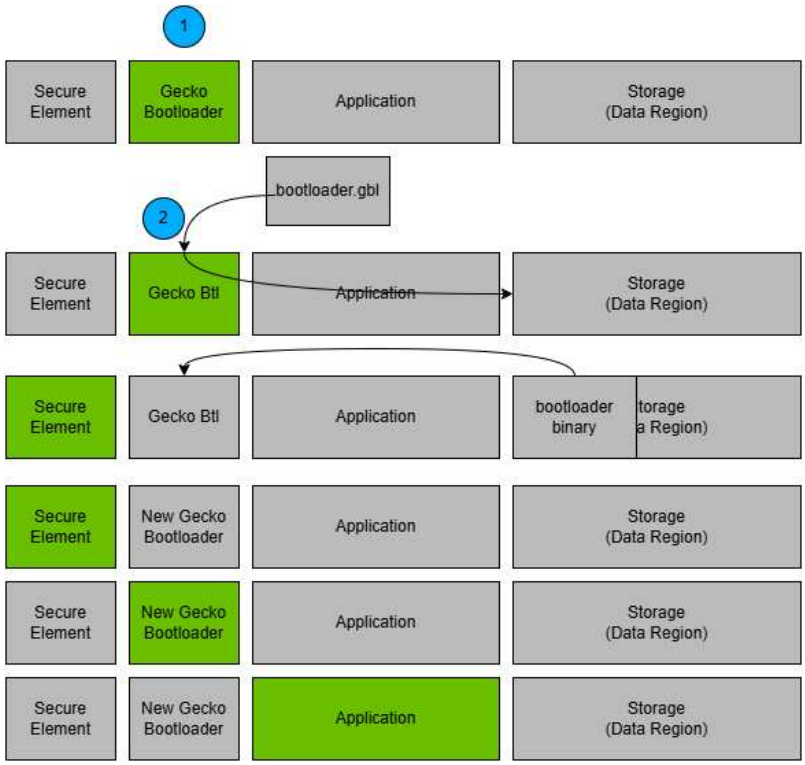
Requirements for upgrading the main bootloader vary depending on the bootloader configuration:

- Application bootloader with storage: Upgrading the main bootloader requires a single GBL file containing both bootloader and application upgrade images.
- Standalone bootloader with communication interface: Upgrading the bootloader requires a GBL file with only the bootloader upgrade image.

Security of the bootloader upgrade process is provided by signing the GBL file. See *Creating a Signed and Encrypted GBL Upgrade Image File from an Application* on the *Gecko Bootloader Security Features* page.

Bootloader Upgrade on Bootloaders with Communication Interface (Standalone Bootloaders)

The process is illustrated in the following figure:



1. The device reboots into the bootloader.
2. A GBL file containing only a bootloader upgrade image is transmitted from the host to the device.
3. The contents of the GBL Bootloader tag are written to the data region in flash
4. The device reboots into the Secure Engine.
5. The Secure Engine replaces the main bootloader with the new version found in the bootloader upgrade GBL.
6. The device boots into the new main bootloader. Bootloader upgrade is complete.

Copyright © 2025 Silicon Laboratories. All rights reserved.

A bootloader upgrade is started in the same way as an application upgrade.

Downloading and Applying a Bootloader GBL Upgrade File

When the bootloader has entered the receive loop, a GBL upgrade file containing a bootloader upgrade is transmitted to the bootloader. When a packet is received, it is passed to the image parser. The image parser parses the data and returns bootloader upgrade data in a callback. The bootloader core implements this callback and flashes the data to the flash at the bootloader upgrade location.

The bootloader prevents a newly uploaded bootloader upgrade image from being interpreted as valid by holding back parts of the bootloader upgrade vector table until the GBL file hash and GBL signature (if required) have been verified.

When a complete bootloader upgrade image is received, the bootloader signals the Secure Engine that it should enter firmware upgrade mode. Secure Engine communication is used to signal that bootloader upgrade is ready to be performed.

The authenticity of the main bootloader optionally can be verified before applying the bootloader upgrade. See *Setting a Version Number* on the *Configuring the Gecko Bootloader* page for more information about versioning bootloader images.

Upgrading Bootloaders without Secure Boot to Bootloaders with Secure Boot

A bootloader without the secure boot feature can be upgraded to a bootloader with the secure boot feature, using the following procedure:

1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the bootloader needs to be higher than the bootloader on the device.
   - Turn on secure boot in Simplicity Studio by going to the **Bootloader Core** component and selecting the **Enable secure boot** option.
   - (Optional) In the **Bootloader Core** component, select the **Require signed firmware upgrade files** option. This means that the Gecko Bootloader will only accept signed GBL files.
2. Generate a public/private Signing Key pair. See the *Generating Keys* section on the *Gecko Bootloader Security Features* page for more information on creating a Signing Key pair.
3. Write the public key generated from the previous step to the device. The public key is stored as a manufacturing token in the device by default. Key locations are defined in the bootloader project file btl_security_tokens.c.
4. Create a GBL file using the Gecko Bootloader image. The GBL file needs to be signed/unsigned depending on the current configuration of the Gecko Bootloader running on the device. For more details on creating a GBL file, see the *Creating a Signed and Encrypted GBL Upgrade Image File from an Application* section on the *Gecko Bootloader Security Features* page.
5. Upload the GBL file. For more details on the upgrade process, see the *Bootloader Upgrade on Bootloaders with Communication Interface (Standalone Bootloaders)* section.

Enabling Secure Boot RTSL on Series 3 Devices

Secure Boot RTSL (Root of Trust and Secure Loader) can be enabled using the following procedure:

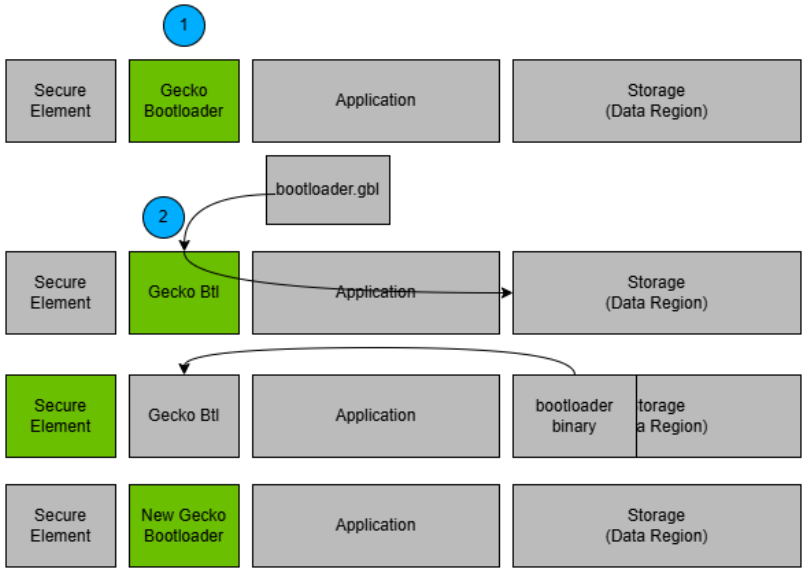
1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the Gecko Bootloader needs to be higher than the Gecko Bootloader on the device.
   - Turn on secure boot in Simplicity Studio by going to the **Bootloader Core** component and selecting the **Enable secure boot** option.
   - (Optional) In the **Bootloader Core** component, select the **Require signed firmware upgrade files** option. This means that the Gecko Bootloader will only accept signed GBL files.
2. Generate a public/private Signing Key pair. See the *Generating Keys* section on the *Gecko Bootloader Security Features* page for more information on creating a Signing Key pair.
3. Prepare an application that installs the public key generated from step 2 to the Secure Engine One-time Programmable memory. Installing a key in the VSE requires a reset routine. Make sure that the application does not end up in the reset loop. Create an unsigned GBL file from this application and upload it. For more information on installing public keys, see the *Creating a Signed and Encrypted GBL Upgrade Image File from an Application* section on the *Gecko Bootloader Security Features* page.
4. Sign the Gecko Bootloader image generated from step 1 using the private key generated in step 2. See the *Signing an Application Image for Secure Boot* section on the *Gecko Bootloader Security Features* page for more information on signing binaries.

Copyright © 2025 Silicon Laboratories. All rights reserved.

5. Make a custom application that turns on secure boot on the Secure Engine and sign this application binary with the private key generated from step 2.
6. Create a GBL file using the Gecko Bootloader image from step 4.
7. Create a GBL file using the application from step 5. The GBL file need to be signed if the **Bootloader Core** component option **Require signed firmware upgrade files** was selected in step 1.
8. Upload the GBL file containing the Gecko Bootloader image.
9. Upload the GBL file containing the application.

Bootloader Upgrade on Application Bootloaders with Storage

The process is illustrated in the following figure.



1. A single GBL file containing a bootloader upgrade image is downloaded onto the storage medium of the device.
2. The device reboots into the main bootloader.
3. The main bootloader verifies the integrity of the upgrade image and then resets the device with reset reason BOOTLOADER_RESET_REASON_UPGRADE to apply the upgrade.
4. The device reboots into the Secure Engine.
5. The Secure Engine replaces the main bootloader with the new version.
6. The device boots into the new main bootloader.

A bootloader upgrade is started in the same way as an Application Upgrade. A GBL file containing a bootloader is written to storage by the application, and the bootloader is entered.

The bootloader iterates over the list of storage slots marked for bootload and attempts to verify the GBL file stored within. Verification returns information about whether the GBL file contains a bootloader, or both a bootloader and an application. The image parser parses the file. If the GBL file contains a bootloader, the bootloader upgrade data is returned in a callback. The bootloader core implements this callback and flashes the data to internal flash at the bootloader upgrade location.

The bootloader prevents a newly uploaded bootloader upgrade image from being interpreted as valid by holding back parts of the bootloader upgrade vector table until the GBL file hash and GBL signature (if required) have been verified.

Secure Engine communication interface is used to signal the Secure Engine that a bootloader upgrade is ready to be performed.

On Series 3 devices, the authenticity of the main bootloader optionally can be verified before applying the bootloader upgrade. See the *Setting a Version Number* section on the *Configuring the Gecko Bootloader* page for more information about versioning bootloader images.

The new main bootloader is entered, and the images in the list of storage slots marked for bootload are verified. When the image parser parses the slot containing the GBL file with the bootloader + application upgrade, the version number of the bootloader upgrade is equal to the running main bootloader version, so another bootloader upgrade will not be performed.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Instead, the application upgrade data are returned in a callback. Bootloading of the new application proceeds as described in the *Application Bootloader Operation* section on the *Gecko Bootloader Operation Application Upgrade* page.

Storage Space Size Configuration

The storage space size must be configured to have enough space to store the upgrade images.

Upgrading Bootloaders without Secure Boot to Bootloaders with Secure Boot

A bootloader without the secure boot feature can be upgraded to a bootloader with the secure boot feature. The procedure is as follows:

1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the bootloader needs to be higher than the bootloader on the device.
   - Turn on secure boot from the **Bootloader Core** component in Simplicity Studio by selecting the **Enable secure boot** option.
2. Generate a public/private Signing Key pair. See the *Generating Keys* section on the *Gecko Bootloader Security Features* page for more information on creating a Signing Key pair.
3. Write the public key generated from the previous step to the device. The public key is stored as a manufacturing token in the device by default. This key can be written by application code running on the device. The Gecko Bootloader prepared from step 1 can be modified to look for the decryption and signature keys in a different location. Key locations are defined in the bootloader project file `btl_security_tokens.c`.
4. Prepare a signed application image using the private key generated in step 2. See the *Signing an Application Image for Secure Boot* section on the *Gecko Bootloader Security Features* page for more information on signing an application.
5. Create a GBL file using the Gecko Bootloader image and the signed application image. The GBL file needs to be signed/unsigned depending on the configuration of the Gecko Bootloader running on the device. For more details on creating a GBL file, see the *Creating a Signed and Encrypted GBL Upgrade Image File from an Application* section on the *Gecko Bootloader Security Features* page.
6. Upload the GBL file. For more details on the upgrade process, see the *Bootloader Upgrade on Application Bootloaders with Storage* section.

Enabling Secure Boot RTSL on Series 3 Devices

Secure Boot RTSL can be enabled by using the following procedure:

1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the Gecko Bootloader needs to be higher than the Gecko Bootloader on the device.
   - Turn on secure boot from the **Bootloader Core** component in Simplicity Studio by selecting the **Enable secure boot** option.
   - (Optional) In the **Bootloader Core** component, select the **Require signed firmware upgrade files** option. This means that the Gecko Bootloader will only accept signed GBL files.
2. Generate a public/private Signing Key pair. See the *Generating Keys* section on the *Gecko Bootloader Security Features* page for more information on creating a Signing Key pair.
3. Prepare an application that installs the public key generated from step 2 to the Secure Engine One-time Programmable memory. Installing a key in VSE requires a reset routine. Make sure that the application does not end up in the reset loop. Create an un- signed GBL file from this application and upload it. For more details on creating a GBL file, see the *Creating a Signed and Encrypted GBL Upgrade Image File from an Application* section on the *Gecko Bootloader Security Features* page.
4. Sign the Gecko Bootloader image generated from step 1 using the private key generated in step 2. See the *Signing an Application Image for Secure Boot* section on the *Gecko Bootloader Security Features* page for more information on signing binaries.
5. Make a custom application that turns on secure boot on the Secure Engine and sign this application binary with the private key generated from step 2. For more details on how to turn on secure boot on the Secure Engine.
6. Create a GBL file using the Gecko Bootloader image from step 4 and the application from step 5. The GBL file must be signed if the **Bootloader Core** component option **Require signed firmware upgrade files** was selected in step 1. For more details on creating a GBL file, see the *Creating a Signed and Encrypted GBL Upgrade Image File from an Application* section on the *Gecko Bootloader Security Features* page.
7. Upload the GBL file containing the Gecko Bootloader image and the application.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader Operation Secure Engine Upgrade

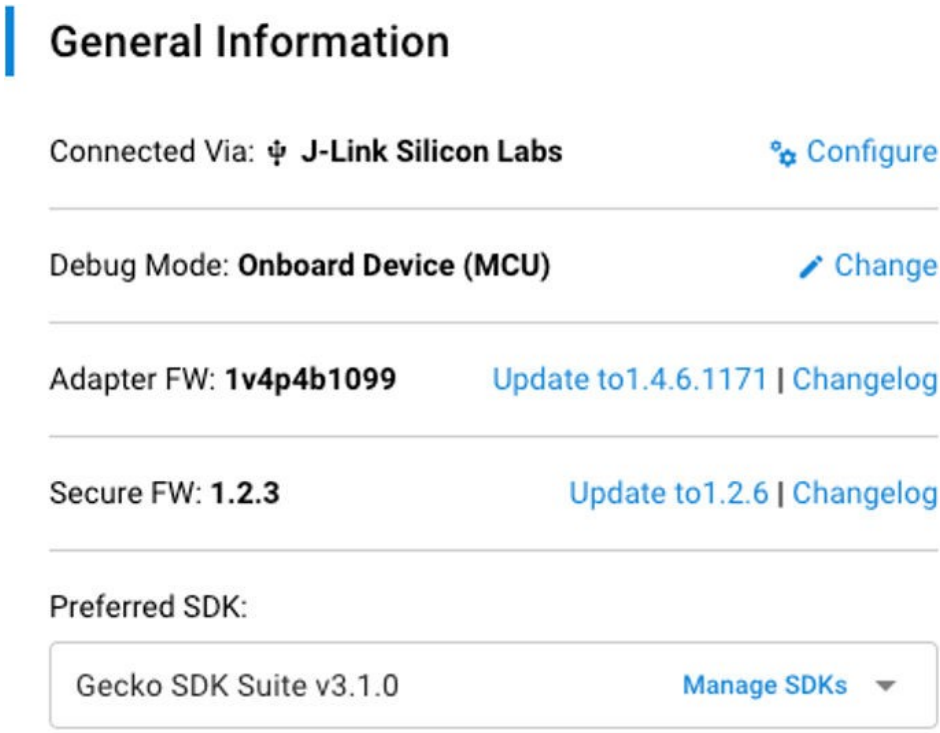
Gecko Bootloader Operation - Secure Engine Upgrade

The Secure Engine is upgradable and for both application and standalone type of bootloader a GBL file containing the Secure Engine Upgrade image has to be flashed or sent to the bootloader.

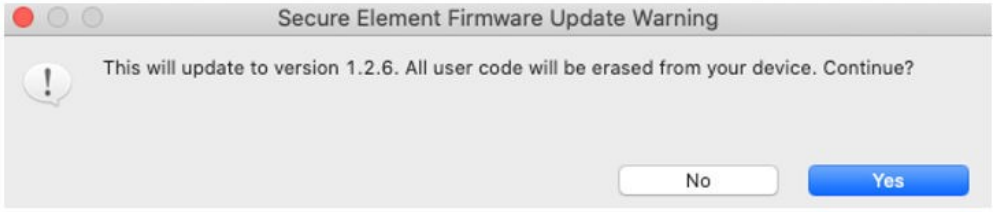
A bootloader upgrade can also be included in the same GBL file in application mode, or as a second GBL file in standalone mode. The figures that illustrate Gecko Bootloader operation in this section do not provide information about the bootloader memory layouts for different devices. For more details refer to the *Memory Space for Bootloading* section in Bootloader Fundamentals.

Signed and encrypted Secure Engine upgrade images are provided by Silicon Labs through Simplicity Studio. Upgrade images with the same or lower version number than the running Secure Engine will be ignored.

To download Secure Engine firmware images, connect a Series 3 device and select a preferred SDK. The Secure Firmware **Update to x.x.x** link appears in the Launcher Perspective, as shown in the following figure.



Click **Update to x.x.x** next to Secure FW: x.x.x. A warning dialog box appears. Click **Yes** to continue.



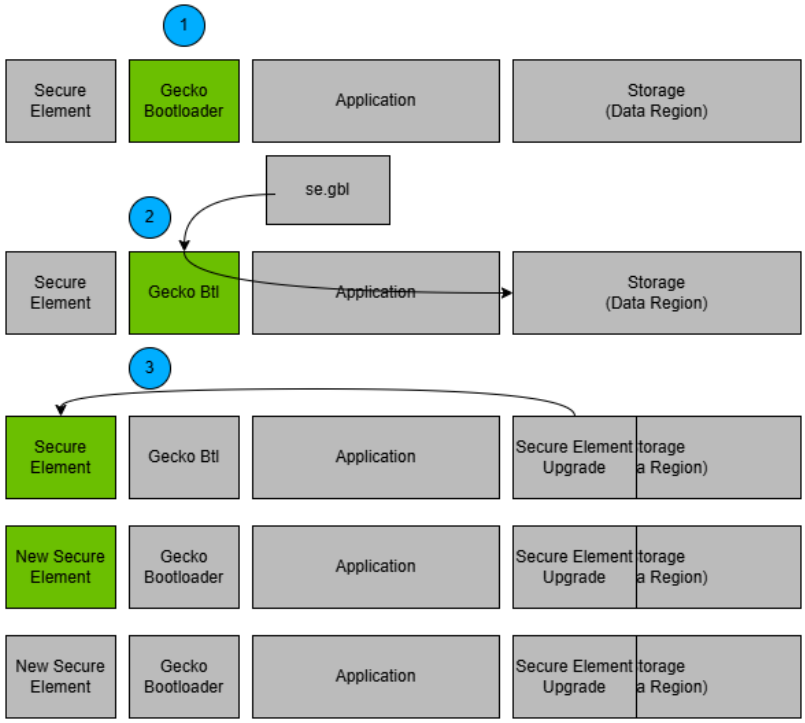
Copyright © 2025 Silicon Laboratories. All rights reserved.

The Launcher Perspective is then updated so that the current Secure Firmware version and link version are the same.

The Secure Engine firmware images can be found in the *util/se_release/public* directory under the Gecko SDK. Simplicity Studio displays the SE firmware version available in the Gecko SDK selected.

Secure Engine Upgrade on Bootloaders with Communication Interface (Standalone Bootloaders)

The process is illustrated in the following figure.



1. The device reboots into the bootloader.
2. A GBL file containing only a Secure Engine upgrade image is transmitted from the host to the device.
3. The device reboots into the Secure Engine.
4. The Secure Engine is replaced by the new version found in the pre-configured upgrade location.
5. The device boots into the bootloader.

Downloading and Applying a Secure Engine GBL Upgrade File

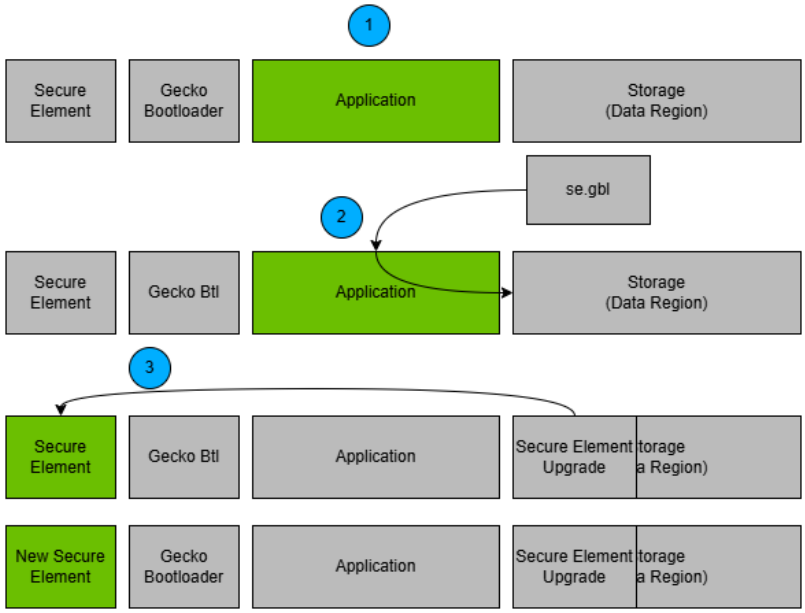
When the bootloader has entered the receive loop, a GBL upgrade file containing a Secure Engine upgrade is transmitted to the bootloader. When a packet is received, it is passed to the image parser. The image parser parses the data and returns Secure Engine upgrade data in a callback. The bootloader core implements this callback and flashes the data to flash at the pre-configured storage data region.

Copyright © 2025 Silicon Laboratories. All rights reserved.

When a complete Secure Engine upgrade image is received, the bootloader signals the Secure Engine that it should enter firmware upgrade mode. This is done by the Secure Engine communication interface that is used to signal that bootloader upgrade is ready to be performed.

Secure Engine Upgrade on Application Bootloaders with Storage

The process is illustrated in the following figure.



1. A single GBL file containing a Secure Engine upgrade image is downloaded onto the storage medium of the device.
2. The device reboots into the bootloader.
3. The upgrade image will be fetched directly from the GBL file in storage instead of first copying the image to the pre-configured upgrade location.
4. The device reboots into the Secure Engine.
5. The Secure Engine is replaced by the new version found in the pre-configured upgrade location (or directly from storage, ref. 3b).
6. The device boots into the main bootloader.
7. The bootloader applies the application image from the GBL upgrade file.
8. The device boots into the application. Secure Engine upgrade is complete.

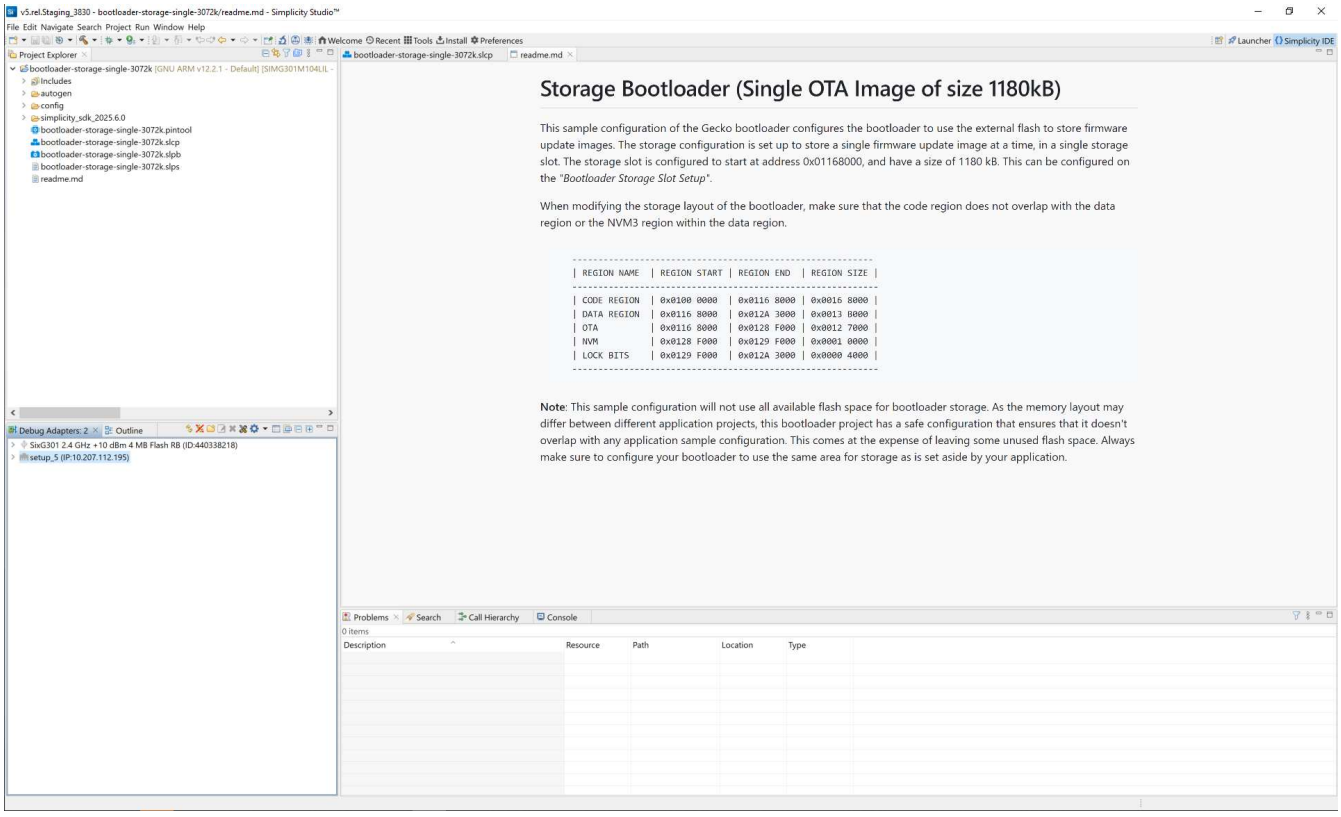
Copyright © 2025 Silicon Laboratories. All rights reserved.

Getting Started with the Gecko Bootloader

Getting Started with the Gecko Bootloader

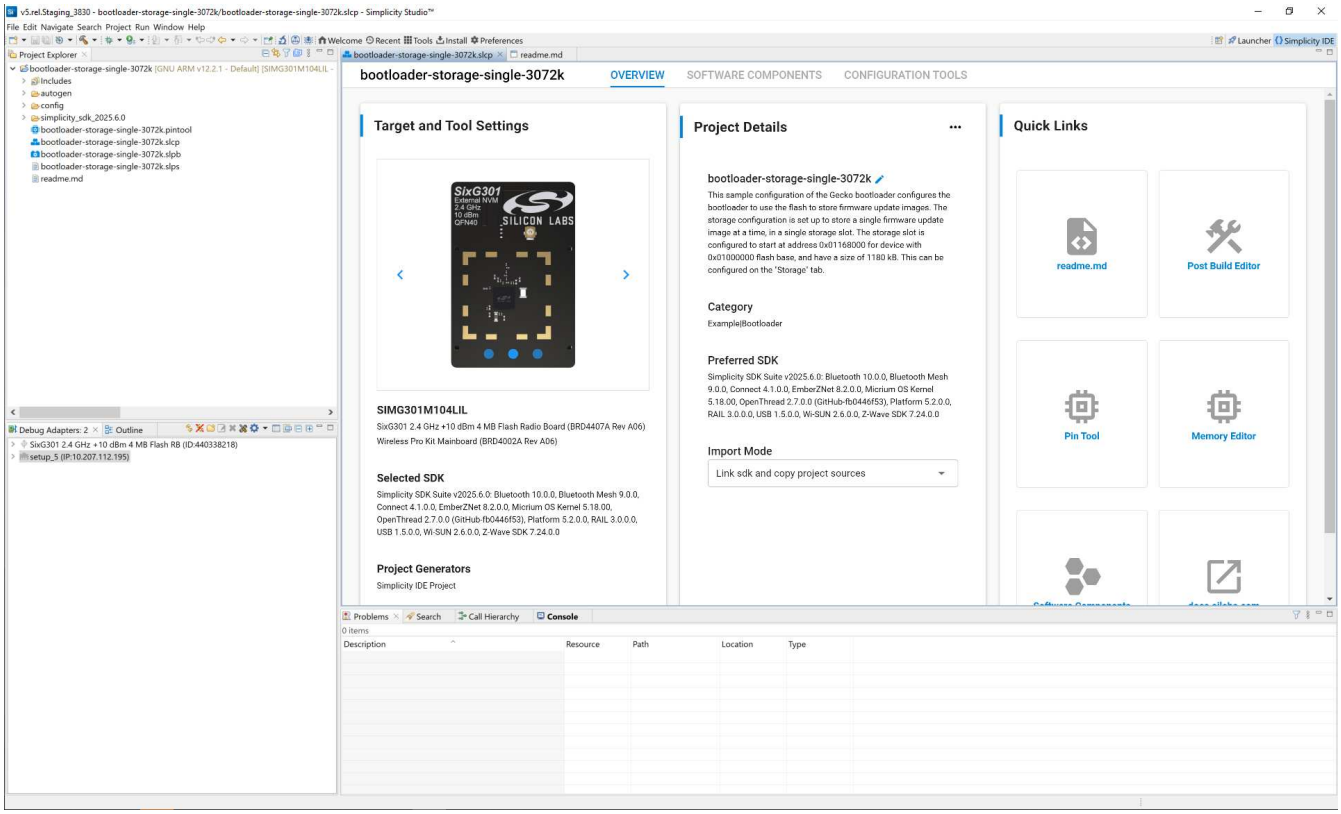
This section describes how to build a Gecko Bootloader from one of the provided examples. These instructions assume that you have installed Simplicity Studio 5, the SiSDK and associated utilities as described in the SDK's quick start guide, and that you are familiar with generating, compiling, and flashing an example application in the relevant version.

1. Create a project based on the Gecko Bootloader example of your choice. The project opens with a tab describing the example.

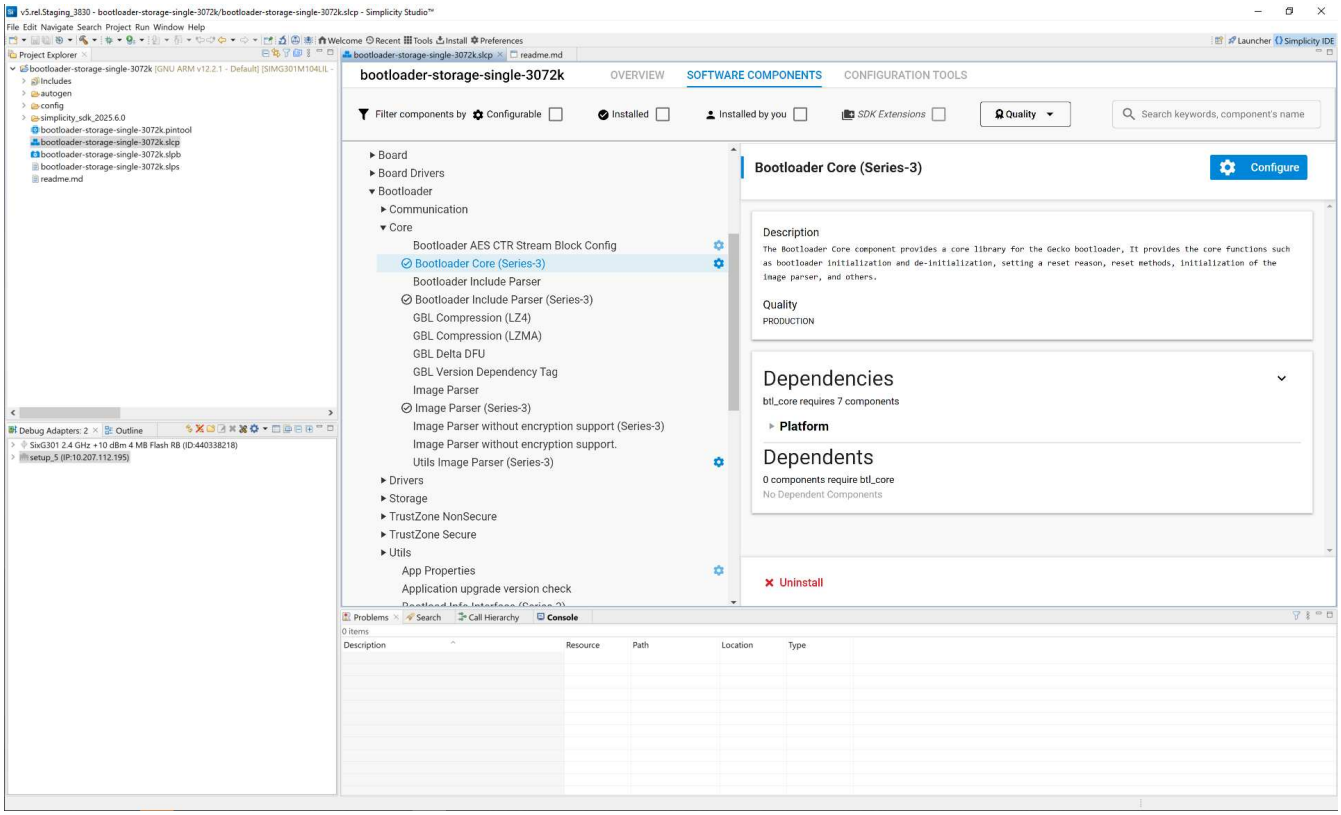


2. Click the project (*.slcp) tab to move to the Project Configurator interface.

Copyright © 2025 Silicon Laboratories. All rights reserved.

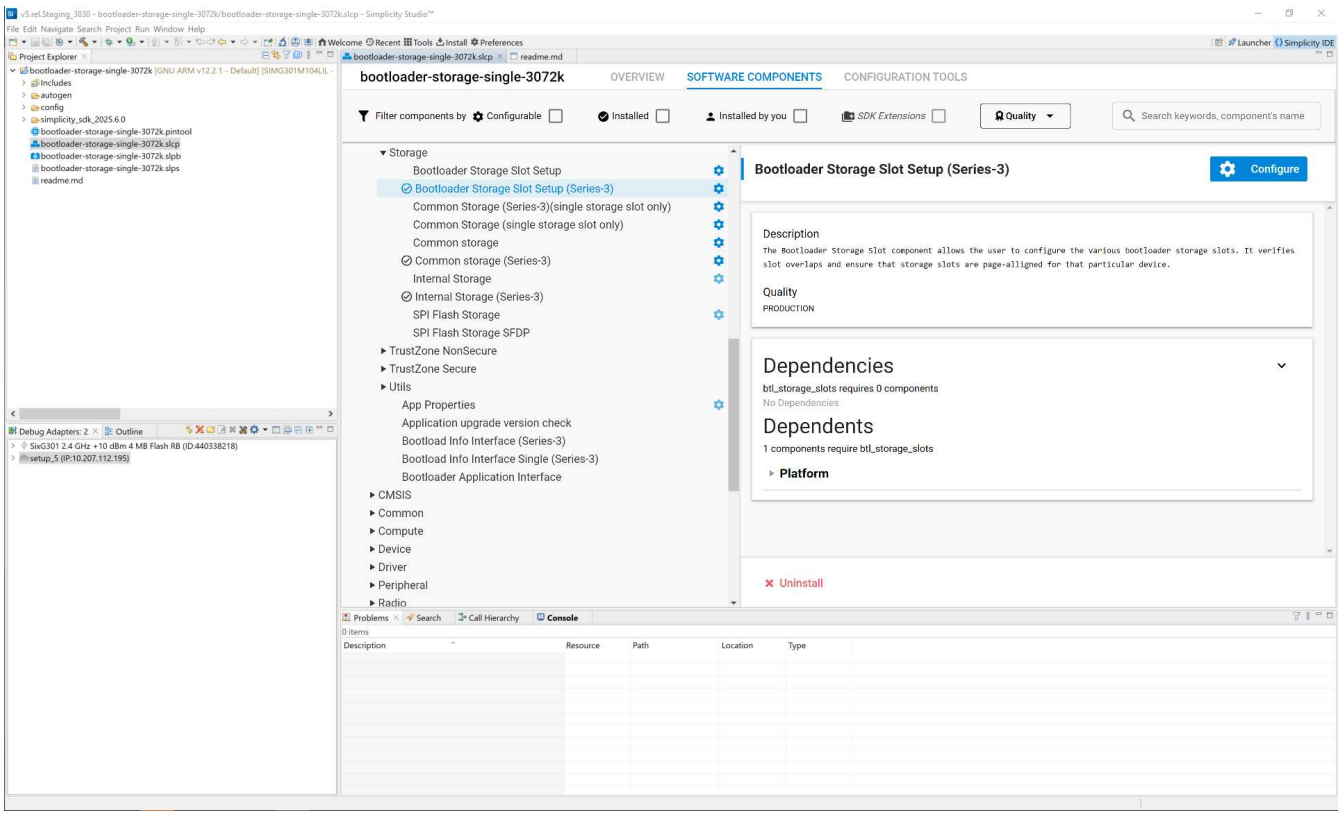


3. The Software Components tab shows the list of available components that can be installed in the project.

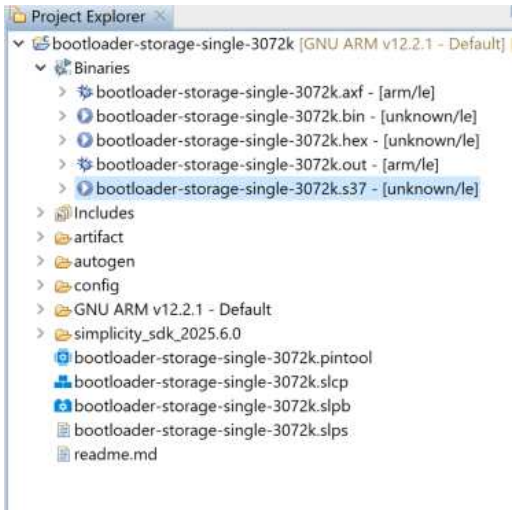


Copyright © 2025 Silicon Laboratories. All rights reserved.

4. The **Storage Slot Setup** component allows you to configure storage slots to be used if a storage component is also installed. The default configuration matches the target part and bootloader type. This component supports a maximum of three storage slots.



5. Click the **Build** (hammer) icon.
6. After the build is complete, the bootloader binaries are available in the **artifact** folder as depicted in the image below.



The image containing only a bootloader must be used to create a GBL file for bootloader upgrade.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configuring the Gecko Bootloader

Configuring the Gecko Bootloader

Configuring Storage

Gecko Bootloaders configured as application bootloaders must include an API to store and access image files. This API is based on the concept of storage slots, where each slot has a predefined size and location in memory and can be used to store a single upgrade image. Slots are configured in the **Bootloader Storage Slot Setup** component.

When multiple storage slots are configured, a bootload list is used to indicate the order in which the bootloader should access slots to find upgrade images. If multiple storage slots are supported, the application should write the bootload list by calling bootloader_setImageToBootload before rebooting into the bootloader to initiate a firmware upgrade process. The bootloader attempts to verify the images in these storage slots in sequence and applies the first image to pass verification. If only a single storage slot is supported, the bootloader uses this slot implicitly. A maximum of three slots may be configured in the **Bootloader Storage Slot Setup** component.

Storage Configuration

When configuring a Gecko Bootloader to obtain images fromflash, modify the following.

The **base address of the storage area** should be configured in the **Common Storage** component. This is the address at which the bootloader will place the prioritized list of storage slots to attempt to bootload from, if more than one storage slot is configured. In the default configuration, only a single storage slot is configured, so this value is set to 0, and isn't used. If more than one storage slot is configured, this value needs to be configured too.

The **location and size of the storage slots** can be configured using the **Bootloader Storage Slot Setup** Component (supports a maximum of three configurable storage slots). The addresses input here are absolute addresses (they are not offsets from the base address). If more than a single slot is configured, enough space must be reserved between the base address as configured in the **Common Storage** component and the first storage slot configured in the **Bootloader Storage Slot Setup** component. Enough space to fit two copies of the bootload list must be reserved. These two copies need to reside on different flash pages, to provide redundancy in case of power loss during writing. Two full flash pages therefore need to be reserved. The following figure illustrates how the storage area can be partitioned.

Storage with one storage slot

0	512kB
Application	Storage Slot 0 Size = 512kB

Storage with two storage slot

0	340kB	344kB	348kB	652kB
Application	Bootload list	Bootload list	Storage slot 0 Size = 340kB	Storage slot 1 Size = 340kB

> **Note**: The storage area partitioning in the example for two storage slots above does not take any NVM system into account. If using an NVM3 , take care to place and size the storage area in such a way that bootloader storage does not overlap with NVM3.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Compressed Upgrade Images

The Gecko Bootloader optionally supports compressed GBL files. In a compressed GBL file, only the application upgrade data and bootloader upgrade data is compressed, any metadata (if present) stays uncompressed. Signature and encryption operations on a compressed GBL work identically to on an uncompressed GBL.

To be able to use compressed upgrade images, a decompressor for the relevant compression algorithm must be added to the Gecko bootloader. The following table shows which compression algorithms are supported by the Gecko Bootloader, and which Bootloader component should be added to enable the feature. The table also shows how much space the decompressor takes up in the bootloader, and how big of a size reduction to expect for the compressed application upgrade image. Be aware of the bootloader size requirement. The bootloader space might be too small to fit the decompressors, depending on the device and enabled components.

Compression Algorithm	Component	Bootloader Size Requirement	Application Upgrade Size Reduction (typical)
LZ4	GBL Compression (LZ4)	< 1 kB	~ 10%
LZMA	Bootloader Compression (LZMA)	~5 kB flash, 18 kB RAM	~ 30%

It is important to note that the compressed GBL file stays compressed while being transferred to the device, and while it is stored in the upgrade area. It is decompressed by the bootloader when the upgrade is applied. This means that the running application in main flash will be identical to one that was installed using an uncompressed (normal) GBL file.

Compressed GBL files can only be decompressed by the bootloader when running standalone, not through the Application Interface. This means that upgrade image verification performed by the application prior to reboot will not attempt to decompress the application upgrade, it will only verify the signature of the compressed payload. After rebooting into the bootloader, it will decompress the image as part of the upgrade process.

LZMA Compression Settings

LZMA decompression is only supported for images compressed with certain compression settings. Simplicity Commander automatically uses these settings during gbl4 creation.

- Probability model counters: lp + lc <= 2. Simplicity Commander uses lp=1, lc=1.
- Dictionary size no greater than 8 kB. Simplicity Commander uses 8 kB.

Together, these settings cause the decompressor to require 18 kB of RAM for decompression – 10 kB for the counters and 8 kB for the dictionary.

The Gecko bootloader only supports decompressing payloads that contain the end mark as the last 8 bytes of the compressed stream.

Bootloader Example Configurations

The following sections describe the key configuration options for the example bootloader applications.

> **Note**: Security features are disabled for all example configurations. In development, Silicon Labs strongly recommends enabling security features to prevent unauthorized parties from uploading untrusted program code. See the *Using Application Image Security Features* section on the *Gecko Bootloader Security Features* page to learn how to configure the security features of the Gecko Bootloader.

UART XMODEM Bootloader

Standalone bootloader for Series 3 devices running the EmberZNet PRO and Silicon Labs Connect protocol stacks, using XMODEM-CRC over UART.

Copyright © 2025 Silicon Laboratories. All rights reserved.

In this configuration, the **UART XMODEM** communication component, **XMODEM Parser** component, and **Bootloader UART Driver** component are installed. For the example application to run on a custom board, the GPIO ports and pins used for UART need to be configured in the **Bootloader UART Driver** component. Here, Hardware Flow Control can be enabled or disabled, and the baud rate and pinout can be configured.

The **GPIO activation** component is also installed by default, allowing bootloader entry into firmware upgrade mode by activating a GPIO through reset. The GPIO pin used can be configured here. This component can be uninstalled if this functionality is not desired.

BGAPI UART DFU Bootloader

Standalone bootloader for the Bluetooth protocol stack, using the BGAPI protocol for UART DFU. This bootloader should be used for all NCP-mode Bluetooth applications.

In this configuration, the **BGAPI UART DFU** communication component and **Bootloader UART Driver** component are installed. For the example application to run on a custom board, the GPIO ports and pins used for UART need to be configured in the **Bootloader UART Driver** component. Here, Hardware Flow Control can be enabled or disabled, and the baud rate and pinout can be configured.

The GPIO activation component is also installed by default, allowing bootloader entry by activating a GPIO through reset. The GPIO pin used can be configured here. This component can be uninstalled if this functionality is not desired.

Storage Bootloader

Application bootloader for all wireless protocol stacks, using flash to store upgrade images received over the air by the application.

Multiple examples are provided, including configurations for 2 MB, 3 MB, 4 MB & 8 MB flash memory devices . **The storage layout should be modified before running the bootloader on any other devices**. In this configuration, the flash and common storage components are installed. The base address of the storage area is configured in the **Common Storage** component. The location and size of the storage slots can be configured using the **Bootloader Storage Slot Setup** component (provides up to three configurable storage slots). Default example applications are provided with configurations for both single storage slot and multiple storage slots.

The default storage slot configurations provided by the Gecko Bootloader **must** be configured to match the use-case-specific application configurations, as shown below.

Sample Applications	Storage Offset	Storage Size
Storage Bootloader (Single OTA Image of size 792kB)	0x010E0000	0x000C6000 (811,008 bytes)
Storage Bootloader (Single OTA Image of size 1180kB)	0x0128F000	0x00127000 (1,210,368 bytes)
Storage Bootloader (Single OTA Image of size 1540kB)	0x01379000	0x00181000 (1,577,984 bytes)

Image Acquisition Application Example Configuration

These examples illustrate applications that acquire and store a GBL upload image for an application bootloader. For the running bootloader to accept an application upgrade, the new application version must be higher than the existing version.

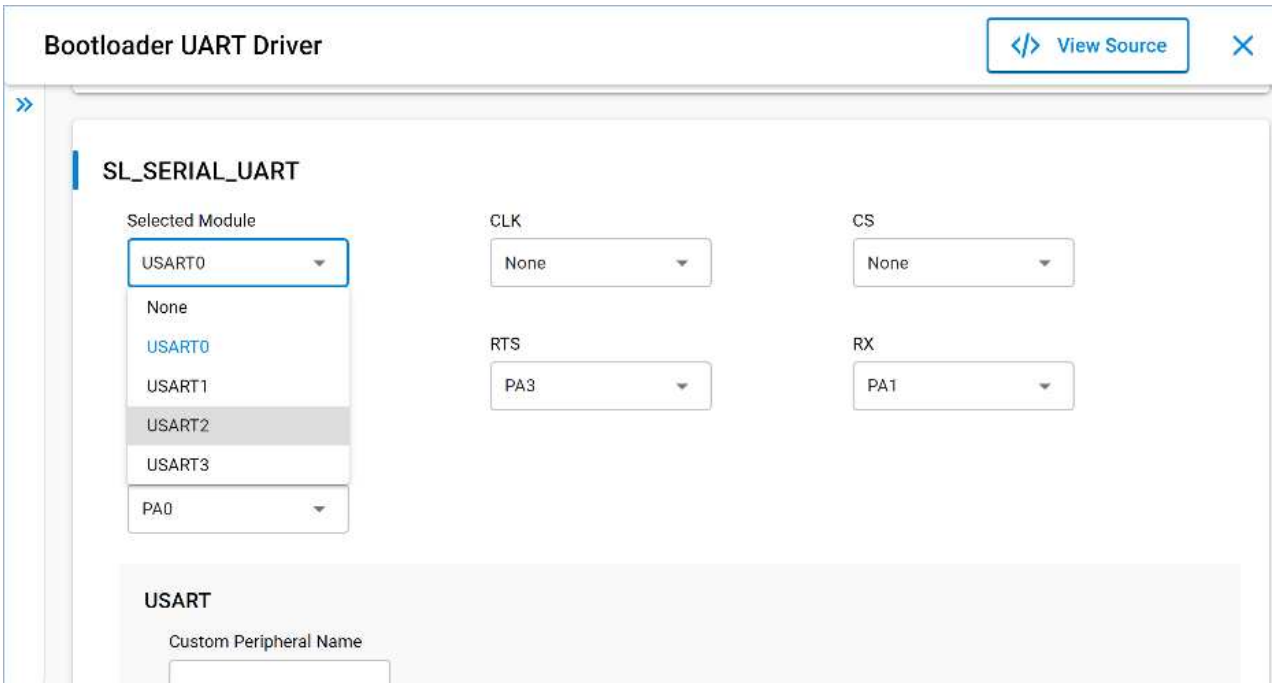
Setting a Version Number

To distinguish between different builds of the Gecko Bootloader, it is useful to set a version number. To perform a bootloader upgrade, not only must the running bootloader pass its integrity checks (see the *Downloading and Applying a Bootloader GBL Upgrade File* section on the *Gecko Bootloader Operation Bootloader Upgrade* page), but the bootloader upgrade image must also have a higher version number than the running bootloader image. A version number can be set using Simplicity Studio by configuring the **Bootloader Version Main Customer** option of the **Bootloader Core** component. This macro will be picked up by the config file **btl_config.h**, where it is combined with the version number of the Gecko Bootloader files provided by Silicon Labs.

Hardware Configuration

Copyright © 2025 Silicon Laboratories. All rights reserved.

The Gecko Bootloader uses the Pin Tool for configuration of pinout and other hardware-related settings. When Pin Tool configuration is available for a bootloader component, the relevant settings are shown in the Component Editor for that component.



The standalone Pin Tool User Interface can also be used to configure settings for Gecko Bootloader if desired.

While the Pin Tool provides configuration for many different peripherals, the Gecko Bootloader uses only the following Pin Tool modules:

- SERIAL is used by the UART Driver component to configure baud rate, flow control mode and pinout.
- VCOM is used by the UART Driver component to enable the serial interface if necessary (only on Silicon Labs Wireless Starter Kits).
- BTL_BUTTON is used by the GPIO Activation component.
- CMU HFXO frequency setting is used by the delay driver to calibrate timing if the core is running from the HFXO.

Other settings, like CMU oscillator configuration or DCDC configuration, are not taken into consideration by the default bootloader code. If using these configuration settings is desired, the required code must be added in btl_main.c.

> **Note**: While the delay driver uses the HFXO frequency setting from Pin Tool, the HFXO enable setting is not used to initialize the HFXO on startup. This setting is only used when calling the bootloader through the Application Interface, and the application has switched to the HFXO prior to calling the Bootloader Application Interface API.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Simplicity Commander and the Gecko Bootloader

Simplicity Commander and the Gecko Bootloader

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It is invoked using a simple CLI (Command Line Interface) that is also scriptable. You can use Simplicity Commander to perform these essential tasks:

- Generating key files for signing and encryption
- Signing application images for Secure Boot
- Creating GBL images (encrypted or unencrypted, signed, or unsigned)
- Parsing GBL images

Simplicity Commander is used throughout the examples in the following sections. For more information on executing commands to complete these tasks, see the Simplicity Commander User Guide.

> **Note**: Simplicity Commander also offers a GUI (Graphical User Interface) that can be used in the lab for typical tasks such as flashing device images. The functions described in this User Guide are performed from the CLI.

Creating GBL V4 Files Using Simplicity Commander

To create an unsigned GBL V4 file from an application **myapp.s37**, execute `commander gbl4 create myapp.gbl --config configfile.yaml`.

Example of a config file is as follows:

```
manifest:
 product_id:"00000000000000000000000000000000"
updates:
  -data:myapp.s37
   block_size: 0
```

To create an unsigned GBL V4 file from a bootloader upgrade **mybootloader.s37**, execute `commander gbl4 create mybootloader.gbl --config confligfile.yaml`.

Example of a config file is as follows:

```
manifest:
 product_id:"00000000000000000000000000000000"
updates:
  -data:mybootloader.s37
   block_size: 0
```

This file can be used with the standalone bootloader configurations of the Gecko Bootloader.

To create an unsigned GBL file from a Secure Engine, upgrade **mySecureElement.seuv2**, and execute `commander gbl4 create mySecureElement.gbl –config configfile.yaml`.

Example of a config file is as follows:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
manifest:
 product_id:“00000000000000000000000000000000”
se_upgrade:
 se_file:mysecureElement.seuv2
```

The Secure Engine images, .seuv2, are provided by Silicon Labs and can be found through Simplicity Studio. See the *Gecko Bootloader Operation - Secure Engine Upgrade* section on the *Gecko Bootloader Operation Secure Engine Upgrade* page.

The command can also be used to create a single upgrade image, suitable for use with application bootloader configurations of the Gecko Bootloader: `commander gbl4 create myupgrade.gbl –config configfile.yaml`.

Example of a config file is as follows:

```
manifest:
 product_id:“00000000000000000000000000000000”
updates:
 -data:myapp.s37
 -data:mybootloader.s37

se_upgrade:
 se_file: mysecureElement.seuv2
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader Security Features

Gecko Bootloader Security Features

About Bootloader Image Security

Secure Boot and Secure Firmware Upgrade, discussed in the following sections, enables Gecko Bootloader to provide authenticity and integrity checks on the Application image, which provides a sufficient level of security for many applications. However, in systems without a hardware root of trust, no process checks the authenticity or integrity of the Gecko Bootloader itself. Its security is provided solely by the device hardware and the robustness of the software running on the device.

The native behavior of Firmware Upgrade will prevent accidental version rollback of Gecko Bootloader under normal usage conditions.

About Application Image Security

The Gecko Bootloader can enforce security on two levels:

- Secure Boot refers to the verification of the authenticity of the application image in main flash on every boot of the device.
- Secure Firmware Upgrade refers to the verification of the authenticity of an upgrade image before performing a bootload, and optionally enforcing that upgrade images are encrypted.

Secure Boot Procedure

When Secure Boot is enabled, the cryptographic signature of the application image in flash is verified on every boot before the application is allowed to run. Secure Boot is not enabled by default in the example configurations provided by Silicon Labs, but enabling it is highly recommended to ensure the validity and integrity of firmware images.

Signature Algorithms

The Gecko Bootloader supports the ECDSA-P256-SHA256 cryptographic signature algorithm. This is the ECDSA (elliptical curve digital signature algorithm) of the SHA-256 digest of the application firmware image, using the NIST P-256 (secp256r1) curve.

Summary of Operation

1. On boot, the bootloader checks the application image for information about whether it is signed.
2. The type of signature and signature location is determined.
3. If the type of signature does not match the requirements of the bootloader, the bootloader enters device firmware upgrade mode and prevents the application from running.
4. According to the chosen signature algorithm, the signature of the contents of flash from the beginning of the application to the location of the signature is compared to the signature at the signature location.
5. If the signatures do not match, the bootloader enters device firmware upgrade mode and prevents the application from running.

Secure Boot using ECDSA-P256-SHA256

For an image to be signed for Secure Boot, the application needs to contain a copy of the **ApplicationProperties_t** struct. This struct contains information about which signature algorithm is used, and where to find the signature.

On every boot, the bootloader calculates the SHA-256 digest of the application image, from the beginning of the application to the start of the signature. The signature of the SHA-256 digest is then verified using ECDSA-P256.

Copyright © 2025 Silicon Laboratories. All rights reserved.

If the signature is valid, the application is allowed to boot. Else, the bootloader is entered, and an application upgrade is attempted if one is available.

Simplicity Commander can be used to generate a key pair and write the public key to the device. See the Simplicity Commander User Guide for more information.

Secure Boot with Application Rollback Protection

On every boot, the application version in the **ApplicationData_t** struct is stored at the end of the bootloader area in flash, which is used to prevent applications from being downgraded. The application version can remain the same for upgrades.

If application rollback prevention component is installed in the bootloader project, then before applying the upgrade image, the bootloader will check the application version in the application properties structure that resides inside the signed/encrypted GBL file and will only apply the OTA image to application area if the application version in ApplicationData structure is equal or higher than the highest application version last seen.

The application rollback prevention feature can be enabled in the **Bootloader Core** component by selecting the **Enable application rollback protection** option. The **Minimum application version allowed** option can be used to configure the minimum application version that should be allowed to boot.

The application versions are stored in the SE OTP area.

Secure Boot Using a Certificate

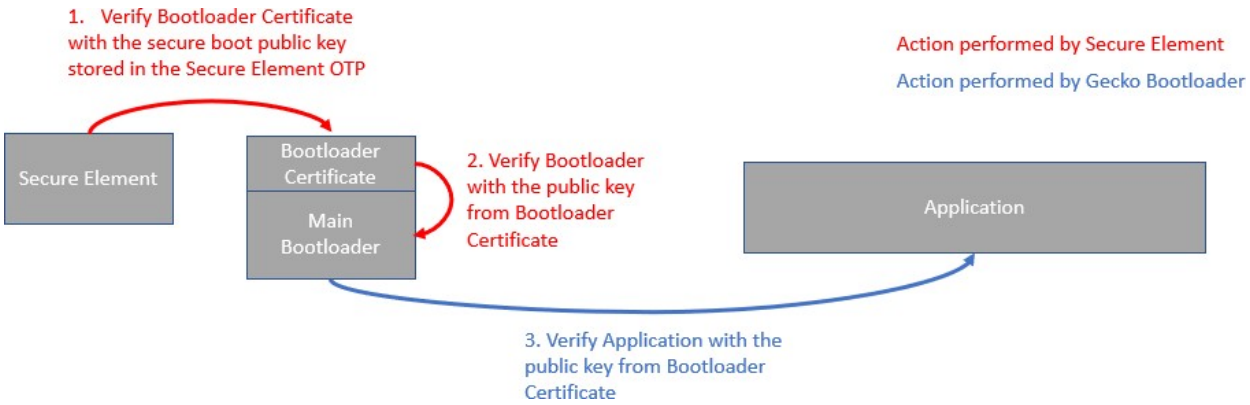
On Series 3 devices, a certificate-based secure boot operation is supported. The Certificate contains:

- Struct version: The version of the certificate structure.
- Public key: ECDSA-P256 public key, X and Y coordinates concatenated, used to validate the image.
- Certificate version: The version of the running certificate.
- Signature: ECDSA-P256 signature, used for the authentication of the public key and the certificate version.

The definition of the certificate struct can be found in `api/application_properties.h`.

To utilize certificate-based secure boot, configure Secure Engine to authenticate the bootloader image by configuring the certificate- based secure boot option in the Secure Engine OTP. Configure the Gecko Bootloader to enable certificate-based secure boot in the **Bootloader Core** component by selecting the **Enable certificate support** option. The Gecko Bootloader certificate must be signed by the private key pair of the public key stored in the Secure Engine OTP. For more information on the key storage, see the *Key Storage* section.

The certificate-based secure boot procedure is illustrated in the following figure.



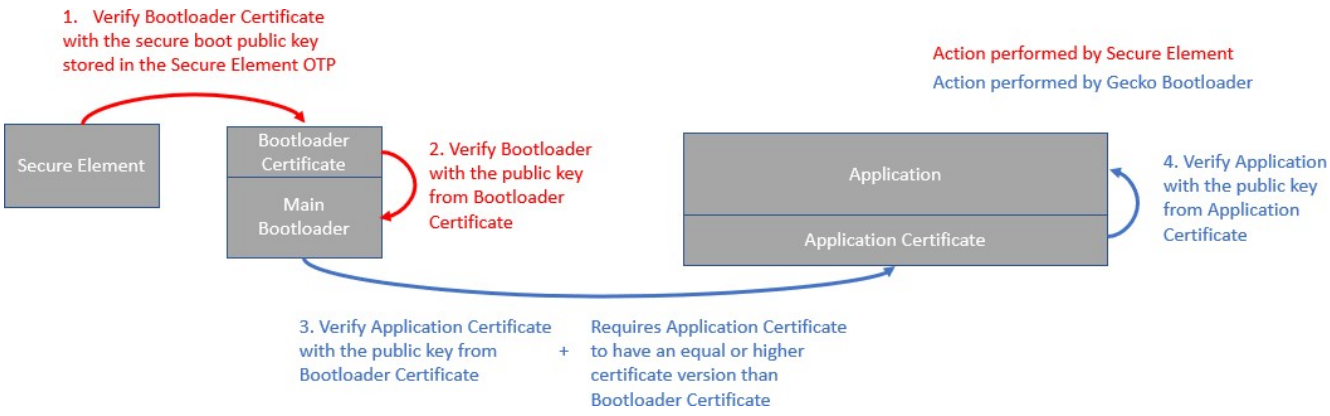
Once the certificate-based secure boot option on Secure Engine is turned on, Secure Engine verifies the Gecko Bootloader certificate. The public key stored in the certificate is used to validate the signature of the Gecko Bootloader. Secure Engine will not accept bootloader images without a certificate.

If only the secure boot option is enabled (not certificate-based) on Secure Engine, and Secure Engine identifies a certificate, the certificate will be used to validate the bootloader image. If the certificate version from the bootloader

Copyright © 2025 Silicon Laboratories. All rights reserved.

image is higher than 0 and it gets verified once, Secure Engine will never again accept direct signed bootloader images without a certificate.

The Gecko Bootloader will authenticate the direct signed application using the public key stored in the Gecko Bootloader certificate. If the application contains a certificate, Gecko Bootloader will authenticate it. The procedure is illustrated in the following figure.



After authentication of the application certificate, Gecko Bootloader verifies the signature of the application using the public key from the application certificate. In addition, Gecko Bootloader compares the Gecko Bootloader certificate version against the application certificate version. All application images with certificate version lower than the certificate version of the Gecko Bootloader will be rejected. Gecko Bootloader can be configured to only allow applications with certificates to boot by configuring the **Bootloader Core** component by selecting the **Reject direct signed images** option.

The **ApplicationProperties_t** struct contains the certificate struct **ApplicationCertificate_t**. The certificate struct can be injected to images that contain an **ApplicationProperties_t** with **ApplicationCertificate_t**. To inject a certificate to an image, issue the following command from Simplicity Commander:

```
commander convert <image file> --secureboot --keyfile <signing key> --certificate <certificate>
--outfile <signed image file with certificate>
```

Secure Firmware Upgrade

The Gecko Bootloader supports a secure firmware upgrade process. This is achieved by using symmetric encryption to encrypt the upgrade image, and asymmetric cryptography to sign the upgrade image. Symmetric encryption provides confidentiality, and asymmetric cryptography provides integrity and authenticity. Note that encryption alone is not enough to provide authenticity.

Encryption Algorithms

The Gecko Bootloader supports the AES-CTR-128 encryption algorithm. The GBL upgrade file is encrypted using 128-bit AES in Counter mode with a random nonce as the initial counter value.

To make use of the OTA decryption key stored in the Secure Engine OTP, the **Use symmetric key stored in Secure Engine storage** option in the **Bootloader Core** component must be selected. Simplicity Commander can be used to generate an OTA decryption key and write the key to the device. For more information on storing the OTA decryption key on Series 3 devices, see Simplicity Commander User Guide.

The Secure Engine OTP key support depends on the SE Manager component, which is enabled by default.

Signature Algorithms

The Gecko Bootloader supports the ECDSA-P256-SHA256 cryptographic signature algorithm. This is the ECDSA signature of the SHA-256 digest of the GBL upgrade file, using the NIST P-256 (secp256r1) curve.

Summary of Operation

Copyright © 2025 Silicon Laboratories. All rights reserved.

Before starting a firmware upgrade process, the application can verify an image in storage by calling into the bootloader verification functions. For bootloaders with Communication Interface, the host device should verify the image before sending it to the NCP or RCP.

During firmware upgrade, the GBL file is parsed, and if encrypted, decrypted on-the-fly. A GBL Signature Tag in the GBL file indicates to the bootloader that the file is signed, and the signature is verified. If signature verification fails, the firmware upgrade process is aborted.

On Series 3 devices, Gecko Bootloader will authenticate the GBL signature tag using the public key stored in the bootloader certificate if the **Enable certificate support** option is selected in the **Bootloader Core** component. A GBL Certificate tag in the GBL file indicates to the bootloader that the GBL certificate tag needs to be authenticated using the public key stored in the bootloader certificate. The certificate version in the GBL certificate tag is compared with the bootloader certificate and only a version equal or higher than the bootloader certificate is accepted. Once the GBL certificate tag is authenticated, the GBL file's signature is verified using the authenticated public key from the GBL certificate tag.

Using Application Image Security Features

This example assumes that a bootloader called **bootloader-uart-xmodem** has been built in Simplicity Studio.The relevant version can be flashed to the device using the Flash Programmer in Simplicity Studio or using Simplicity Commander.

This example provides two ways of signing the upgrade images. The first option uses Simplicity Commander to generate key material and sign data. This is suitable for development. The second option uses an external signer, such as a dedicated Hardware Security Module (HSM) to protect private key material and perform signing operations. Silicon Labs recommends using an HSM to safeguard private keys.

Generating Keys

To use the security features of the Gecko Bootloader, encryption and signing keys need to be generated. These keys must then be written to the device. The encryption key is used with the GBL file for secure firmware upgrade. The signing keys are used both with the GBL file for secure firmware upgrade and to sign the application image for Secure Boot.

Generating a Signing Key Using Simplicity Commander

```
commander util genkey --type ecc-p256 --privkey signing-key --pubkey signing-key.pub
```

This creates an ECDSA-P256 key pair for signing; `signing-key` contains the private key in PEM format and **must be kept secret from third parties**. This key will later be used to sign images and GBL files. `signing-key.pub` contains the public key in PEM format and can be used to verify GBL files using commander gbl4 info.

Generating a Signing Key Using a Hardware Security Module

When using a Hardware Security Module, the private key is kept secret inside the HSM. According to the instructions from your HSM vendor, have it generate an ECDSA-P256 key pair and export the public key in PEM format to the file **signing-key.pub**. Then use Simplicity Commander to convert the key to token format, suitable for writing to the Series 3 device.

```
commander gbl keyconvert --type ecc-p256 signing-key.pub
```

Generating an Encryption Key

```
commander util genkey --type aes-ccm --outfile encryption-key
```

This creates an AES-128 key for encryption in the file **encryption-key**.

Writing Keys to the Device

> **Note**: Refer to the sections, *Writing the AES Decryption Key* and *Writing the Public Key to the Device* in the

Copyright © 2025 Silicon Laboratories. All rights reserved.

> Simplicity Commander User Guide.

Signing an Application Image for Secure Boot

If the bootloader enforces Secure Boot, the application needs to be signed to pass verification. On every boot, a SHA-256 digest of the application is calculated. The signature is verified using ECDSA-P256, with the same public key as for the GBL file signing. Signature verification failure prevents the application from booting.

Application images should contain an **ApplicationProperties_t** struct declaring the application version, capabilities, and other metadata. If **ApplicationProperties_t** is missing, the application image cannot be signed. For more details on adding **ApplicationProperties_t**, see the *Application Properties* section on the *Application Interface* page.

Using Simplicity Commander

Signing the application can be done with the command:

```
commander convert myapp.s37 --secureboot --keyfile signing-key --outfile myapp-signed.s37
```

Using a Hardware Security Module

The application can be prepared for signing by issuing the command:

```
commander convert myapp.s37 --secureboot --extsign --outfile myapp-for-signing.s37
```

Using an HSM, sign the output file **myapp-for-signing.s37**, and supply the resulting DER-formatted signature file **signature.der** back to Simplicity Commander:

```
commander convert myapp-for-signing.s37 --secureboot --signature signature.der --verify signing-key.pub
--outfile myapp-signed.s37
```

Creating a Signed and Encrypted GBL Upgrade Image File from an Application

To create a GBL file from an application, use `commander gbl4 create`.

Note that, as of this writing, secure application images can only be constructed through Simplicity Commander, not through the configuration options available through Simplicity Studio.

Application images should contain an **ApplicationProperties_t** struct declaring the application version, capabilities, and other metadata. If **ApplicationProperties_t** is missing, the application image cannot be signed. For more details on adding **ApplicationProperties_t**, see the *Application Properties* section on the *Application Interface* page.

Using Simplicity Commander to Sign

For an application called **myapp.s37**, use:

```
commander gbl4 create myapp.gbl –-config configfile.yaml
```

Example of a config file is as follows:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
manifest:
 product_id: "[16 B ID] (default from app or 00000000000000000000000000000000)"
 bundle_version: "[version] (default 0x00000000)"
 min_version: "[version] (default 0x00000000)"
 hash_function: "[none | sha256] (default sha256)"
 certificate: "[path to file] (optional)"
 signing_key: "[path to file] (optional)"
se_upgrade:
 se_file: "[path to file] (optional)"
updates:
 - data: <path to file>
  app_type: "[uint32] (default from app or 0x00000000)"
  app_capabilities: "[uint32] (default from app or 0x00000000)"
  version: "[uint32] (default from app or 0x00000000)"
  compression_scheme: "[none | lz4 | lzma] (default none)"
  encryption_scheme: "[none | aes-ctr-128] (default none)"
  block_size: "[size in bytes] (default 0/disabled)"
```

This single command performs three actions:

- Creates a GBL file
- Encrypts the GBL file
- Signs the GBL file

If Secure Boot is also desired, the application must be signed using `commander convert --secureboot` prior to creating the GBL.

System Security Considerations

The Gecko bootloader security features can be used to create a secure device, but do not create a secure system by themselves. This section goes over considerations that need to be taken when designing a secure system where the Gecko Bootloader is a component.

Key Storage

On Series 3 devices, the decryption key and the sign key used by the Gecko Bootloader in the Secure Engine OTP. The decryption key can be provisioned in the Secure Engine OTP using Simplicity Commander or using the Secure Engine Mailbox interface. Once a key value has been programmed into the Secure Engine OTP, it cannot be changed.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Application Interface

Application Interface

The bootloader has an application interface exposed through a function table in the bootloader. The application interface provides APIs to use bootloader functions for storing and retrieving upgrade images and verifying their integrity. APIs to reboot into the bootloader are also provided. For details see the Gecko Bootloader API Reference, at https://docs.silabs.com/gecko-platform/5.1.2/platform-bootloading-overview/folder.

If you are not using a protocol stack from Silicon Labs, the **api/btl_interface.h** header provides the bootloader application interface API. If you are using a protocol stack from Silicon Labs, the recommended bootloader interface API for the specific protocol stack should be used instead. The following files provide the implementation of the bootloader interface:

api/btl_interface.c (common interface)

api/btl_interface_storage.c (interface to storage functionality)

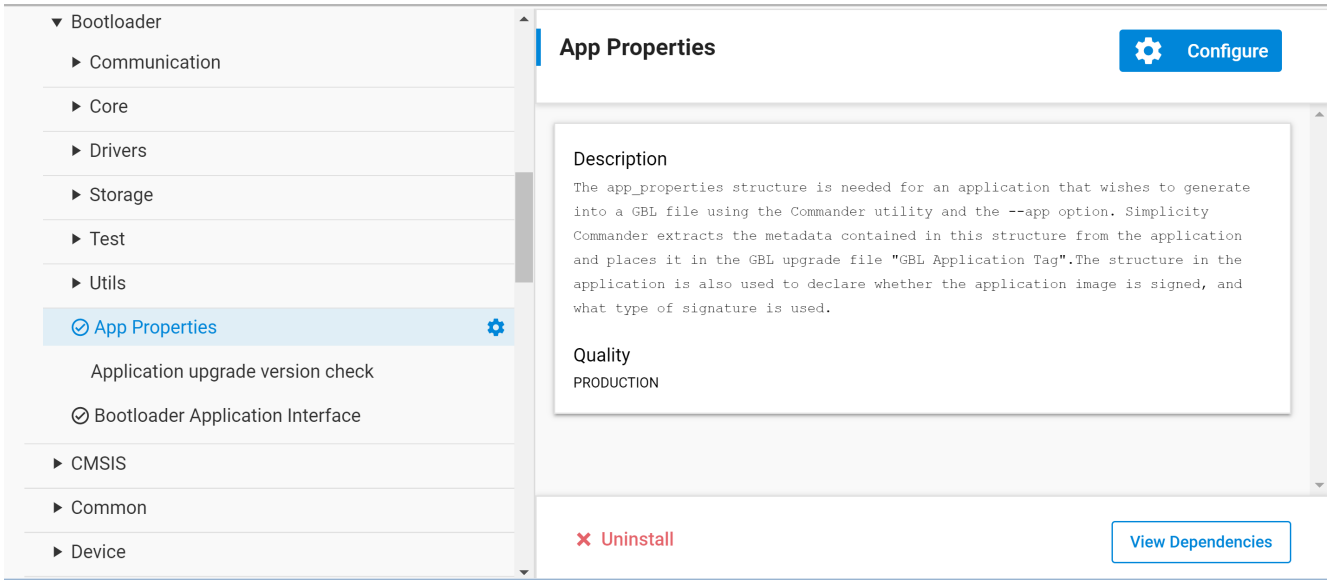
The application interface consists of functions that can be included in the customer application, and that communicate with the bootloader through the **MainBootloaderTable_t**. This table contains function pointers into the bootloader. The 10th word of the bootloader contains a pointer to this structure, allowing any application to easily locate it. Using the wrapper functions provided in the Bootloader Interface API is preferred over accessing the bootloader table directly. Modules include:

- **Application Storage Interface**: Application interface for interfacing with bootloader storage. The Storage Interface is only available on bootloaders that support the storage interface.
- **Common Application Interface**: Generic application interface available on all versions of the bootloader, independently of which components are present.

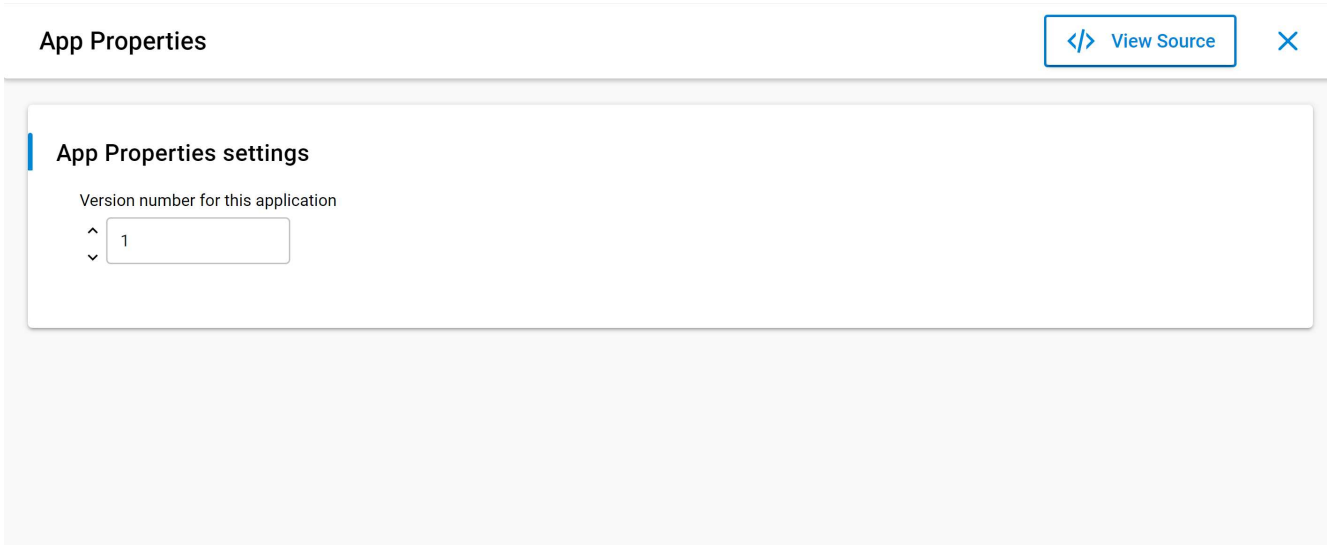
Application Properties

Application images should contain an **ApplicationProperties_t** struct declaring the application version, capabilities, and other metadata. The Simplicity Commander extracts the metadata contained in this structure from the application and places it in the UPDATE_MEMORY_SECTION tag in the GBL upgrade file. If the structure is not present in the application, Simplicity Commander will raise an error. The **ApplicationProperties_t** struct is added to the application on installing the **bootloader_interface** component to the application. The **bootloader_interface** component installs **bootloader_app_properties** component which adds an instance of **ApplicationProperties_t** named **sl_app_properties** to the project. The component adds a source file named **app_properties.c** and a configuration file named **app_properties_config.h**. This configuration file allows users to configure the application version via Simplicity Studio's Component Editor. To open the Component Editor, locate the **App Properties** component under **Platform > Bootloader** as shown below.

Copyright © 2025 Silicon Laboratories. All rights reserved.



Click **Configure** to open the Component Editor.



The application type is automatically populated based on the wireless stack used in building the project. The value of the application type is automatically set during the project generation step and can be found in **autogen/sl_application_type.h** file.

The contents of the **GBL Application Tag** can be extracted from a GBL file by a running application using the Application Storage interface. Note that the **GBL Application Tag** will only be added if the GBL file contains an application image, not if the GBL file only contains a bootloader upgrade or metadata.

The structure in the application is also used to declare whether the application image is signed, and what type of signature is used. This information is added by Simplicity Commander when signing the image using `commander convert (--secureboot, --extsign or -- signature)`. For the bootloader to locate the **ApplicationProperties_t** struct, if not already done by the linker, Simplicity Commander modifies word 13 of the application to insert a pointer to the **ApplicationProperties_t** struct when signing the application image for Secure Boot.

Error Codes

Copyright © 2025 Silicon Laboratories. All rights reserved.

Most Gecko bootloader APIs return error codes. The following table lists the groups of error codes that may be returned. The full list of error codes within each group can be found in *api/btl_errorcode.h* in the platform/bootloader directory of the SDK, as well as in the API Reference.

ID	Description
0x0	OK
0x01xx	Initialization error
0x02xx	Image verification error
0x04xx	Storage error
0x05xx	Bootload error
0x06xx	Security error
0x07xx	Communication error
0x09xx	XMODEM parser error
0x10xx	GBL file parser error
0x11xx	SPI slave driver error
0x12xx	UART driver error
0x13xx	Compression error
0x14xx	RTOS error

Copyright © 2025 Silicon Laboratories. All rights reserved.

Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher

Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher

Note: This section replaces *UG489: Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher*. Further updates to this user guide will be provided here.

This document describes the high-level implementation of the Silicon Labs Gecko Bootloader for EFM32 and EFR32 Series 1 and Series 2 microcontrollers, SoCs (System on Chips) and NCPs (Network Co-Processors), and provides information on different aspects of configuring the Gecko Bootloader. If you are not familiar with the basic principles of performing a firmware upgrade or want more information about upgrade image files, refer to *UG103.6: Bootloader Fundamentals*. For more information on using the Gecko Bootloader with different wireless stacks, see the following:

- Using the Gecko Bootloader with EmberZNet
- *UG435.06: Bootloading and OTA with Silicon Labs Connect v3.x*
- Using the Gecko Bootloader with Silicon Labs Bluetooth Applications

For more information on Series 2 device security, see:

- Series 2 Secure Boot with RTSL
- Series 2 Secure Debug
- *AN1222: Production Programming of Series 2 Devices*

For more information on security using Series 2 devices with Secure Vault, see:

- Anti-Tamper Protection Configuration and Use
- Authenticating Silicon Labs Devices using Device Certificates
- Secure Key Storage

Copyright © 2025 Silicon Laboratories. All rights reserved.

Overview

Overview

The Silicon Labs Gecko Bootloader is a common bootloader for all the newer MCUs and wireless MCUs from Silicon Labs. The Gecko Bootloader can be configured to perform a variety of functions, from device initialization to firmware upgrades. Key features of the bootloader are:

- Useable across Silicon Labs Gecko microcontroller and wireless microcontroller families
- In-field upgradeable
- Configurable
- Enhanced security features, including:
  - **Secure Boot**: When Secure Boot is enabled, the bootloader enforces cryptographic signature verification of the application image on every boot, using asymmetric cryptography. This ensures that the application was created and signed by a trusted party.
  - **Signed upgrade image file**: The Gecko Bootloader supports enforcing cryptographic signature verification of the upgrade image file. This allows the bootloader and application to verify that the application or bootloader upgrade comes from a trusted source before starting the upgrade process, ensuring that the image file was created and signed by a trusted party.
  - **Encrypted upgrade image file**: The image file can also be encrypted to prevent eavesdroppers from acquiring the plaintext firmware image.

The Gecko Bootloader uses a proprietary format for its upgrade images, called GBL (Gecko Bootloader file). These files have the file extension ".gbl". See section Gecko Bootloader File Format for more details.

On Series 1 devices, the Gecko Bootloader has a two-stage design, first stage and main stage, where a minimal first stage bootloader is used to upgrade the main bootloader. The first stage bootloader only contains functionality to read from and write to fixed addresses in internal flash. To perform a main bootloader upgrade, the running main bootloader verifies the integrity and authenticity of the bootloader upgrade image file. The running main bootloader then writes the upgrade image to a fixed location in internal flash and issues a reboot into the first stage bootloader. The first stage bootloader verifies the integrity of the main bootloader firmware upgrade image, by computing a CRC32 checksum before copying the upgrade image to the main bootloader location.

On Series 2 devices, the Gecko bootloader consists only of the main stage bootloader. The main bootloader is upgradable through the Secure Engine. The Secure Engine may be hardware-based, or virtual (software). If hardware-based, the implementation may be either with or without Secure Vault. Throughout this document, the following conventions will be used.

- HSE - Hardware Secure Engine, either with or without Secure Vault if not specified
- VSE - Virtual Secure Engine
- SE - Secure Engine (either HSE or VSE, in general)

The Secure Engine provides functionality to install an image to address 0x0 in internal flash, by copying from a configurable location in internal flash. This makes it possible to have a 2-stage design, where the main bootloader is not present. However, the presence of a main bootloader is assumed throughout this document.

To perform a main bootloader upgrade, the running main bootloader verifies the integrity and authenticity of the bootloader upgrade image file. The running main bootloader then writes the upgrade image to the upgrade location in flash and requests that the Secure Engine installs it. On some devices, the Secure Engine is also capable of verifying the authenticity of the main bootloader update image against a root of trust. The Secure Engine itself is upgradable using the same mechanism. See Gecko Bootloader Operation - Secure Engine Upgrade for more details.

The main bootloader consists of a common core, drivers, and a set of components that give the bootloader specific capabilities. The common bootloader core is now provided as a full-source delivery, as opposed to previously being a

Copyright © 2025 Silicon Laboratories. All rights reserved.

combination of compiled libraries and source code while the components continue to be delivered as source code. The common bootloader core contains functionality to parse GBL files and flash their contents to the device.

The Gecko Bootloader can be configured to perform firmware upgrades in standalone mode (also called a standalone bootloader) or in application mode (also called an application bootloader), depending on the component configuration. Components can be installed in and configured through the Simplicity Studio IDE.

A standalone bootloader uses a communications channel to get a firmware upgrade image. NCP (network co-processor) devices always use standalone bootloaders. Standalone bootloaders perform firmware image upgrades in a single-stage process that allows the application image to be placed into flash memory, overwriting the existing application image, without the participation of the application itself. In general, the only time that the application interacts with a standalone bootloader is when it requests to reboot into the bootloader. Once the bootloader is running, it receives packets containing the firmware upgrade image over-the-air using Bluetooth or by a physical connection such as UART or SPI. To function as a standalone bootloader with a physical connection, a component providing a communication interface such as UART or SPI must be configured.

An application bootloader relies on the application to acquire the firmware upgrade image. The application bootloader performs a firmware image upgrade by writing the firmware upgrade image to a region of flash memory referred to as the download space. The application transfers the firmware upgrade image to the download space in any way that is convenient (UART, over-the-air, Ethernet, USB, and so on). The download space is either an external memory device such as an EEPROM or dataflash or a section of the device's internal flash. The Gecko Bootloader can partition the download space into multiple storage slots and store multiple firmware upgrade images simultaneously. To function as an application bootloader, a component providing a bootloader storage implementation has to be configured.

Silicon Labs provides example bootloaders that come with a preconfigured set of installed components for configuration in either standalone or application mode. See section Configuring the Gecko Bootloader. The Silicon Labs Gecko SDK Suite also includes precompiled bootloader images for several different EFR32 devices. As of this writing, the images shown in the following table are provided.

> **Note**: The bootloader security features are not enabled in these precompiled images.

Table Prebuilt Bootloader Images

Use	Wireless Stack	Image Name	Mode	Interface
SoC	EmberZNet PRO	SPI Flash Storage Bootloader	Application	SPI Serial Flash
SoC	Bluetooth	Bluetooth In-Place OTA DFU Bootloader	Application	OTA/internal flash
NCP	EmberZNet PRO	UART XMODEM Bootloader	Standalone	UART (EZSP)
NCP	Bluetooth	BGAPI UART DFU Bootloader	Standalone	UART (BGAPI)

Note that on devices with a dedicated bootloader area (EFR32xG12 and later Series 1 devices), if the device is configured to boot to the bootloader area (that is, if bit 1 of the Config Lock Word 0 CLW0[1] is set), an image always must be present in the bootloader area. The device is factory-programmed with a dummy bootloader that simply jumps directly to the application in main flash. This means that when flashing a bootloader to a device with a dedicated bootloader area, this dummy bootloader is replaced. If later during development using the bootloader is no longer desired, CLW0[1] must be cleared or the dummy bootloader needs to be re-flashed. Platform-specific prebuilt dummy bootloader images are located in *./platform/bootloader/util/bin/*. Note that since the dummy bootloader only consists of a few instructions and doesn't pad out the remainder of the bootloader area, only the first flash page (where the first-stage bootloader resides) is overwritten, so the main stage bootloader would likely remain intact after programming the dummy bootloader. If desired, the rest of the flash pages in the bootloader area can then be erased separately.

On devices that do not have a dedicated bootloader area (EFR32xG1 and EFR32 Series 2), a dummy bootloader is not needed.

The following sections provide an overview of the Gecko Bootloader common core, drivers, and components. For details, including details on error codes and conditions, see the Gecko Bootloader API Reference, shipped with the SDK in the platform/bootloader/documentation folder.

Copyright © 2025 Silicon Laboratories. All rights reserved.

The bootloader area can be fully erased using the `commander device pageerase --region @bootloader` command with Simplicity Commander. In this state, the device will not boot until CLW0[1] is cleared or the dummy bootloader is flashed. For more information on how to use Simplicity Commander with Gecko bootloader, see section Simplicity Commander and the Gecko Bootloader.

Core

The bootloader core contains the bootloader's main functions. It also contains functionality to write to the internal flash, an image parser to parse and act upon the contents of GBL upgrade files, and functionality to boot the application in main flash.

The image parser can also optionally support the legacy Ember Bootloader (EBL) file format, but none of the security features offered by the Gecko Bootloader are supported if support for legacy EBL files is enabled.

A version of the GBL image parser without support for encrypted upgrade images is also available. This version can be used in flash space constrained bootloader applications where encryption of the upgrade image is not required.

Shared Memory

To exchange information between the bootloader and application, a section of SRAM is used. The contents of SRAM are preserved through a software reset, making the SRAM suitable as a communication channel between bootloader and application.

The shared memory has a size of 4 bytes, and is located at the first address of SRAM, 0x20000000. It is used to store a single word containing the reason for a reset. The structure of the reset cause word is defined in the Reset Information part of the Application Interface, in the file **btl_reset_info.h**, as 16 bits containing the reason, and 16 bits of signature indicating if the word is valid or not. If the signature reads 0xF00F, the reset reason is valid.

All 16-bit reset reasons used by Silicon Labs have the most significant bit set to zero. If custom reset reasons are desired, it is recommended to set the most significant bit to avoid conflicting definitions.

In addition to the reset causes defined in the Reset Information documentation, the bootloader will enter firmware upgrade mode if the shared memory contains the value 0x00000001. This value is supported for compatibility with certain legacy Bluetooth applications.

Drivers

Different applications of firmware upgrade require different hardware drivers for use by the other components of the bootloader.

Driver modules include:

- Delay: Simple delay routines for use with components that require small delays or timeouts.
- SPI: Simple, blocking SPI master implementation for communication with external devices such as SPI flashes.
- SPI Slave: Flexible SPI Slave driver implementation for use in communication components implementing SPI protocols. This driver supports both blocking and non-blocking operation, with DMA (Direct Memory Access) backing the background transfers to support non- blocking operation.
- UART: Flexible serial UART driver implementation for use in communication components implementing UART protocols. This driver supports both blocking and non-blocking operation, with DMA backing the background transfers to support non-blocking operation. Additionally, support for hardware flow control (RTS/CTS) is included.

Components

All parts of the bootloader that are either optional or that may be exchanged for different configurations are implemented as components. Each component may have a configuration header file, and one or more implementations. Components include:

- Communication
  - UART: XMODEM
  - UART: BGAPI
  - SPI: EZSP

Copyright © 2025 Silicon Laboratories. All rights reserved.

    - Bluetooth: AppLoader
- Compression
- Debug
- GPIO Activation
- Security
- Storage
    - Internal flash
    - External SPI flash

Communication

The Communication components provide an interface for implementing communication with a host device, such as a computer or a micro- controller. Several components implement the communication interface, using different transports and protocols.

- BGAPI UART DFU: By enabling the BGAPI communication component, the bootloader communication interface implements the UART DFU protocol using BGAPI commands. See *AN1053: Bluetooth® Device Firmware Update over UART for EFR32xG1 and BGM11x Series Products* for more information about this legacy bootloader.
- Bluetooth:AppLoader: By enabling the Bluetooth AppLoader communication component, the bootloader communication interface implements over-the-air device firmware upgrade functionality using Bluetooth. See Using the Gecko Bootloader with the Silicon Labs Bluetooth Applications for more information.
- EZSP-SPI: By enabling the EZSP-SPI communication component, the bootloader communication interface implements the EZSP protocol over SPI. This component makes the bootloader compatible with the legacy ezsp-spi-bootloader that was previously released with the EmberZNet wireless stack. See *AN760: Using the Ember Standalone Bootloader* for more information about legacy Ember standalone bootloaders.
- UART XMODEM: By enabling the UART XMODEM communication component, the bootloader communication interface implements the XMODEM-CRC protocol over UART. This component makes the bootloader compatible with the legacy serial-uart-bootloader that was previously released with the EmberZNet wireless stack. See *AN760: Using the Ember Standalone Bootloader* for more information about legacy Ember standalone bootloaders.

Compression

The Compression components provide capability for the bootloader GBL file parser to handle compressed GBL upgrade images. Each compression component provides support for one (de)compression algorithm. At the time of writing, decompression of data compressed with the LZ4 and LZMA algorithms is supported, through the *GBL Compression (LZ4)* and *GBL Compression (LZMA)* components.

Debug

This component provides the bootloader with support for debugging output. If the component is configured to enable debug prints, short debug messages will be printed over Serial Wire Output (SWO), which can be accessed in multiple ways, including using Simplicity Commander, and by connecting to port 4900 of the Wireless Starter Kit TCP/IP interface.

To turn on debug prints, enable the Debug component and select **Debug prints**. Select **Debug asserts** to enable assertions in the source code.

On Series 1 devices, also select the GPIO peripheral in the Pin Tool.

GPIO Activation

This component provides functionality to enter firmware upgrade mode automatically after reset if a GPIO pin is active during boot. The GPIO pin location and polarity are configurable.

- GPIO: By enabling the GPIO activation component, the firmware upgrade mode can be activated by the push buttons.
- EZSP GPIO: The EZSP communication protocol over SPI can be used together with this component. By enabling the EZSP GPIO component, the firmware upgrade mode can be entered by activating the *nWake* pin.

Security

Security components provide implementations of cryptographic operations as well as functionality to compute checksums and to read cryptographic keys from manufacturing tokens.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Modules include:

- AES: AES decryption functionality
- CRC16: CRC16 functionality
- CRC32: CRC32 functionality
- ECDSA: ECDSA signature verification functionality
- SHA-256: SHA-256 digest functionality

Storage

These components provide the bootloader with multiple storage options for SoCs. All storage implementations have to provide an API to access image files to be upgraded. This API is based on the concept of dividing the download space into storage slots, where each slot has a predefined size and location in memory and can be used to store a single upgrade image. Some storage implementations also support a raw storage API to access the underlying storage medium. This can be used by applications to store other data in parts of the storage medium that are not used for storing firmware upgrade images. Implementations include:

- **Internal Flash**: The internal flash storage implementation uses the internal flash of the device for upgrade image storage. Note that this storage area is only a download space and is separate from the portion of internal flash used to hold the active application code.
- **SPI Flash**: Two components are available for SPI Flash storage implementation.
    1. **SPI Flash Storage**: The SPI flash storage implementation supports a variety of SPI flash parts. The subset of devices supported can be configured in this component in Gecko Bootloader. (The default configuration if no devices are selected is to include drivers for all supported parts.) Including support for multiple devices requires more flash space in the bootloader. The SPI flash storage implementation does not support any write protection functionality. Supported SPI flash parts are shown in Table 1.2.
    2. **SPI Flash Storage SFDP**: The SPI Flash storage SFDP implementation supports all the SPI flash parts that support SFDP (Serial Flash Description Parameter) Standard from JEDEC. The SPI Flash type is detected automatically by querying the SFDP Parameter table present within the flash memory. All the properties of the SPI Flash are accessed from this parameter table. This component is not configurable since the flash is automatically detected. Any flash that supports the SFDP standard can be used with the Gecko Bootloader. However, performance delays might occur when compared to using the **SPI Flash Storage** component.

> **Note**: Low power devices are recommended for battery-operated applications. Use of the other listed devices will decrease battery life due to higher quiescent current, but this can be mitigated with external shutdown FET circuitry, if desired.

Table Supported Serial Dataflash/EEPROM External Memory Parts

Manufacturer Part Number	Size (kB)	Quiescent Current (µA Typical)*
Macronix MX25R8035F (low power)	1024	0.007
Macronix MX25R6435SF (low power)	8192	0.007
Macronix MX25R3235F (low power)	4096	0.007
Spansion S25FL208K	1024	15
Winbond W25X20BVSNIG (W25X20CVSNJG for high-temperature support)	256	1
Winbond W25Q80BVSNIG (W25Q80BVSNJG for high-temperature support)	1024	1
Macronix MX25L2006EM1I-12G (MX25L2006EM1R-12G for high-temperature support)	256	2
Macronix MX25L4006E	512	2
Macronix MX25L8006EM1I-12G (MX25L8006EM1R-12G for high-temperature support)	1024	2

Copyright © 2025 Silicon Laboratories. All rights reserved.

Manufacturer Part Number	Size (kB)	Quiescent Current (µA Typical)*
Macronix MX25L1606E	2048	2
Macronix MX25U1635E (2V)	2048	2
Atmel/Adesto AT25DF041A	512	15
Atmel/Adesto AT25DF081A	1024	5
Atmel/Adesto AT25SF041	512	2
Micron (Numonyx) M25P20	256	1
Micron (Numonyx) M25P40	512	1
Micron (Numonyx) M25P80	1024	1
Micron (Numonyx) M25P16	2048	1
ISSI IS25LQ025B	32	8
ISSI IS25LQ512B	64	8
ISSI IS25LQ010B	126	8
ISSI IS25LQ020B	256	8
ISSI IS25LQ040B	512	8

* Quiescent current values are as of December 2017; check the latest part specifications for any changes.

Compatibility

1. Silicon Labs recommends building the Gecko Bootloader and the application using the same GSDK. This implies that, if the Gecko Bootloader has been built using GSDK v4.0, it is recommended that the application also be built using GSDK v4.0.
2. Backward compatibility is supported wherein the Gecko Bootloader is built using a previous SDK version and the application is built using a newer SDK version.
3. In general, Silicon Labs does not recommend building the Gecko Bootloader using a newer SDK version with the application built using an older SDK version.

The below table attempts to explain the above-mentioned guidelines:

SDK version X < SDK version Y	Application built using SDK version X	Application built using SDK version Y
Gecko Bootloader built using SDK version X	Recommended	Compatible
Gecko Bootloader built using SDK version Y	Not recommended	Recommended

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader File Format

Gecko Bootloader File Format

The GBL file format is used by the Gecko Bootloader. File formats described in this section are generated by Simplicity Commander commands. For more information, see Simplicity Commander Reference Guide.

File Structures

The GBL file format is composed of several tags that indicate a format of the subsequent data and the length of the entire tag. The format of a tag is as follows:

Tag ID	Tag Length	Tag Payload
4 bytes	4 bytes	Variable (according to tag length)

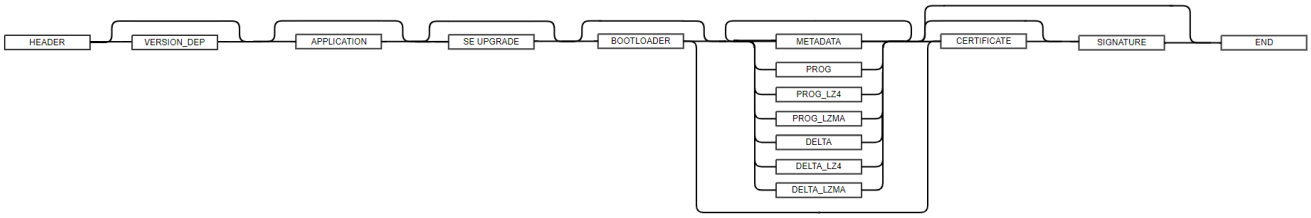
Plaintext Tag Description

Tag Name	ID	Description
GBL Header Tag	0x03A617EB	This must be the first tag in the file. The header tag contains the version number of the GBL file specification, and flags indicating the type of GBL file - whether it is signed or encrypted.
GBL Version Dependency Tag	0x76A617EB	This optional tag contains encoded version dependencies that the software currently running on the device must satisfy before an upgrade can be attempted. Only available on Series 2 devices.
GBL Application Info Tag	0xF40A0AF4	This tag contains information about the application update image that is contained in this GBL file
GBL SE Upgrade Tag	0x5EA617EB	This tag contains a complete encrypted Secure Element update image. Only applicable on Series 2 devices.
GBL Bootloader Tag	0xF50909F5	This tag contains a complete bootloader update image.
GBL Program Data Tag	0xFE0101FE / 0xFD0303FD	This tag contains information about what application data to program at a specific address into the main flash memory.
GBL Delta Tag	0xF80A0AF8UL	This tag contains the information about the delta patch that should be used to create the new app.
GBL Program LZ4 Compressed Data Tag	0xFD0505FD	This tag contains LZ4 compressed information about what application data to program at a specific address into the main flash memory.
GBL Delta LZ4 Compressed Data Tag	0xF80B0BF8UL	This tag contains LZ4 compressed information about the delta patch that should be used to create the new app.
GBL Program LZMA Compressed Data Tag	0xFD0707FD	This tag contains LZMA compressed information about what application data to program at a specific address into the main flash memory.
GBL Delta LZMA Compressed Data Tag	0xF80C0CF8UL	This tag contains LZMA compressed information about the delta patch that should be used to create the new app.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Tag Name	ID	Description
GBL Metadata Tag	0xF60808F6	This tag contains metadata that the bootloader does not parse but can be returned to the application through a callback.
GBL Certificate Tag	0xF30B0BF3	This tag contains a certificate that will be used to verify the authenticity of the GBL file.
GBL Signature Tag	0xF70A0AF7	This tag contains the ECDSA-P256 signature of all preceding data in the file.
GBL End Tag	0xFC0404FC	This tag indicates the end of the GBL file. It contains a 32-bit CRC for the entire file as an integrity check. The CRC is a non-cryptographic check. This must be the last tag.

The allowed sequence of GBL tags in a GBL file is shown in the following figure.

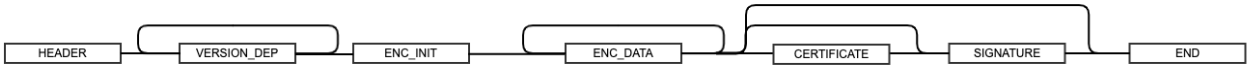


Encrypted Tag Descriptions

The encrypted GBL file format is like the unencrypted version. It introduces several new tags.

Tag Name	ID	Description
GBL Header Tag	0x03A617EB	The GBL header is the same as for a plaintext GBL file, but the flag indicating that the GBL file is encrypted must be set.
GBL Encryption Init Header	0xFA0606FA	This contains information about the image encryption such as the Nonce and the amount of encrypted data.
GBL Encrypted Program Data	0xF90707F9	This contains an encrypted payload containing a plaintext GBL tag, one of Application Info, Bootloader, Metadata or Program Data. The data is encrypted using AES-CTR-128.

The allowed sequence of GBL tags in an encrypted GBL file is shown in the following figure.



GBL Tag Data Structures and Definitions

GBL Tag Header

```
typedef struct {
uint32_t  tagId;  // Tag ID representing the type of tag (GBL Header Tag, GBL Bootloader Tag, etc.).
uint32_t  length; // Length of the subsequent tag data in bytes.
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
} GblTagHeader_t;
```

GBL Header Tag

```
typedef struct {
GblTagHeader_t header;  // Tag header.
uint32_t      version; // Version of the GBL file format specification. E.g. 0×03000000.
uint32_t      type;    // Flags indicating whether the GBL file is encrypted and/or signed.
// See definitions below.
} GblHeader_t;
```

```
// GBL types
#define GBL_TYPE_NONE               0x00000000UL
#define GBL_TYPE_ENCRYPTION_AESCCM  0x00000001UL
#define GBL_TYPE_SIGNATURE_ECDSA    0x00000100UL
```

GBL Version Dependency Tag

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct {
uint8_t  imageType;  // Type of image (application, bootloader, SE)
uint8_t  statement;  // Encoded dependency statement (ex. appVersion > (0).1.2.3)
uint16_t reserved;   // Reserved
uint32_t version;    // The version number used in the statement (ex. (0).1.2.3)
} VersionDependency_t;
// Image types
#define GBL_VERSION_DEPENDENCY_TYPE_APPLICATION          0x01U
#define GBL_VERSION_DEPENDENCY_TYPE_BOOTLOADER           0x02U
#define GBL_VERSION_DEPENDENCY_TYPE_SE                   0x03U
// Operator encoding
#define GBL_VERSION_DEPENDENCY_OPERATOR_MASK             0x0FU
#define GBL_VERSION_DEPENDENCY_OPERATOR_SHIFT            0x00U
#define GBL_VERSION_DEPENDENCY_OPERATOR_TYPE_MASK        0x0EU
#define GBL_VERSION_DEPENDENCY_OPERATOR_NEGATOR_BIT_MASK 0x01U
#define GBL_VERSION_DEPENDENCY_OPERATOR_LT               0x00U
#define GBL_VERSION_DEPENDENCY_OPERATOR_LEQ              0x02U
#define GBL_VERSION_DEPENDENCY_OPERATOR_EQ               0x04U
#define GBL_VERSION_DEPENDENCY_OPERATOR_GEQ              0x06U
#define GBL_VERSION_DEPENDENCY_OPERATOR_GT               0x08U
// Connective encoding
#define GBL_VERSION_DEPENDENCY_CONNECTIVE_MASK           0xF0U
#define GBL_VERSION_DEPENDENCY_CONNECTIVE_SHIFT          0x04U
#define GBL_VERSION_DEPENDENCY_CONNECTIVE_TYPE_MASK      0x0EU
#define GBL_VERSION_DEPENDENCY_CONNECTIVE_NEGATOR_BIT_MASK 0x01U
#define GBL_VERSION_DEPENDENCY_CONNECTIVE_AND            0x00U
// SE version mask for ignoring the compatibility byte when comparing versions
#define GBL_VERSION_DEPENDENCY_SE_VERSION_MASK           0x00FFFFFFUL
```

GBL Application Info Tag

```
typedef struct {
GblTagHeader_t    header;  // Tag header.
ApplicationData_t appInfo; // Application information structure. See definition below.
} GblApplication_t;
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct ApplicationData {
uint32_t type;          // Bitfield representing the type of application.
// See definitions below for possible values.
uint32_t version;       // Version number for this application (customer-defined).
uint32_t capabilities;  // Bitfield representing the capabilities of this application.
uint8_t  productId[16]; // Unique ID (UUID or GUID) for the product this application is built for.
} ApplicationData_t;
```

```
// Application types
#define APPLICATION_TYPE_ZIGBEE         (1UL << 0UL)
#define APPLICATION_TYPE_THREAD         (1UL << 1UL)
#define APPLICATION_TYPE_FLEX           (1UL << 2UL)
#define APPLICATION_TYPE_BLUETOOTH      (1UL << 3UL)
#define APPLICATION_TYPE_MCU            (1UL << 4UL)
#define APPLICATION_TYPE_BLUETOOTH_APP  (1UL << 5UL)
#define APPLICATION_TYPE_BOOTLOADER     (1UL << 6UL)
#define APPLICATION_TYPE_ZWAVE          (1UL << 7UL)
```

GBL SE Upgrade Tag

```
typedef struct {
GblTagHeader_t header;  // Tag header.
uint32_t      blobSize; // Size of the SE upgrade data blob.
uint32_t      version;  // Version of the SE image.
uint8_t       data[];   // Array of data containing the SE upgrade blob.
} GblSeUpgrade_t;
```

GBL Bootloader Tag

```
typedef struct {
GblTagHeader_t header;          // Tag header.
uint32_t      bootloaderVersion; // Version number of the bootloader.
uint32_t      address;          // Base address of the bootloader image.
uint8_t       data[];           // Array of data containing the bootloader upgrade image.
} GblBootloader_t;
```

GBL Program Data Tag

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct {
GblTagHeader_t header;          // Tag header.
uint32_t       flashStartAddress; // Address at which to start flashing data.
uint8_t        data[];          // Array of data to be flashed
// (compressed data in the LZ4 and LZMA variants of the tag).
} GblProg_t;
```

GBL Metadata Tag

```
typedef struct {
GblTagHeader_t header;    // Tag header.
uint8_t        metadata[]; // Array containing the metadata.
} GblMetadata_t;
```

GBL Certificate Tag

```
typedef struct {
GblTagHeader_t          header;     // Tag header.
ApplicationCertificate_t certificate; // Certificate used to verify the GBL file. See definition below.
} GblCertificateEcdsaP256_t;
typedef struct ApplicationCertificate {
uint8_t  structVersion;    // Version of the certificate structure.
uint8_t  flags[3];         // Reserved flags.
uint8_t  key[64];          // Public key used to verify the GBL file.
uint32_t version;          // Version number of this certificate.
uint8_t  signature[64];    // The signature of the certificate itself (not the GBL file).
} ApplicationCertificate_t;
```

GBL Signature Tag

```
typedef struct {
GblTagHeader_t header; // Tag header.
uint8_t        r[32];  // The r value of the ECDSA signature.
uint8_t        s[32];  // The s value of the ECDSA signature.
} GblSignatureEcdsaP256_t;
```

GBL End Tag

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct {
GblTagHeader_t header; // Tag header.
uint32_t      gblCrc; // CRC32 checksum of the entire GBL file.
} GblEnd_t;
```

GBL Encryption Init Header Tag

```
typedef struct {
GblTagHeader_t header;    // Tag header.
uint32_t      msgLen;    // Length of the ciphertext in bytes.
uint8_t       nonce[12]; // Random AES-CTR nonce.
} GblEncryptionInitAesCcm_t;
```

GBL Encrypted Data Tag

```
typedef struct {
GblTagHeader_t header;           // Tag header.
uint8_t       encryptedGblData[]; // Array containing the ciphertext (encrypted GBL data).
// Note that the corresponding plaintext must contain one or
// more complete GBL tags. Put differently: A single plaintext GBL
// tag cannot be split across multiple encrypted tags.
} GblEncryptionData_t;
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader Operation Application Upgrade

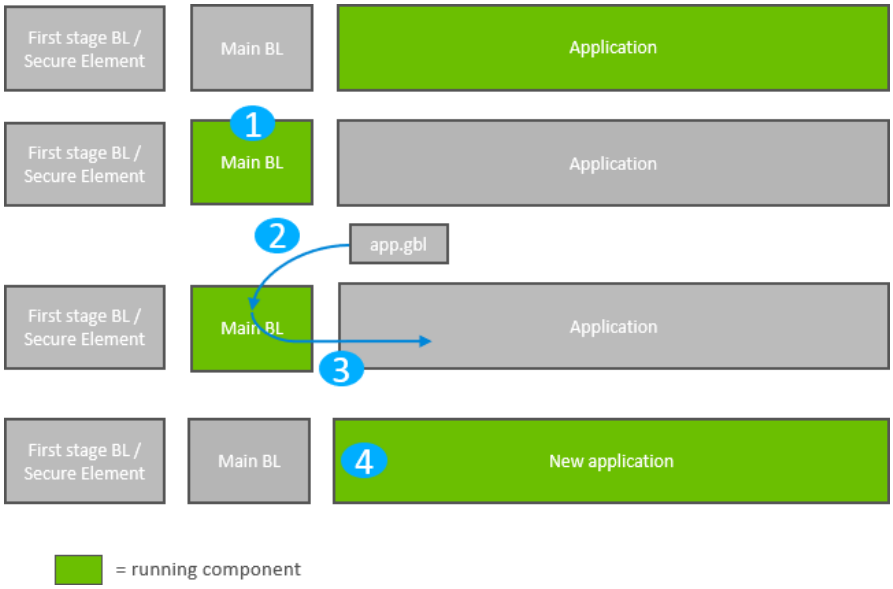
Gecko Bootloader Operation - Application Upgrade

This section summarizes Gecko Bootloader operation for updating application firmware, first if the Gecko Bootloader is configured in standalone mode and then if it is configured in application mode. Section Gecko Bootloader Operation - Bootloader Upgrade provides the same information for updating the bootloader firmware.

The figures that illustrate Gecko Bootloader operation in this section do not provide information about the bootloader memory layouts for different devices. For more details refer to the section "Memory Space for Bootloading" in *UG103.6: Bootloader Fundamentals*.

Standalone Bootloader Operation

Standalone bootloader operation is illustrated in the following figure:



1. The device reboots into the bootloader.
2. A GBL file containing an application image is transmitted from the host to the device. If image encryption is enabled in the main stage bootloader and the image is encrypted, decryption is performed during the process of receiving and parsing the GBL file.
3. The bootloader applies the application upgrade from the GBL upgrade file on-the-fly. If image authentication is enabled in the main stage bootloader and the GBL file contains a signature, the authenticity of the image is verified before completing the process.
4. The device boots into the application. Application upgrade is complete.

Rebooting Into the Bootloader

The Gecko Bootloader supports multiple mechanisms for triggering the bootloader. If the **GPIO activation** component is installed, the host device can keep this pin low/high (depending on configuration) through reset to make the device enter the bootloader. The bootloader can also be entered through software. The `bootloader_rebootAndInstall` API first signals to the bootloader that it should enter firmware upgrade mode by writing a command to the shared memory location at the

Copyright © 2025 Silicon Laboratories. All rights reserved.

bottom of SRAM, and then performs a software reset. If the bootloader finds the correct command in shared memory upon boot, it will enter firmware upgrade mode instead of booting the existing application.

Downloading and Applying a GBL Upgrade File

When the bootloader enters firmware upgrade mode, it enters a receive loop waiting for data from the host device. The specifics of the receive loop depend on the protocol. Received packets are passed to the image parser, a state machine that parses the data and returns a callback containing any data that should be acted upon. The bootloader core implements this callback and flashes the data to internal flash at the address specified. If GBL file authentication or encryption is enabled, the image parser will enforce this, and abort the image upgrade if the authentication fails.

The bootloader prevents a newly uploaded image from being bootable by holding back parts of the application vector table until the GBL file CRC and GBL signature (if required) have been verified.

Booting Into the Application

When an application upgrade is completed, the bootloader triggers a reboot with a message in shared memory at the bottom of SRAM signaling that an application upgrade has been successfully completed. The application can use this reset information to learn that an application upgrade was just performed.

Before jumping to the main application, the bootloader verifies that the application is ready to run. This includes verifying if the Program counter and Stack Pointer are valid. If secure boot is enabled, the bootloader expects a signed application and attempts to validate the signature of the application. In scenarios where secure boot is not enabled, the bootloader attempts to validate if the Application properties pointer points to valid app properties structure in the flash. If a valid app properties struct is found, the bootloader proceeds based on the signature type indicated by the application properties struct or else the bootloader assumes that the Application properties pointer points to the Reset Handler of the application (an app without application properties) and proceeds to boot into the application. In case the verification of the application fails at any stage, the bootloader enters the bootload mode instead of booting into the application.

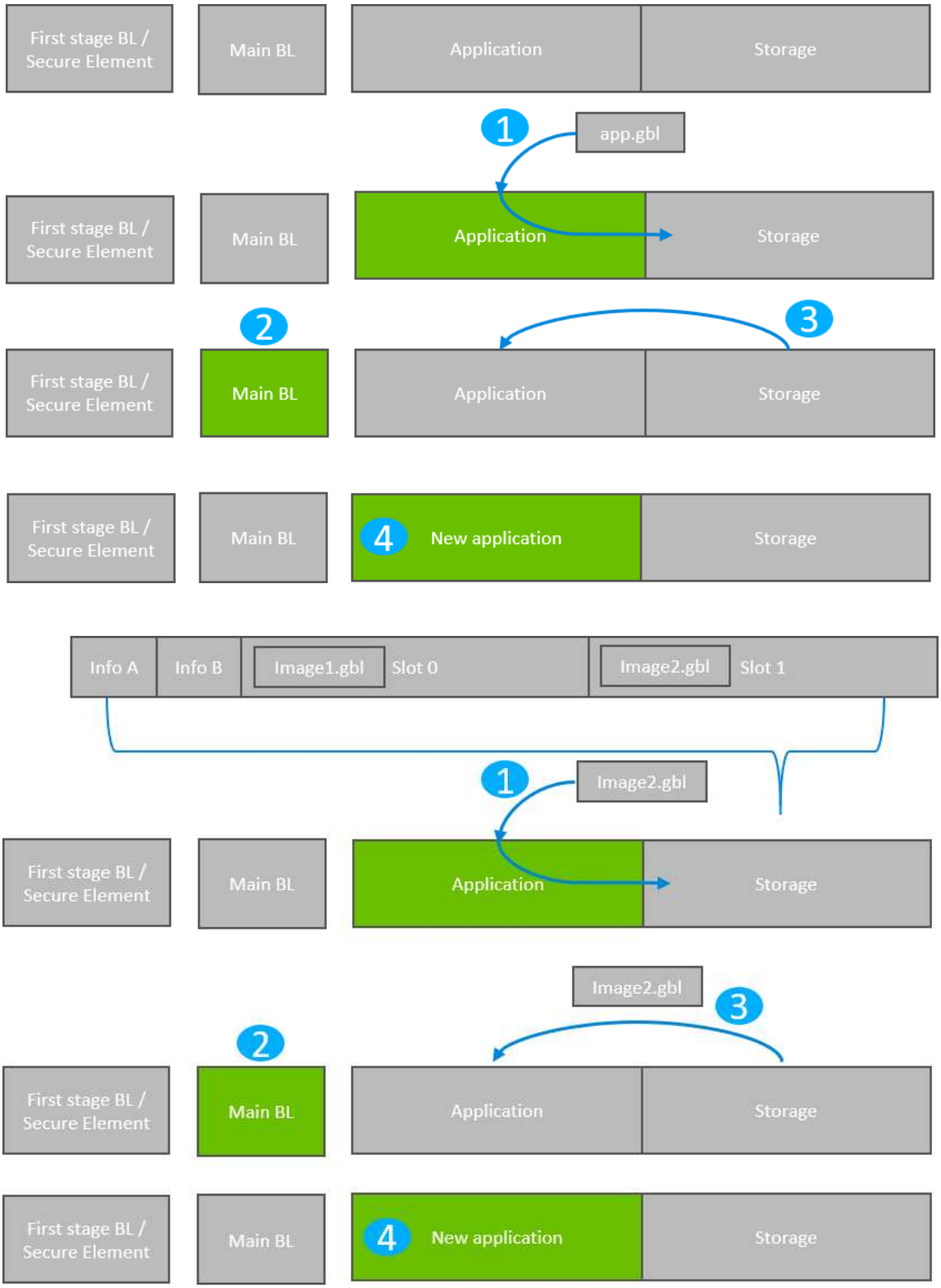
Error Handling

If the application upgrade is interrupted at any time, the device will be without a working application. The bootloader then resets the device, and re-enters firmware upgrade mode. The host device can easily restart the application upgrade process, to try loading the upgrade image again.

Application Bootloader Operation

The following figure illustrates the application bootloader operation both for a single image/single storage slot, and multiple images/ multiple storage slots.

Copyright © 2025 Silicon Laboratories. All rights reserved.



1. A GBL file is downloaded onto the storage medium of the device (internal flash or external dataflash), as described below, and the presence of an upgrade image is indicated.
2. The device reboots into the bootloader, and the bootloader enters firmware upgrade mode.
3. The bootloader applies the application upgrade from the GBL upgrade file.
4. The device boots into the application. Application upgrade is complete.

Downloading and Storing a GBL Upgrade Image File

Copyright © 2025 Silicon Laboratories. All rights reserved.

To prepare for receiving an upgrade image, the application finds an available storage slot, or erases an existing one using `bootloader_eraseStorageSlot`. If the bootloader only supports a single storage slot, a value of 0 should be used for the slot ID.

The application then receives a GBL file using an applicable protocol, such as Ethernet, USB, Zigbee, OpenThread, or Bluetooth, and stores it in the slot by calling `bootloader_writeStorage`.

When download is complete, the application can optionally verify the integrity of the GBL file by calling `bootloader_verifyImage`. This is also done by the bootloader before applying the image but can be done from the application in order to avoid rebooting into the bootloader if the received image was corrupt.

If multiple storage slots are supported, the application should write a bootload list by calling `bootloader_setImageToBootload`. The list is written to the two bootload info pages as shown in Figure The bootload list is a prioritized list of slots indicating the order the bootloader should use when attempting to perform a firmware upgrade. The bootloader attempts to verify the images in these storage slots in sequence and applies the first image to pass verification. If only a single storage slot is supported, the bootloader treats the entire download space as a single storage slot.

Rebooting and Applying a GBL Upgrade File

The bootloader can be entered through software. The `bootloader_rebootAndInstall` API signals to the bootloader that it should enter firmware upgrade mode by writing a command to the shared memory location at the bottom of SRAM, and then performs a software reset. If the bootloader finds the correct command in shared memory upon boot, it enters firmware upgrade mode instead of booting the existing application.

The bootloader iterates over the list of storage slots marked for bootload and attempts to verify the image stored in each. Once it finds a valid GBL upgrade file, firmware upgrade is attempted from this GBL file. If the upgrade fails, the bootloader moves to the next image in the list. If no images pass verification, the bootloader reboots back into the existing application with a message in the shared memory location in SRAM indicating that no good upgrade images were found.

Booting Into the Application

When an application upgrade is completed, the bootloader triggers a reboot with a message in shared memory at the bottom of SRAM signaling that an application upgrade has been successfully completed. The application can use this reset information to learn that an application upgrade was just performed.

Before jumping to the main application, the bootloader verifies that the application is ready to run. This includes verifying if the Program counter and Stack Pointer are valid. If secure boot is enabled, the bootloader expects a signed application and attempts to validate the signature of the application. In scenarios where secure boot is not enabled, the bootloader attempts to validate if the Application properties pointer points to valid app properties structure in the flash. If valid app properties struct is found, the bootloader proceeds based on the signature type indicated by the application properties struct or else the bootloader assumes that the Application properties pointer points to the Reset Handler of the application (an app without application properties) and proceeds to boot into the application. In case the verification of the application fails at any stage, the bootloader enters the bootload mode instead of booting into the application.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader Operation Bootloader Upgrade

Gecko Bootloader Operation - Bootloader Upgrade

Bootloader upgrade functionality is provided by the first stage bootloader on Series 1 devices, or the Secure Engine on Series 2 devices. The Secure Engine itself is also upgradable. For more details, see section Gecko Bootloader Operation - Secure Engine Upgrade. On Series 1 devices, the first stage bootloader is not upgradable.

Requirements for upgrading the main bootloader vary depending on the bootloader configuration:

- Application bootloader with storage: Upgrading the main bootloader requires a single GBL file containing both bootloader and application upgrade images.
- Standalone bootloader with communication interface: Upgrading the bootloader requires two GBL files, one with only the bootloader upgrade image, and one with only the application upgrade image.

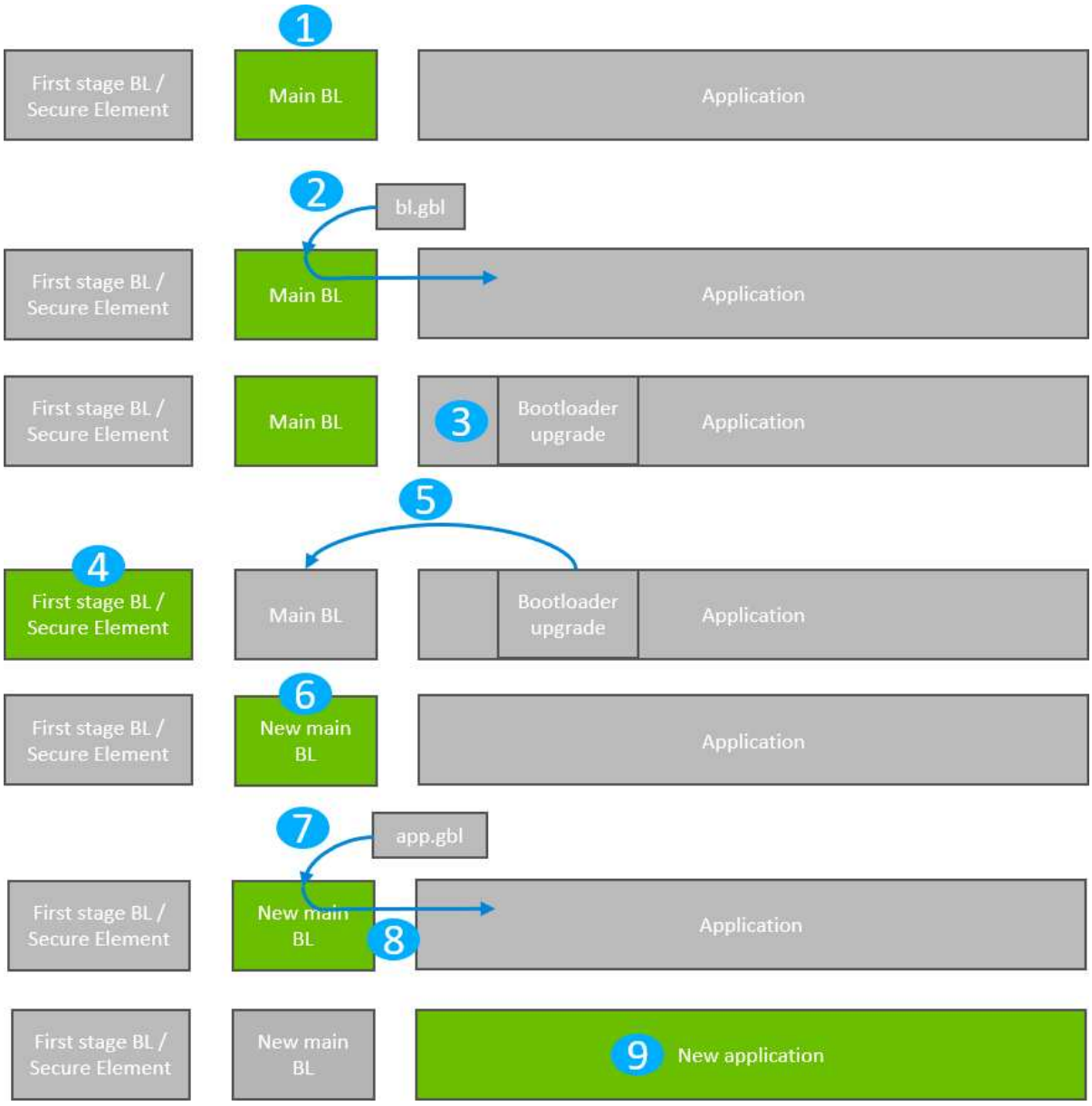
Security of the bootloader upgrade process is provided by signing the GBL file. See section Creating a Signed and Encrypted GBL Upgrade Image File from an Application.

The figures that illustrate Gecko Bootloader operation in this section do not provide information about the bootloader memory layouts for different devices. For more details refer to the “Memory Space for Bootloading” section in *UG103.6: Bootloader Fundamentals*. For convenience, the figures do not distinguish between SE and VSE.

Bootloader Upgrade on Bootloaders with Communication Interface (Standalone Bootloaders)

The process is illustrated in the following figure:

Copyright © 2025 Silicon Laboratories. All rights reserved.



1. The device reboots into the bootloader.
2. A GBL file containing only a bootloader upgrade image is transmitted from the host to the device.
3. The contents of the GBL Bootloader tag are written to the bootloader upgrade location in internal flash, overwriting the existing application.
4. The device reboots into the first stage bootloader / Secure Engine.
5. The first stage bootloader / Secure Engine replaces the main bootloader with the new version found in the bootloader upgrade location.
6. The device boots into the new main bootloader.
7. A GBL file containing only an application image is transmitted from the host to the device.
8. The bootloader applies the application image from the GBL upgrade file on-the-fly.
9. The device boots into the application. Bootloader upgrade is complete.

A bootloader upgrade is started in the same way as an application upgrade.

Downloading and Applying a Bootloader GBL Upgrade File

When the bootloader has entered the receive loop, a GBL upgrade file containing a bootloader upgrade is transmitted to the bootloader. When a packet is received, it is passed to the image parser. The image parser parses the data and returns

Copyright © 2025 Silicon Laboratories. All rights reserved.

bootloader upgrade data in a callback. The bootloader core implements this callback and flashes the data to internal flash at the bootloader upgrade location.

The bootloader prevents a newly uploaded bootloader upgrade image from being interpreted as valid by holding back parts of the bootloader upgrade vector table until the GBL file CRC and GBL signature (if required) have been verified.

When a complete bootloader upgrade image is received, the main bootloader signals the first stage bootloader / Secure Engine that it should enter firmware upgrade mode. On Series 1 devices, this is done by writing a command to the shared memory location at the bottom of SRAM, and then performing a software reset. On Series 2 devices, Secure Engine communication is used to signal that bootloader upgrade is ready to be performed.

On Series 1 devices, the first stage bootloader verifies the CRC on the current main bootloader and verifies the CRC of the bootloader upgrade present in the bootloader upgrade location in internal flash.

- If the CRC in the upgrade image location and the CRC in the current main bootloader are both valid, then the upgrade image will be copied over the main bootloader if the version number of the upgrade is higher than the current main bootloader version.
- If the CRC in the upgrade image location is valid and the CRC in the current main bootloader location is not valid, the upgrade image will be copied over the main bootloader regardless of the version. This is because the version of the main bootloader cannot be relied upon if the main bootloader image is corrupt.
- If the CRC in the upgrade location is not valid, the upgrade will not be applied.

On Series 2 devices, the authenticity of the main bootloader optionally can be verified before applying the bootloader upgrade. See section Setting a Version Number for more information about versioning bootloader images.

Upgrading Bootloaders Without Secure Boot to Bootloaders with Secure Boot

A bootloader without the secure boot feature can be upgraded to a bootloader with the secure boot feature, using the following procedure:

1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the bootloader needs to be higher than the bootloader on the device.
   - Turn on secure boot in Simplicity Studio by going to the **Bootloader Core** component and selecting the **Enable secure boot** option.
   - (Optional) In the **Bootloader Core** component, select the **Require signed firmware upgrade files** option. This means that the Gecko Bootloader will only accept signed GBL files.
2. Generate a public/private Signing Key pair. See section Generating Keys for more information on creating a Signing Key pair.
3. Write the public key generated from the previous step to the device. The public key is stored as a manufacturing token in the device by default. This key can be written by application code running on the device as long as the Lock Bits page is configured to allow flash writes. If the Lock Bits page is locked, it can only be erased by the debugger. Therefore, signing/decryption keys residing in the Lock Bits page cannot be erased from firmware. This means that, for devices in the field, those areas in flash cannot be replaced with new ones. However, the Gecko Bootloader prepared from step 1 can be modified to look for the decryption and signature keys in a different location. Key locations are defined in the bootloader project file `btl_security_tokens.c`.
4. Create a GBL file using the Gecko Bootloader image. The GBL file needs to be signed/unsigned depending on the current configuration of the Gecko Bootloader running on the device. For more details on creating a GBL file, see section Creating a Signed and Encrypted GBL Upgrade Image File from an Application.
5. Upload the GBL file. For more details on the upgrade process, see section Bootloader Upgrade on Bootloaders with Communication Interface (Standalone Bootloaders).

Enabling Secure Boot RTSL on Series 2 Devices

Secure Boot RTSL (Root of Trust and Secure Loader) can be enabled using the following procedure:

1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the Gecko Bootloader needs to be higher than the Gecko Bootloader on the device.
   - Turn on secure boot in Simplicity Studio by going to the **Bootloader Core** component and selecting the **Enable secure boot** option.
   - For EFR32xG21, in the **Bootloader Core** component, disable the **Allow use of public key from manufacturing token storage** option. This means that the Gecko Bootloader will never make use of the public key stored in the last page of the main flash

Copyright © 2025 Silicon Laboratories. All rights reserved.

    - (Optional) In the **Bootloader Core** component, select the **Require signed firmware upgrade files** option. This means that the Gecko Bootloader will only accept signed GBL files.
2. Generate a public/private Signing Key pair. See section Generating Keys for more information on creating a Signing Key pair.
3. Prepare an application that installs the public key generated from step 2 to the Secure Engine One-time Programmable memory. Installing a key in the VSE requires a reset routine. Make sure that the application does not end up in the reset loop. Create an unsigned GBL file from this application and upload it. For more information on installing public keys, see section Creating a Signed and Encrypted GBL Upgrade Image File from an Application.
4. Sign the Gecko Bootloader image generated from step 1 using the private key generated in step 2. See section Signing an Application Image for Secure Boot for more information on signing binaries.
5. Make a custom application that turns on secure boot on the Secure Engine and sign this application binary with the private key generated from step 2. For more details on how to turn on secure boot on the Secure Engine, see Series 2 Secure Boot with RTSL.
6. Create a GBL file using the Gecko Bootloader image from step 4.
7. Create a GBL file using the application from step 5. The GBL file need to be signed if the **Bootloader Core** component option **Require signed firmware upgrade files** was selected in step 1.
8. Upload the GBL file containing the Gecko Bootloader image.
9. Upload the GBL file containing the application.

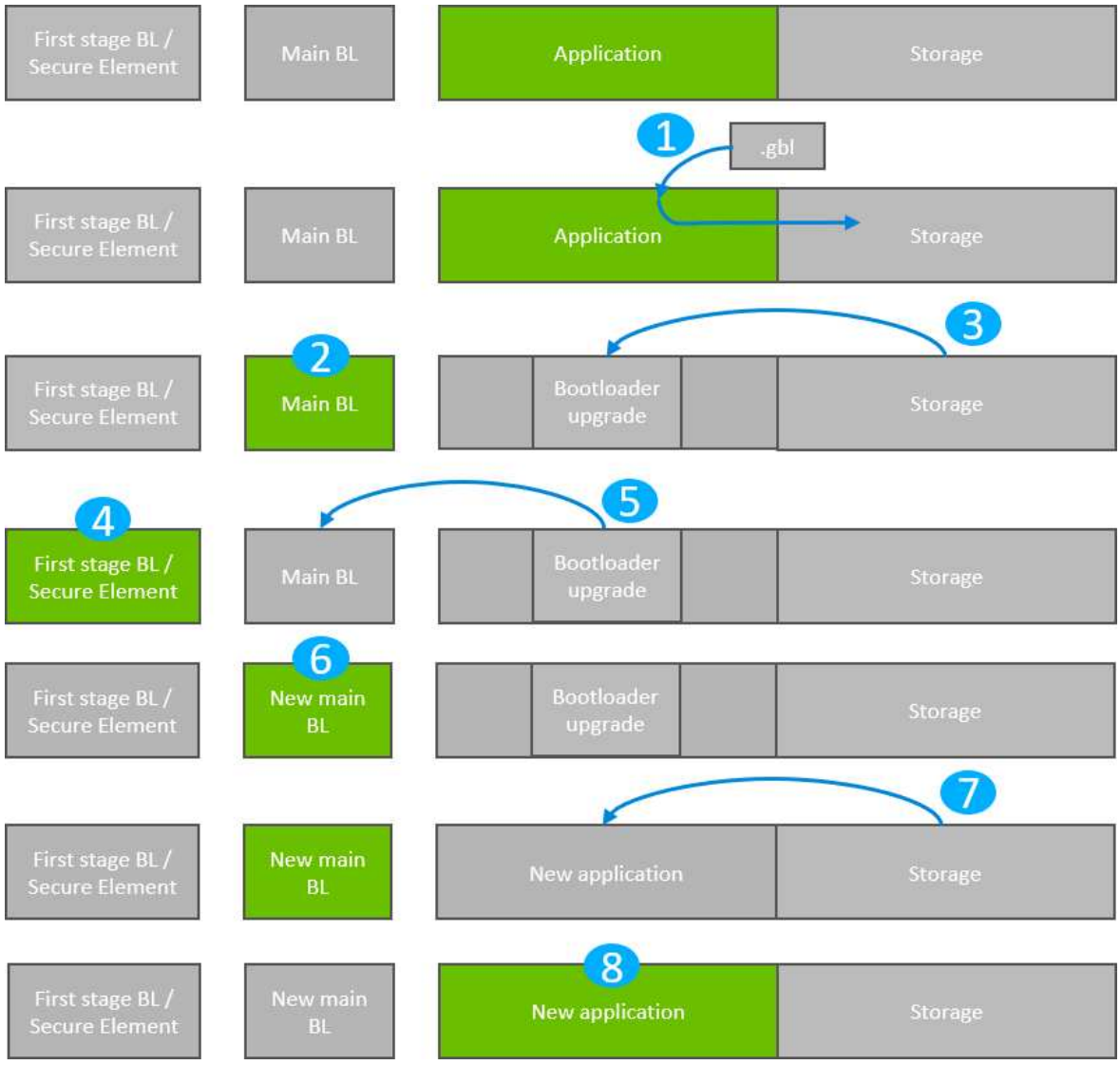
Downloading and Applying an Application GBL Upgrade File

Once the bootloader upgrade is completed, the existing application is rendered invalid, since the bootloader upgrade location overlaps with the application. A GBL upgrade file containing an application upgrade is transmitted to the bootloader. The application upgrade process follows that in section Standalone Bootloader Operation.

Bootloader Upgrade on Application Bootloaders with Storage

The process is illustrated in the following figure.

Copyright © 2025 Silicon Laboratories. All rights reserved.



1. A single GBL file containing both a bootloader upgrade image and an application image is downloaded onto the storage medium of the device (internal flash or external SPI flash).
2. The device reboots into the main bootloader.
3. The main bootloader parses its upgrade image into internal flash at the bootloader upgrade location. Depending on the application size as well as how the “scratch space” is configured for the bootloader on Series 2 devices, the existing application may be overwritten.
4. The main bootloader verifies the integrity of the upgrade image and then resets the device with reset reason BOOTLOADER_RESET_REASON_UPGRADE to apply the upgrade.
5. The device reboots into the first stage bootloader / Secure Engine.
6. The first stage bootloader / Secure Engine replaces the main bootloader with the new version.
7. The device boots into the new main bootloader.
8. The bootloader applies the application image from the GBL upgrade file.
9. The device boots into the application. Bootloader upgrade is complete.

A bootloader upgrade is started in the same way as an Application Upgrade. A single GBL file containing both a bootloader and an application upgrade is written to storage by the application, and the bootloader is entered.

The bootloader iterates over the list of storage slots marked for bootload and attempts to verify the GBL file stored within. Verification returns information about whether the GBL file contains an application, or both a bootloader and an application.

Copyright © 2025 Silicon Laboratories. All rights reserved.

The image parser parses the file. If the GBL file contains a bootloader, the bootloader upgrade data is returned in a callback. The bootloader core implements this callback and flashes the data to internal flash at the bootloader upgrade location.

The bootloader prevents a newly uploaded bootloader upgrade image from being interpreted as valid by holding back parts of the bootloader upgrade vector table until the GBL file CRC and GBL signature (if required) have been verified.

On Series 1 devices, the main bootloader signals the first stage bootloader that it should enter firmware upgrade mode by writing a command to the shared memory location at the bottom of SRAM, and then performing a software reset. On Series 2 devices, Secure Engine communication interface is used to signal the Secure Engine that a bootloader upgrade is ready to be performed.

On Series 1 devices, the first stage bootloader verifies the CRC of the bootloader upgrade present in the bootloader upgrade location in internal flash and copies the bootloader upgrade over the main bootloader if the version number of the upgrade is higher than the version number of the existing main bootloader. On Series 2 devices, the authenticity of the main bootloader optionally can be verified before applying the bootloader upgrade. See section Setting a Version Number for more information about versioning bootloader images.

The new main bootloader is entered, and the images in the list of storage slots marked for bootload are verified. When the image parser parses the slot containing the GBL file with the bootloader + application upgrade, the version number of the bootloader upgrade is equal to the running main bootloader version, so another bootloader upgrade will not be performed. Instead, the application upgrade data are returned in a callback. Bootloading of the new application proceeds as described in section Application Bootloader Operation.

Storage Space Size Configuration

The storage space size must be configured to have enough space to store the upgrade images. Depending on the configuration, the bootloader size can vary. For size requirements of the bootloader, see section Size Requirements for Different Bootloader Configurations for Series 1 Devices.

Upgrading Bootloaders without Secure Boot to Bootloaders with Secure Boot

A bootloader without the secure boot feature can be upgraded to a bootloader with the secure boot feature. The procedure is as follows:

1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the bootloader needs to be higher than the bootloader on the device.
   - Turn on secure boot from the **Bootloader Core** component in Simplicity Studio by selecting the **Enable secure boot** option.
2. Generate a public/private Signing Key pair. See section Generating Keys for more information on creating a Signing Key pair.
3. Write the public key generated from the previous step to the device. The public key is stored as a manufacturing token in the device by default. This key can be written by application code running on the device as long as the Lock Bits page is configured to allow flash writes. If the Lock Bits page is locked, it can only be erased by the debugger. Therefore, signing/decryption keys residing in the Lock Bits page cannot be erased from firmware. This means that, for devices in the field, those areas in flash cannot be replaced with new ones. However, the Gecko Bootloader prepared from step 1 can be modified to look for the decryption and signature keys in a different location. Key locations are defined in the bootloader project file `btl_security_tokens.c` .
4. Prepare a signed application image using the private key generated in step 2. See section Signing an Application Image for Secure Boot for more information on signing an application.
5. Create a GBL file using the Gecko Bootloader image and the signed application image. The GBL file needs to be signed/unsigned depending on the configuration of the Gecko Bootloader running on the device. For more details on creating a GBL file, see section Creating a Signed and Encrypted GBL Upgrade Image File from an Application.
6. Upload the GBL file. For more details on the upgrade process, see section Bootloader Upgrade on Application Bootloaders with Storage.

Enabling Secure Boot RTSL on Series 2 Devices

Secure Boot RTSL can be enabled by using the following procedure:

1. Prepare a Gecko Bootloader image with secure boot enabled. The version of the Gecko Bootloader needs to be higher than the Gecko Bootloader on the device.
   - Turn on secure boot from the **Bootloader Core** component in Simplicity Studio by selecting the **Enable secure boot** option.

Copyright © 2025 Silicon Laboratories. All rights reserved.

- For EFR32xG21, in the **Bootloader Core** component, disable the **Allow use of public key from manufacturing token storage** option. This means that the Gecko Bootloader will never make use of the public key stored in the last page of the main flash.
- (Optional) In the **Bootloader Core** component, select the **Require signed firmware upgrade files** option. This means that the Gecko Bootloader will only accept signed GBL files.

2. Generate a public/private Signing Key pair. See section Generating Keys for more information on creating a Signing Key pair.
3. Prepare an application that installs the public key generated from step 2 to the Secure Engine One-time Programmable memory. Installing a key in VSE requires a reset routine. Make sure that the application does not end up in the reset loop. Create an un- signed GBL file from this application and upload it. For more information on installing public keys, see Series 2 Secure Boot with RTSL. For more details on creating a GBL file, see section Creating a Signed and Encrypted GBL Upgrade Image File from an Application.
4. Sign the Gecko Bootloader image generated from step 1 using the private key generated in step 2. See section Signing an Application Image for Secure Boot for more information on signing binaries.
5. Make a custom application that turns on secure boot on the Secure Engine and sign this application binary with the private key generated from step 2. For more details on how to turn on secure boot on the Secure Engine, see Series 2 Secure Boot with RTSL.
6. Create a GBL file using the Gecko Bootloader image from step 4 and the application from step 5. The GBL file must be signed if the **Bootloader Core** component option **Require signed firmware upgrade files** was selected in step 1. For more details on creating a GBL file, see section Creating a Signed and Encrypted GBL Upgrade Image File from an Application.
7. Upload the GBL file containing the Gecko Bootloader image and the application.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader Operation Secure Engine Upgrade

Gecko Bootloader Operation - Secure Engine Upgrade

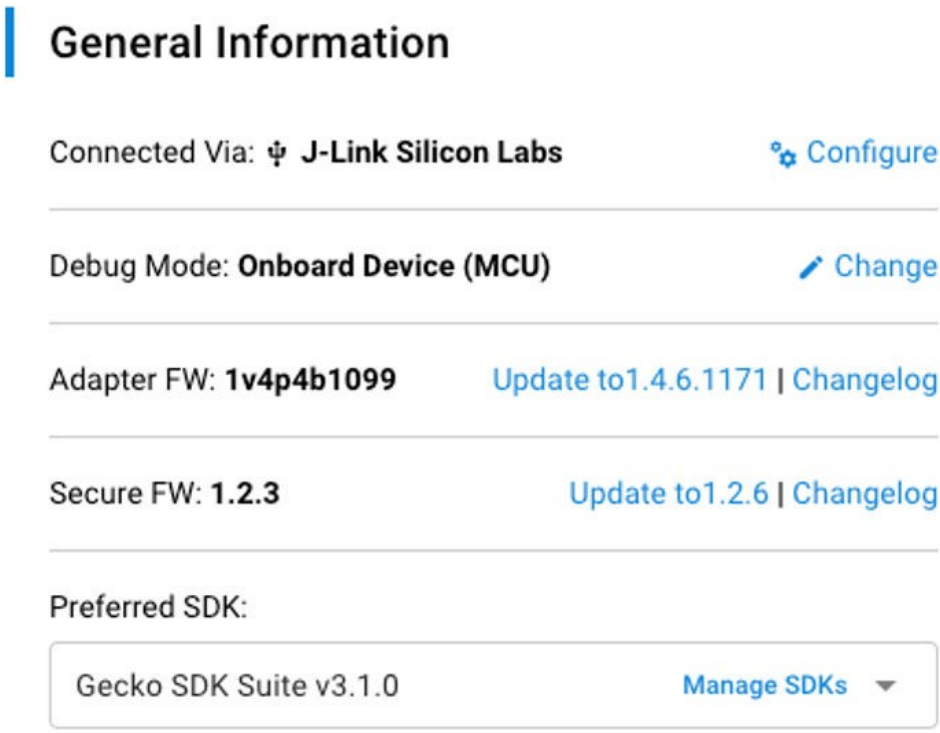
The Secure Engine is upgradable and requirements for upgrading the Secure Engine vary depending on the bootloader configuration:

- **Application bootloader with storage**: Upgrading the Secure Engine requires a single GBL file containing both Secure Engine and application upgrade images.
- **Standalone bootloader with communication interface**: Upgrading the Secure Engine requires two GBL files, one with only the Secure Engine upgrade image, and one with only the application upgrade image and optionally a third image containing only the Main bootloader upgrade.

A bootloader upgrade can also be included in the same GBL file in application mode, or as a third GBL file in standalone mode. The figures that illustrate Gecko Bootloader operation in this section do not provide information about the bootloader memory layouts for different devices. For more details refer to the section “Memory Space for Bootloading” in *UG103.6: Bootloader Fundamentals*.

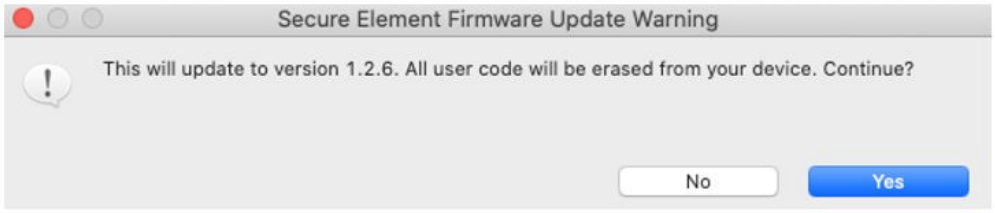
Signed and encrypted Secure Engine upgrade images are provided by Silicon Labs through Simplicity Studio. Upgrade images with the same or lower version number than the running Secure Engine will be ignored.

To download Secure Engine firmware images, connect a Series 2 device and select a preferred SDK. The Secure Firmware **Update to x.x.x** link appears in the Launcher Perspective, as shown in the following figure.



Click **Update to x.x.x** next to Secure FW: x.x.x. A warning dialog box appears. Click **[Yes]** to continue.

Copyright © 2025 Silicon Laboratories. All rights reserved.



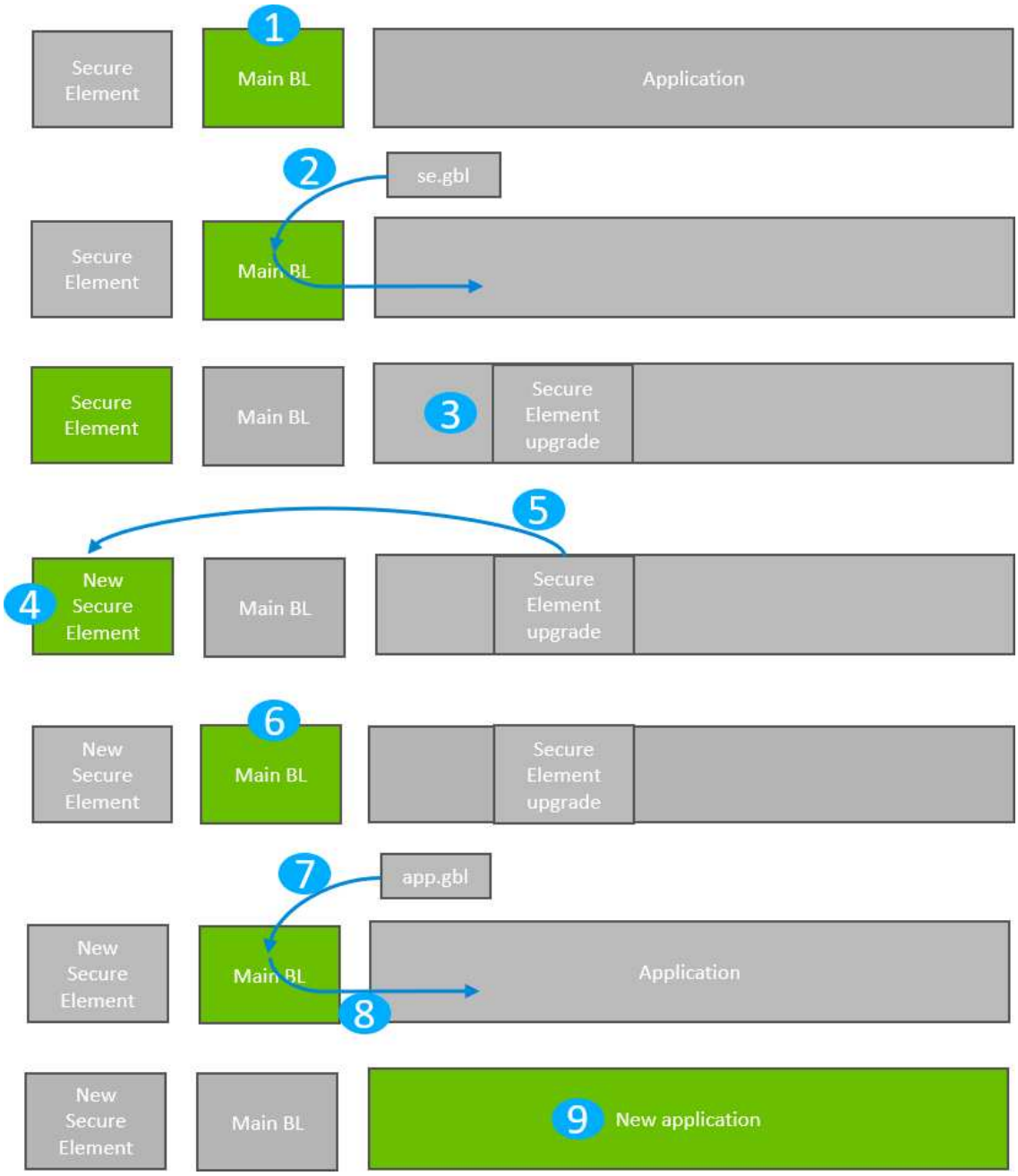
The Launcher Perspective is then updated so that the current Secure Firmware version and link version are the same.

The Secure Engine firmware images can be found in the *util/se_release/public* directory under the Gecko SDK. Simplicity Studio displays the SE firmware version available in the Gecko SDK selected.

Secure Engine Upgrade on Bootloaders with Communication Interface (Standalone Bootloaders)

The process is illustrated in the following figure.

Copyright © 2025 Silicon Laboratories. All rights reserved.



1. The device reboots into the bootloader.
2. A GBL file containing only a Secure Engine upgrade image is transmitted from the host to the device.
3. The contents of the GBL Secure Engine tag are written to the pre-configured upgrade location in internal flash, overwriting the existing application.
4. The device reboots into the Secure Engine.
5. The Secure Engine is replaced by the new version found in the pre-configured upgrade location.
6. The device boots into the main bootloader.
7. A GBL file containing only an application image is transmitted from the host to the device.
8. The bootloader applies the application image from the GBL upgrade on the fly.
9. The device boots into the application. Secure Engine upgrade is complete.

Downloading and Applying a Secure Engine GBL Upgrade File

When the bootloader has entered the receive loop, a GBL upgrade file containing a Secure Engine upgrade is transmitted to the bootloader. When a packet is received, it is passed to the image parser. The image parser parses the data and

Copyright © 2025 Silicon Laboratories. All rights reserved.

returns Secure Engine upgrade data in a callback. The bootloader core implements this callback and flashes the data to internal flash at the pre-configured bootloader upgrade location.

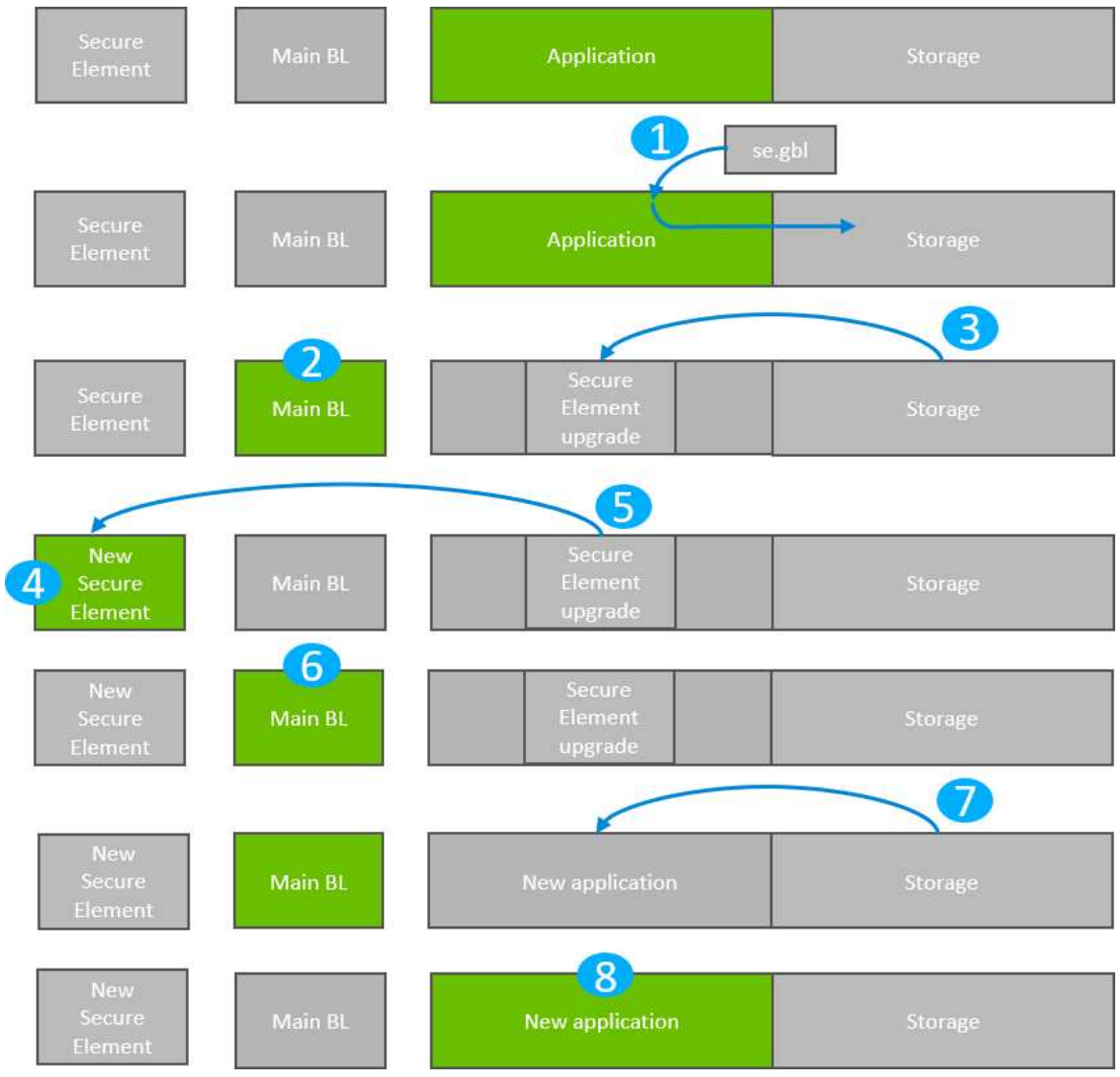
When a complete Secure Engine upgrade image is received, the main bootloader signals the Secure Engine that it should enter firmware upgrade mode. This is done by the Secure Engine communication interface that is used to signal that bootloader upgrade is ready to be performed.

Downloading and Applying an Application GBL Upgrade File

Once the Secure Engine upgrade is completed, the existing application is rendered invalid if the Secure Engine upgrade location overlaps with the application. A GBL upgrade file containing an application upgrade is transmitted to the bootloader. The application upgrade process follows that. For more information, see section Standalone Bootloader Operation.

Secure Engine Upgrade on Application Bootloaders with Storage

The process is illustrated in the following figure.



1. A single GBL file containing both a Secure Engine upgrade image and an application image is downloaded onto the storage medium of the device (internal flash or external SPI flash).
2. The device reboots into the bootloader.
3. a) The main bootloader copies its upgrade image into internal flash at the pre-configured upgrade location.
   b) Alternatively, if the no-staging Secure Engine upgrade option has been enabled, the upgrade image will be fetched directly from the GBL file in storage instead of first copying the image to the pre-configured upgrade location.

Copyright © 2025 Silicon Laboratories. All rights reserved.

4. The device reboots into the Secure Engine.
5. The Secure Engine is replaced by the new version found in the pre-configured upgrade location (or directly from storage, ref. 3b).
6. The device boots into the main bootloader.
7. The bootloader applies the application image from the GBL upgrade file.
8. The device boots into the application. Secure Engine upgrade is complete.

Storage Space Size Configuration

The storage space size must be configured to have enough space to store the upgrade images. The following table shows the reserved SE upgrade image sizes.

Device Family	Reserved Flash for SE Upgrade Image
EFR32xG21	48 kB
EFR32xG22	24 kB
EFR32xG23	96 kB

Depending on the configuration, the bootloader size can vary. For size requirements of the bootloader, see section Size Requirements for Different Bootloader Configurations for Series 1 Devices. The bootloader size for EFR32xG21 devices can be up to 16 kB and for EFR32xG22, EFR32xG23, and EFR32xG24 devices the bootloader size can be up to 24 kB. For more details, see *UG103.6: Bootloader Fundamentals*.

Copyright © 2025 Silicon Laboratories. All rights reserved.

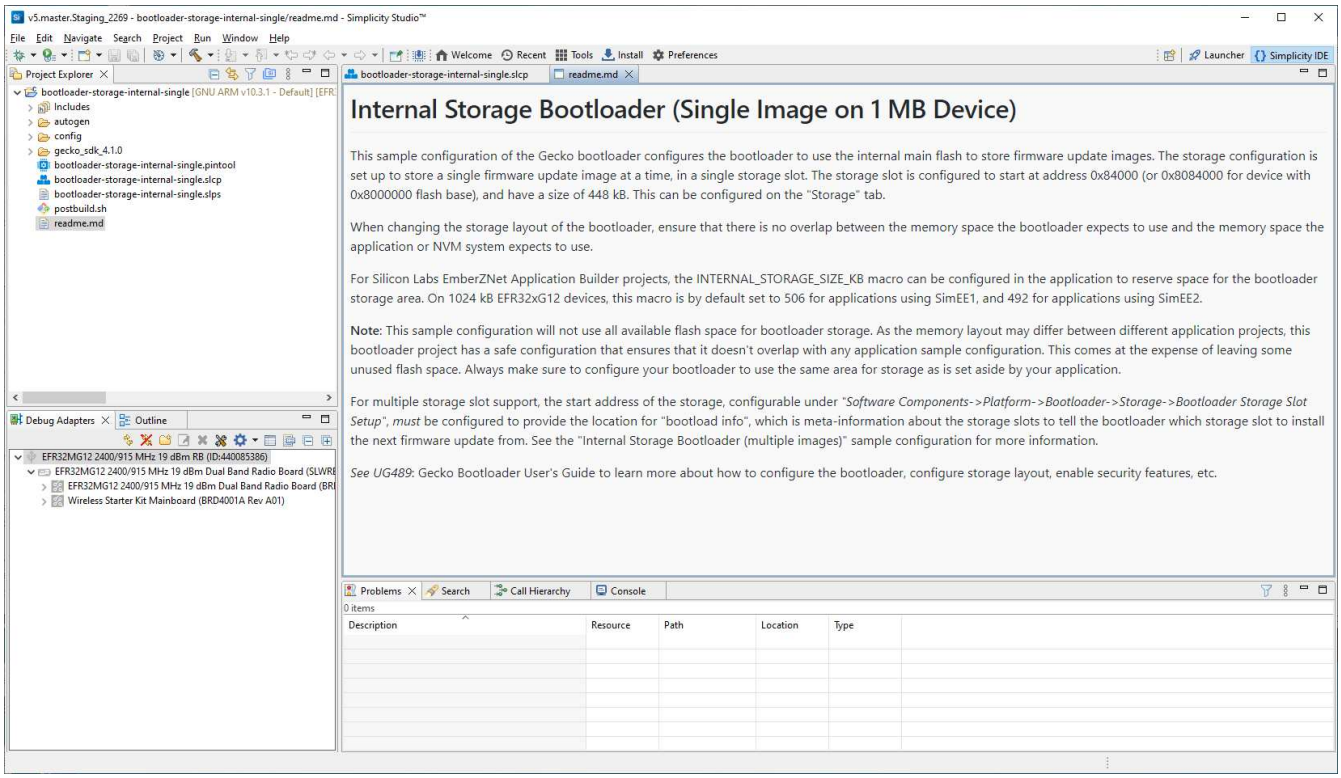
Getting Started with the Gecko Bootloader

Getting Started with the Gecko Bootloader

This section describes how to build a Gecko Bootloader from one of the provided examples. Simplicity Studio 5, used with Gecko SDK Suite (GSDK) v3.x and higher, differs from Simplicity Studio 4, used with GSDK v2.x, in how sample applications are selected and how projects are created. Refer to the documentation provided with your SDK for details. These instructions assume that you have installed Simplicity Studio 5, the GSDK and associated utilities as described in the SDK's quick start guide, and that you are familiar with generating, compiling, and flashing an example application in the relevant version.

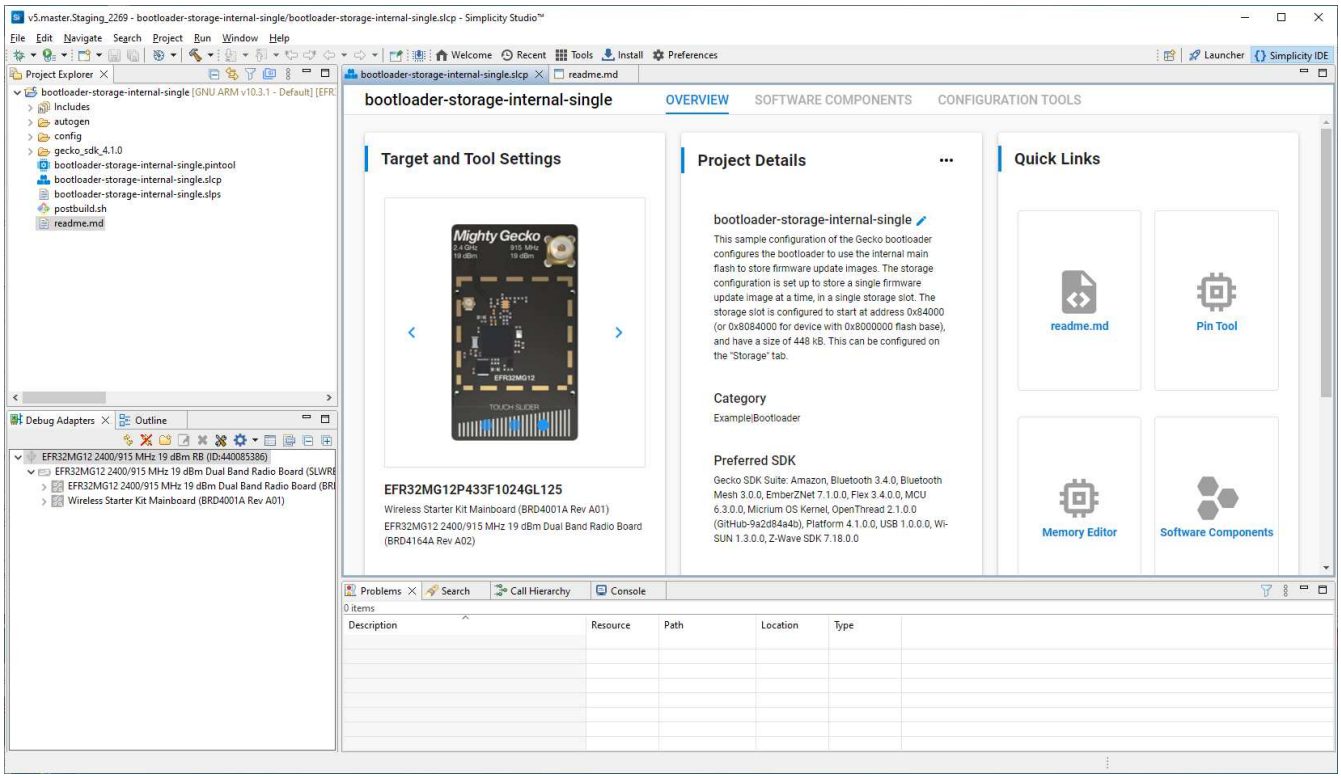
- *QSG169: Bluetooth® SDK v3.x Quick-Start Guide*
- Proprietary Flex SDK v3.x Quick-Start Guide
- *QSG170: Silicon Labs OpenThread SDK Quick-Start Guide*
- *QSG180: Zigbee EmberZNet Quick-Start Guide for SDK 7.0 and Higher*

1. Create a project based on the Gecko Bootloader example of your choice. The project opens with a tab describing the example.

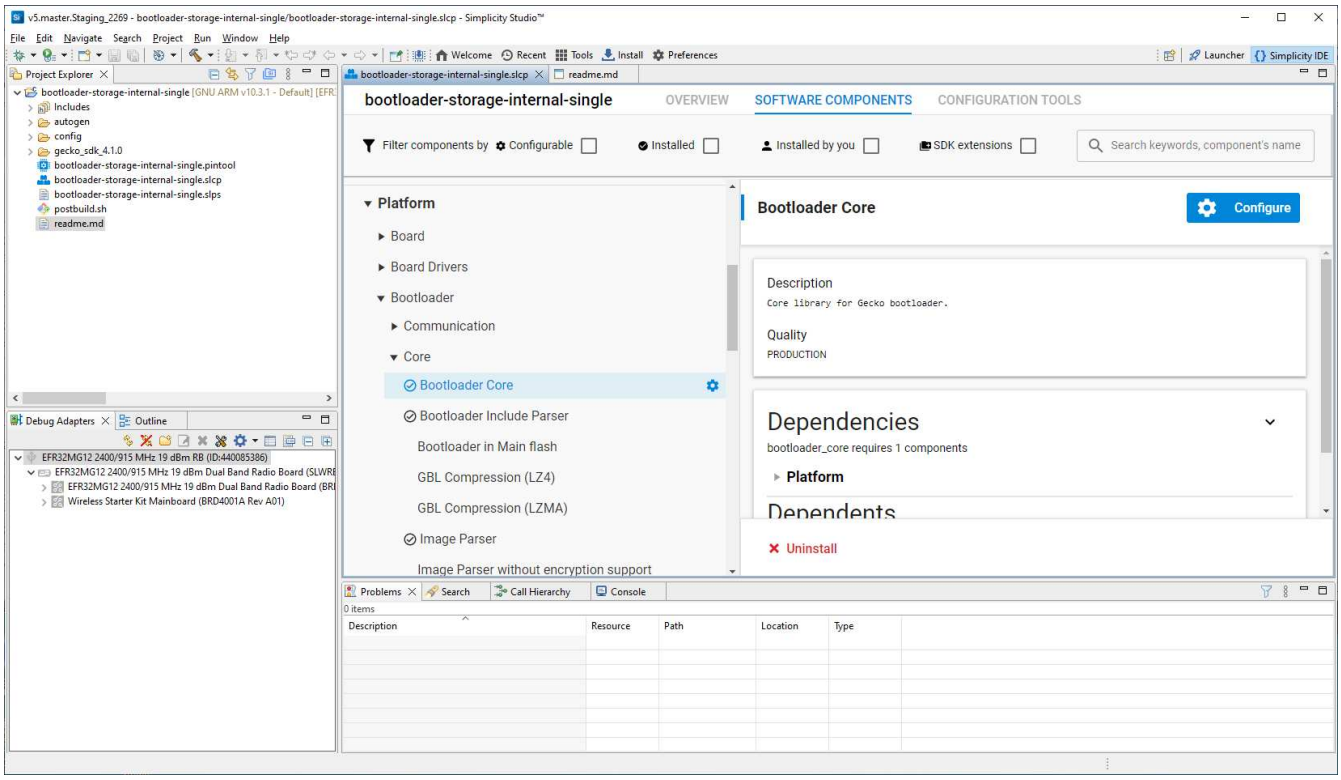


2. Click the project (*.slcp) tab to move to the Project Configurator interface.

Copyright © 2025 Silicon Laboratories. All rights reserved.

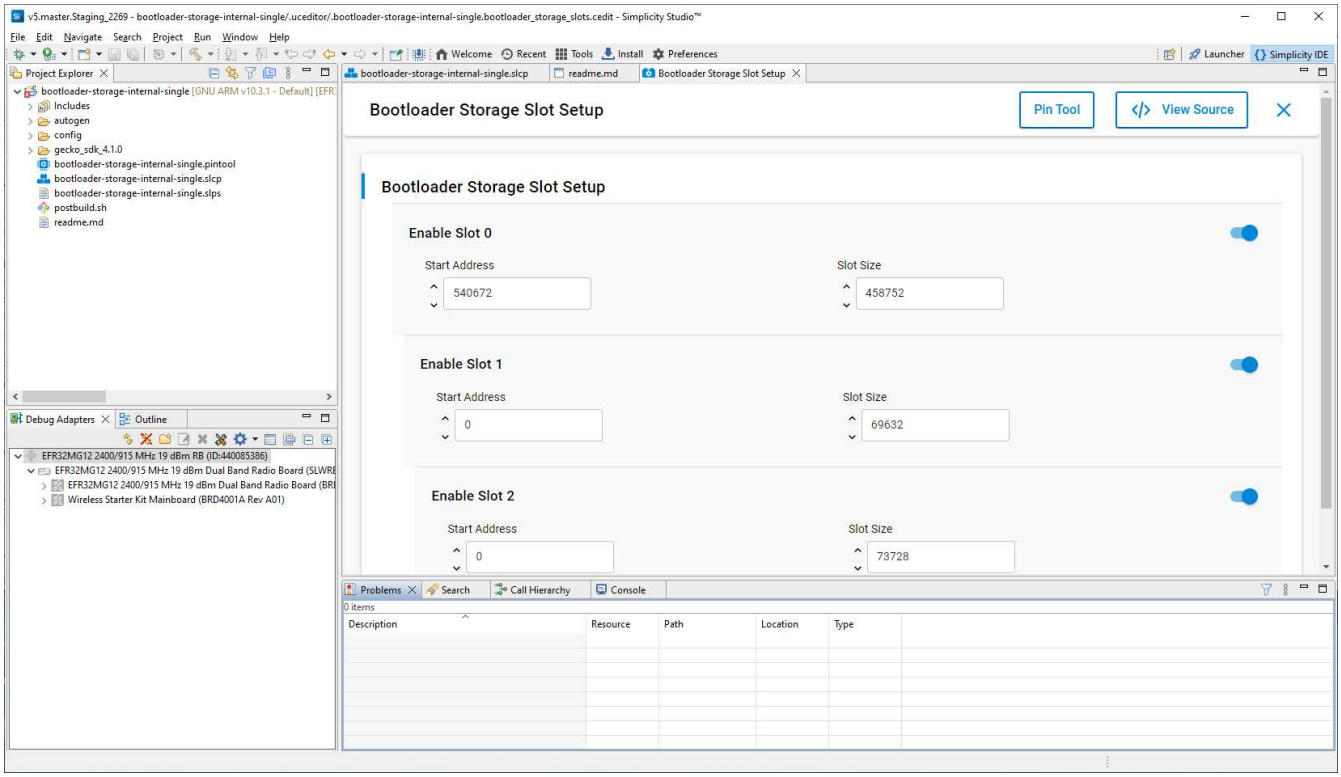


3. The Software Components tab shows the list of available components that can be installed in the project.

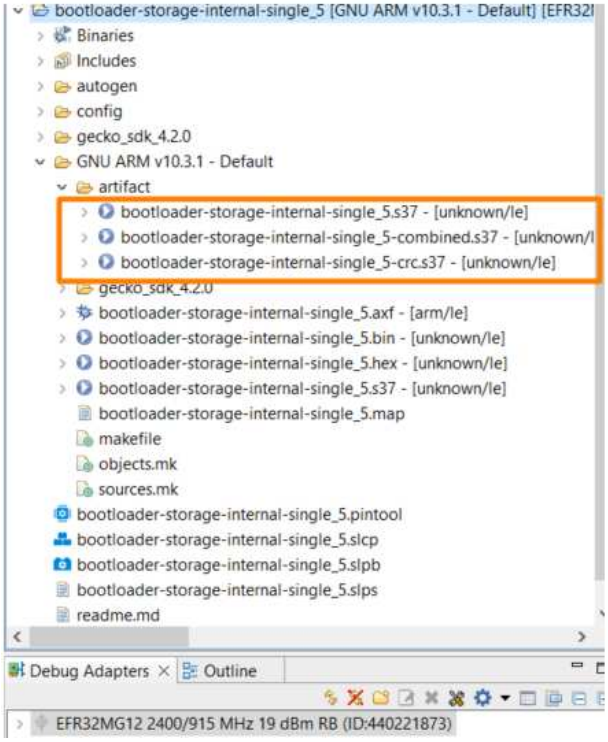


4. The **Storage Slot Setup** component allows you to configure storage slots to be used if a storage component is also installed. The default configuration matches the target part and bootloader type. This component supports a maximum of three storage slots.

Copyright © 2025 Silicon Laboratories. All rights reserved.



5. Click the **Build** (hammer) icon.
6. After the build is complete, the bootloader binaries are available in the **artifact** folder as depicted in the image below.



On Series 1 devices, three bootloader images are generated into the build directory: a main bootloader, a main bootloader with CRC32 checksum, and a combined first stage and main bootloader with CRC32 checksum. The main bootloader image is called **<project- name>.s37**, the main bootloader with CRC32 checksum is called **<projectname>-crc.s37**, while the

Copyright © 2025 Silicon Laboratories. All rights reserved.

combined first stage image + main bootloader image with a CRC32 checksum is called **<projectname>-combined.s37**. The first time a device is programmed, whether during development or manufacturing, the combined image needs to be programmed. For subsequent programming, when a first stage bootloader is already present on the device, it is okay to download an image containing just the main bootloader. In this case, the main bootloader with CRC32 should be used.

The requirement is that any main bootloader image that is programmed via serial wire must contain the CRC32 in the image. Files downloaded via serial wire are “s37” files. Most often, the **<projectname>-combined.s37** file is the one downloaded during production programming. However, it is possible to download only the main bootloader over serial wire, in which case **<projectname>-crc.s37** should be used.

Any main bootloader that is upgraded with the OTA or host method should already contain CRC32 because bootloader-initiated upgrades use GBL files (not “s37” files) and Simplicity Commander adds the CRC32 when it constructs the GBL file. The input files to Simplicity Commander can (and should) use the non-CRC “s37” file.

On Series 2 devices, the combined image is not present, since the first stage bootloader does not exist. The image containing only a main bootloader is the image that must be used to create a GBL file for bootloader upgrade.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configuring the Gecko Bootloader

Configuring the Gecko Bootloader

Configuring Storage

Gecko Bootloaders configured as application bootloaders must include an API to store and access image files. This API is based on the concept of storage slots, where each slot has a predefined size and location in memory and can be used to store a single upgrade image. Slots are configured in the **Bootloader Storage Slot Setup** component.

When multiple storage slots are configured, a bootload list is used to indicate the order in which the bootloader should access slots to find upgrade images. If multiple storage slots are supported, the application should write the bootload list by calling bootloader_setImageToBootload before rebooting into the bootloader to initiate a firmware upgrade process. The bootloader attempts to verify the images in these storage slots in sequence and applies the first image to pass verification. If only a single storage slot is supported, the bootloader uses this slot implicitly. A maximum of three slots may be configured in the **Bootloader Storage Slot Setup** component.

SPI Flash Storage Configuration

When configuring a Gecko Bootloader to obtain images from SPI flash, modify the following.

The **base address of the storage area** should be configured in the **Common Storage** component. This is the address at which the bootloader places the bootload list, if more than one storage slot is configured. In the default configuration, this address is set to 0. If only a single storage slot is configured, the bootload list is not used, so configuring it may be omitted.

The **location and size of the storage slots** can be configured using the **Bootloader Storage Slot Setup** component (supports a maximum of three configurable storage slots). The addresses input here are absolute addresses (they are not offsets from the base address). If more than a single slot is configured, space must be reserved between the base address as configured in the **Common Storage** component and the first storage slot configured in the **Bootloader Storage Slot Setup** component. Enough space to fit two copies of the bootload list must be reserved. These two copies need to reside on different flash pages, to provide redundancy in case of power loss during writing. Two full flash pages therefore need to be reserved. In the default example application, a SPI flash part with 4 kB flash sectors is used. This means that 8 kB must be reserved before the first storage slot. The following figure illustrates how the storage area can be partitioned, where the numbers in the top row represent the starting addresses.

External storage with two storage slots

0	4 kB	8 kB	260 kB
Bootload list	Bootload list	Storage slot 0 Size = 252 kB	Storage Slot 1 Size = 252 kB

External storage with one storage slot

0
Storage slot 0 Size = 512 kB

Internal Storage Configuration

Copyright © 2025 Silicon Laboratories. All rights reserved.

When configuring a Gecko Bootloader to obtain images from internal flash, modify the following.

The **base address of the storage area** should be configured in the **Common Storage** component. This is the address at which the bootloader will place the prioritized list of storage slots to attempt to bootload from, if more than one storage slot is configured. In the default configuration, only a single storage slot is configured, so this value is set to 0, and isn't used. If more than one storage slot is configured, this value needs to be configured too.

The **location and size of the storage slots** can be configured using the **Bootloader Storage Slot Setup** Component (supports a maximum of three configurable storage slots). The addresses input here are absolute addresses (they are not offsets from the base address). If more than a single slot is configured, enough space must be reserved between the base address as configured in the **Common Storage** component and the first storage slot configured in the **Bootloader Storage Slot Setup** component. Enough space to fit two copies of the bootload list must be reserved. These two copies need to reside on different flash pages, to provide redundancy in case of power loss during writing. Two full flash pages therefore need to be reserved. The following figure illustrates how the storage area can be partitioned.

Internal storage with one storage slot

0	512 kB
Application	Storage Slot 0 Size = 512 kB

Internal storage with two storage slots

0	340 kB	342 kB	344 kB	648 kB
Application	Bootload list	Bootload list	Storage slot 0 Size = 340 kB	Storage slot 1 Size = 340 kB

> **Note**: The storage area partitioning in the example for two storage slots above does not take any NVM system into account. If using an NVM system like SimEE or PS Store, take care to place and size the storage area in such a way that bootloader storage does not overlap with NVM.

Compressed Upgrade Images

The Gecko Bootloader optionally supports compressed GBL files. In a compressed GBL file, only the application upgrade data is compressed, any metadata and bootloader upgrade data (if present) stay uncompressed. This means that a compressed GBL file is identical to a normal (uncompressed) GBL file, except that the GBL Programming Tag containing the application upgrade image (as described in *UG103.6: Bootloader Fundamentals*) has been replaced by a GBL LZ4 Compressed Programming Tag or GBL LZMA Compressed Programming Tag. Signature and encryption operations on a compressed GBL work identically to on an uncompressed GBL.

To be able to use compressed upgrade images, a decompressor for the relevant compression algorithm must be added to the Gecko bootloader. The following table shows which compression algorithms are supported by the Gecko Bootloader, and which Bootloader component should be added to enable the feature. The table also shows how much space the decompressor takes up in the bootloader, and how big of a size reduction to expect for the compressed application upgrade image. Be aware of the bootloader size requirement. The bootloader space might be too small to fit the decompressors, depending on the device and enabled components.

Compression Algorithm	Component	Bootloader Size Requirement	Application Upgrade Size Reduction (typical)
LZ4	GBL Compression (LZ4)	< 1 kB	~ 10%

Copyright © 2025 Silicon Laboratories. All rights reserved.

Compression Algorithm	Component	Bootloader Size Requirement	Application Upgrade Size Reduction (typical)
LZMA	Bootloader Compression (LZMA)	~5 kB flash, 18 kB RAM	~ 30%

It is important to note that the compressed GBL file stays compressed while being transferred to the device, and while it is stored in the upgrade area. It is decompressed by the bootloader when the upgrade is applied. This means that the running application in main flash will be identical to one that was installed using an uncompressed (normal) GBL file.

Compressed GBL files can only be decompressed by the bootloader when running standalone, not through the Application Interface. This means that upgrade image verification performed by the application prior to reboot will not attempt to decompress the application upgrade, it will only verify the signature of the compressed payload. After rebooting into the bootloader, it will decompress the image as part of the upgrade process.

> **Note**: The above means that Bluetooth in-place application upgrades cannot be compressed, as they are processed by the Bluetooth Supervisor or AppLoader using functionality in the bootloader through the Application Interface. Supervisor/stack and AppLoader updates can be compressed, but the user application cannot.

LZMA Compression Settings

LZMA decompression is only supported for images compressed with certain compression settings. Simplicity Commander automatically uses these settings when using the `commander gbl create --compress lzma` command.

- Probability model counters: lp + lc <= 2. Simplicity Commander uses lp=1, lc=1.
- Dictionary size no greater than 8 kB. Simplicity Commander uses 8 kB.

Together, these settings cause the decompressor to require 18 kB of RAM for decompression – 10 kB for the counters and 8 kB for the dictionary.

The GBL LZMA Compressed Programming Tag contains a full LZMA file, containing the LZMA header, raw stream, and end mark. The Gecko bootloader only supports decompressing payloads that contain the end mark as the last 8 bytes of the compressed stream.

Bootloader Example Configurations

The following sections describe the key configuration options for the example bootloader applications.

> **Note**: Security features are disabled for all example configurations. In development, Silicon Labs strongly recommends enabling security features to prevent unauthorized parties from uploading untrusted program code. See section Using Application Image Security Features to learn how to configure the security features of the Gecko Bootloader.

UART XMODEM Bootloader

Standalone bootloader for EFM32 and EFR32 devices running the EmberZNet PRO and Silicon Labs Connect protocol stacks, using XMODEM-CRC over UART.

In this configuration, the **UART XMODEM** communication component, **XMODEM Parser** component, and **Bootloader UART Driver** component are installed. For the example application to run on a custom board, the GPIO ports and pins used for UART need to be configured in the **Bootloader UART Driver** component. Here, Hardware Flow Control can be enabled or disabled, and the baud rate and pinout can be configured.

The **GPIO activation** component is also installed by default, allowing bootloader entry into firmware upgrade mode by activating a GPIO through reset. The GPIO pin used can be configured here. This component can be uninstalled if this

Copyright © 2025 Silicon Laboratories. All rights reserved.

functionality is not desired.

BGAPI UART DFU Bootloader

Standalone bootloader for the Bluetooth protocol stack, using the BGAPI protocol for UART DFU. This bootloader should be used for all NCP-mode Bluetooth applications.

In this configuration, the **BGAPI UART DFU** communication component and **Bootloader UART Driver** component are installed. For the example application to run on a custom board, the GPIO ports and pins used for UART need to be configured in the **Bootloader UART Driver** component. Here, Hardware Flow Control can be enabled or disabled, and the baud rate and pinout can be configured.

The GPIO activation component is also installed by default, allowing bootloader entry by activating a GPIO through reset. The GPIO pin used can be configured here. This component can be uninstalled if this functionality is not desired.

EZSP SPI Bootloader

Standalone bootloader for EmberZNet PRO and Silicon Labs Connect protocol stacks using EZSP for SPI.

In this configuration, the **EZSP-SPI** communication component, **XMODEM Parser** component, and **Bootloader SPI Peripheral Driver** component are installed. For the example application to run on a custom board, the GPIO ports and pins used for SPI and EZSP signaling need to be configured in the **Bootloader SPI Peripheral Driver** and **EZSP-SPI** components, respectively.

The **EZSP GPIO activation** component is also installed by default, allowing bootloader entry into firmware upgrade mode by activating a GPIO through reset. The GPIO pin used can be configured here. This component can be uninstalled if this functionality is not desired.

Bluetooth AppLoader OTA DFU Bootloader

Standalone bootloader for the Bluetooth protocol stack, using Bluetooth for Over-The-Air DFU.

In this configuration, the **Bluetooth AppLoader** communication component is installed.

SPI Flash Storage Bootloader

Application bootloader for all wireless protocol stacks, using an external SPI flash to store upgrade images received over the air by the application.

In this configuration, the **SPI Flash Storage** and **Common Storage** components, as well as the **Bootloader SPI Controller Driver** component, are installed. For the example application to run on a custom board, the GPIO ports and pins used for SPI communication with the external flash need to be configured in the **Bootloader SPI Controller Driver** component, and the type of SPI flash needs to be configured in the **SPI Flash Storage** component. The base address of the storage area can be configured in the **Common Storage** component. The location and size of the storage slots themselves can be configured using the **Bootloader Storage Slot Setup** component (supports up to three configurable storage slots).

The SPI Flash storage bootloader provides customizable callback functions, which can be found in *bootloader-callback-stubs.c*. The callback functions *storage_customInit* and *storage_customShutdown*, for example, can be used for custom hardware setups. Those callback functions can be interfaced from the applications using the bootloader interface functions such as *bootloader_init* and *bootloader_deinit* from *api/btl_interface.h*.

Currently, Silicon Labs supports the following SPI Flash storage bootloaders:

1. SPI Flash storage bootloader (multiple images)
2. SPI Flash storage bootloader (single image)
3. SPI Flash storage bootloader (single image with slot size of 1MB). **Note: This bootloader should only be used for systems with an external SPI Flash size >= 1 MB**.

SPI Flash Storage Bootloader SFDP

Application bootloader for all wireless protocol stacks, using an external SPI flash to store upgrade images received over the air by the application.

In this configuration, the **SPI Flash Storage SFDP** and **Common Storage** components, as well as the **Bootloader SPI Controller Driver** component, are installed. For the example application to run on a custom board, the GPIO ports and pins

Copyright © 2025 Silicon Laboratories. All rights reserved.

used for SPI communication with the external flash need to be configured in the **Bootloader SPI Controller Driver** component. The type of SPI flash is detected at runtime. The base address of the storage area can be configured in the **Common Storage** component. The location and size of the storage slots themselves can be configured using the **Bootloader Storage Slot Setup** component (supports up to three configurable storage slots).

The SPI Flash storage bootloader SFDP provides customizable callback functions, which can be found in *bootloader-callback-stubs.c*. The callback functions *storage_customInit* and *storage_customShutdown*, for example, can be used for custom hardware setups. Those callback functions can be interfaced from the applications using the bootloader interface functions such as *bootloader_init* and *bootloader_deinit* from *api/btl_interface.h*.

Internal Storage Bootloader

Application bootloader for all wireless protocol stacks, using internal flash to store upgrade images received over the air by the application.

Multiple examples are provided, including configurations for 512 kB flash memory devices like EFR32xG13, 1024 kB flash memory devices like EFR32xG12, and 2048 kB flash memory devices like EFM32GG11. **The storage layout should be modified before running the bootloader on any other devices**. In this configuration, the internal flash and common storage components are installed. The base address of the storage area is configured in the **Common Storage** component. The location and size of the storage slots can be configured using the **Bootloader Storage Slot Setup** component (provides up to three configurable storage slots). Default example applications are provided with configurations for both single storage slot and multiple storage slots.

The default storage slot configurations provided by the Gecko Bootloader **must** be configured to match the use-case-specific application configurations.

Table: Internal Storage Bootloader Default Storage Configurations

Sample Applications	Storage Offset	Storage Size	Additional Notes
Internal Storage Bootloader (single image on 256 kB device)	0x21800 (137216)	88064	This configuration is available for devices where flash base address is 0x00
'	0x8021800 (134354944)	88064	This configuration is only available for devices where the flash base address is 0x08000000
Internal Storage Bootloader (single image on 352 kB device)	0x28000 (163840)	147456	This configuration is available for devices where flash base address is 0x00
'	0x8028000 (134381568)	147456	This configuration is only available for devices where the flash base address is 0x08000000
Internal Storage Bootloader (single image on 512 kB device)	0x44000 (278528)	196608	This configuration is available for devices where flash base address is 0x00
'	0x8044000 (134496256)	196608	This configuration is only available for devices where the flash base address is 0x08000000
Internal Storage Bootloader (single image on 1 MB device)	0x84000 (540672)	458752	This configuration is available for devices where flash base address is 0x00
'	0x8084000 (134758400)	458752	This configuration is only available for devices where the flash base address is 0x08000000

Copyright © 2025 Silicon Laboratories. All rights reserved.

Sample Applications	Storage Offset	Storage Size	Additional Notes
Internal Storage Bootloader (single image on 2 MB device)	0x10000 (1048576)	1011712	This configuration is available for devices where flash base address is 0x00
'	0x8100000 (135266304)	1011712	This configuration is only available for devices where the flash base address is 0x08000000
Internal Storage Bootloader (multiple images on 1 MB device)	0x5A000 (368640)	337920	This configuration is available for devices where flash base address is 0x00
'	0xAC800 (706560)	337920	This configuration is available for devices where flash base address is 0x00
Internal Storage Bootloader (single image on 1536kB device)	0xC0000 (786432)	737280	This configuration is available for devices where flash base address is 0x00
'	0x80C0000 (135004160)	737280	This configuration is only available for devices where the flash base address is 0x08000000
Internal Storage Bootloader (single image on 1920kB device)	0x80E8000 (135168000)	966656	This example application is available for device families with flash size of 1920kB or higher (for example xG25 device family)

Image Acquisition Application Example Configuration

These examples illustrate applications that acquire and store a GBL upload image for an application bootloader. For the running bootloader to accept an application upgrade, the new application version must be higher than the existing version. The version number can be set in Appbuilder by defining the macro “APP_PROPERTIES_VERSION” in the **Additional Macros** field on the **Other** tab.

Setting a Version Number

To distinguish between different builds of the Gecko Bootloader, it is useful to set a version number. To perform a bootloader upgrade, not only must the running bootloader pass its integrity checks (see section Downloading and Applying a Bootloader GBL Upgrade File), but the bootloader upgrade image must also have a higher version number than the running bootloader image. A version number can be set using Simplicity Studio by configuring the **Bootloader Version Main Customer** option of the **Bootloader Core** component. This macro will be picked up by the config file **btl_config.h**, where it is combined with the version number of the Gecko Bootloader files provided by Silicon Labs.

Placing Bootloader in Main Flash

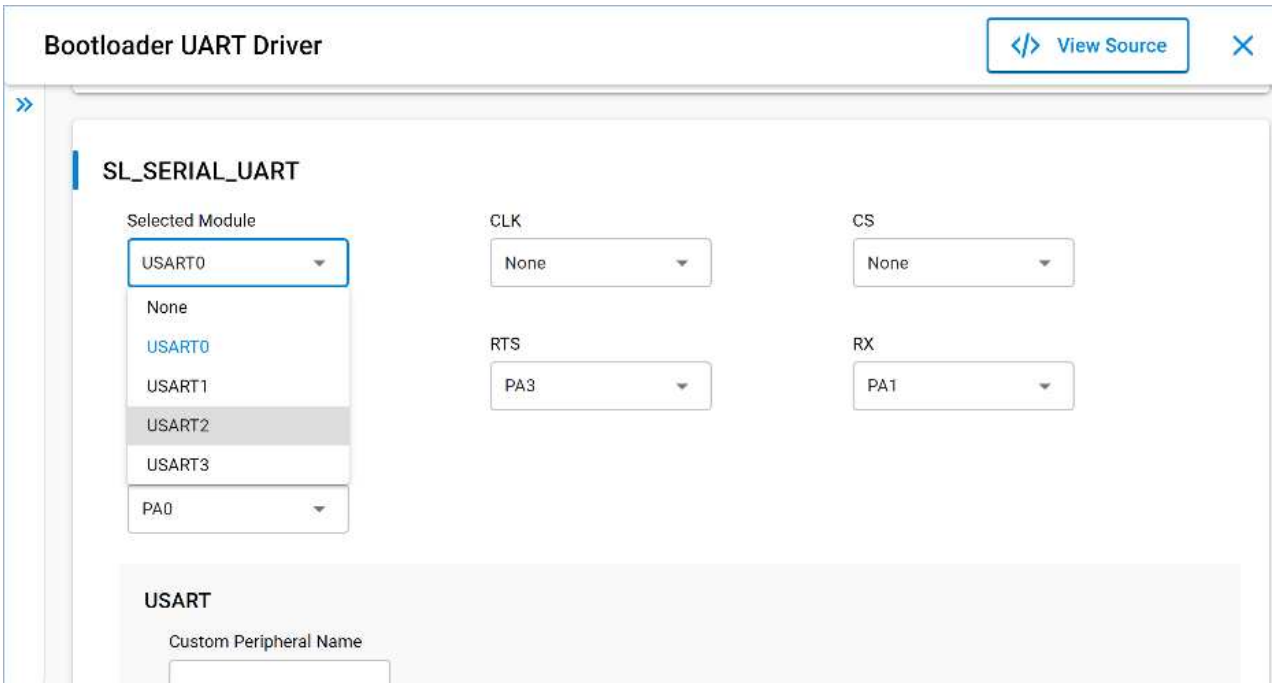
For device families xG13 and xG14, the entire main stage bootloader might not fit into the bootloader flash if the user installs some extra components. In such scenarios, the main stage bootloader can be placed in the main flash by installing the Bootloader in Main Flash core component. This component internally provides a MACRO named MAIN_BOOTLOADER_IN_MAIN_FLASH which is used to set the appropriate address required to place the main bootloader in main flash. The MACRO is used to generate the linker file from the common linker template with the appropriate addresses to place the main stage bootloader in the main flash.

Hardware Configuration

The Gecko Bootloader uses the Pin Tool for configuration of pinout and other hardware-related settings. When Pin Tool configuration is available for a bootloader component, the relevant settings are shown in the Component Editor for that

Copyright © 2025 Silicon Laboratories. All rights reserved.

component.



The standalone Pin Tool User Interface can also be used to configure settings for Gecko Bootloader if desired.

While the Pin Tool provides configuration for many different peripherals, the Gecko Bootloader uses only the following Pin Tool modules:

- SERIAL is used by the UART Driver component to configure baud rate, flow control mode and pinout.
- VCOM is used by the UART Driver component to enable the serial interface if necessary (only on Silicon Labs Wireless Starter Kits).
- EXTFLASH is used by the SPI Driver to configure frequency and pinout.
- SPINCP is used by the SPI Slave Driver to configure pinout.
- BTL_BUTTON is used by the GPIO Activation component.
- CMU HFXO frequency setting is used by the delay driver to calibrate timing if the core is running from the HFXO.

Other settings, like CMU oscillator configuration or DCDC configuration, are not taken into consideration by the default bootloader code. If using these configuration settings is desired, the required code must be added in btl_main.c.

> **Note**: While the delay driver uses the HFXO frequency setting from Pin Tool, the HFXO enable setting is not used to initialize the HFXO on startup. This setting is only used when calling the bootloader through the Application Interface (see section Gecko Bootloader and TrustZone), and the application has switched to the HFXO prior to calling the Bootloader Application Interface API.

Size Requirements for Different Bootloader Configurations for Series 1 Devices

Enabling different configuration options for the Gecko Bootloader changes the size of the resulting image. The following table shows a list of example bootloader configurations and the resulting approximate size of the main bootloader. Note that any size above 14 kB will be too large to fit in the bootloader area of flash of EFR32xG13, and that any size above 16 kB will be too large to fit in the bootloader area of flash on EFR32xG14 and EFM32TG11. This table was last updated for Gecko SDK Suite 4.0 (Gecko Bootloader 2.0), and no guarantees are made that a configuration of a specific size with one SDK version will maintain that size in future releases.

Table: Bootloader Size Requirements

Copyright © 2025 Silicon Laboratories. All rights reserved.

Base Configuration	Enabled Options	Size (kB)
XMODEM UART	Default configuration	12.8
'	Secure boot, signed and encrypted upgrade	12.8
'	Secure boot, signed and encrypted upgrade, LZ4 compression	13.7
'	Secure boot, signed and encrypted upgrade, LZMA compression	17.9
Internal storage, single slot	Default configuration	11.5
'	Secure boot, signed and encrypted upgrade	11.5
'	Secure boot, signed and encrypted upgrade, LZ4 compression	12.4
'	Secure boot, signed and encrypted upgrade, LZMA compression	16.6
Internal storage, multiple slots	Default configuration	12.1
'	Secure boot, signed and encrypted upgrade	12.1
'	Secure boot, signed and encrypted upgrade, LZ4 compression	13
'	Secure boot, signed and encrypted upgrade, LZMA compression	17.1
SPI flash, single slot	Default configuration	12.6
'	Secure boot, signed and encrypted upgrade	12.6
'	Secure boot, signed and encrypted upgrade, LZ4 compression	13.5
'	Secure boot, signed and encrypted upgrade, LZMA compression	17.7
SPI flash, multiple slots	Default configuration	13.1
'	Secure boot, signed and encrypted upgrade	13.1
'	Secure boot, signed and encrypted upgrade, LZ4 compression	14
'	Secure boot, signed and encrypted upgrade, LZMA compression	18
SPI flash using SFDP, single slot	Default configuration	12.8
'	Secure boot, signed and encrypted upgrade	12.8
'	Secure boot, signed and encrypted upgrade, LZ4 compression	13.8
'	Secure boot, signed and encrypted upgrade, LZMA compression	18
BGAPI UART	Default configuration	11.8
'	Secure boot, signed and encrypted upgrade	11.8
'	Secure boot, signed and encrypted upgrade, LZ4 compression	12.7
'	Secure boot, signed and encrypted upgrade, LZMA compression	16.9
EZSP SPI	Default configuration	13.1
'	Secure boot, signed and encrypted upgrade	13.1
'	Secure boot, signed and encrypted upgrade, LZ4 compression	13.9

Copyright © 2025 Silicon Laboratories. All rights reserved.

Base Configuration	Enabled Options	Size (kB)
'	Secure boot, signed and encrypted upgrade, LZMA compression	18.1

Copyright © 2025 Silicon Laboratories. All rights reserved.

Simplicity Commander and the Gecko Bootloader

Simplicity Commander and the Gecko Bootloader

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It is invoked using a simple CLI (Command Line Interface that is also scriptable. You can use Simplicity Commander to perform these essential tasks:

- Generating key files for signing and encryption
- Signing application images for Secure Boot
- Creating GBL images (encrypted or unencrypted, signed, or unsigned)
- Parsing GBL images

Simplicity Commander is used throughout the examples in the following sections. For more information on executing the commands to complete these tasks, see *UG162: Simplicity Commander Reference Guide*.

> **Note**: Simplicity Commander also offers a GUI (Graphical User Interface) that can be used in the lab for typical tasks such as flashing device images. The functions described in this User Guide are performed from the CLI.

Creating GBL Files Using Simplicity Commander

To create an unsigned GBL file from an application **myapp.s37**, execute `commander gbl create myapp.gbl --app myapp.s37`.

To create an unsigned GBL file from a main bootloader upgrade **mybootloader.s37**, execute `commander gbl create mybootloader.gbl --bootloader mybootloader.s37`. This file can be used with the standalone bootloader configurations of the Gecko Bootloader.

To create an unsigned GBL file from a Secure Engine, upgrade **mySecureElement.seu**, and execute `commander gbl create mySecureElement.gbl --seupgrade mySecureElement.seu`. The Secure Engine images, .seu, are provided by Silicon Labs and can be found through Simplicity Studio. See section Gecko Bootloader Operation - Secure Engine Upgrade.

These commands can also be combined to create a single upgrade image, suitable for use with application bootloader configurations of the Gecko Bootloader: `commander gbl create myupgrade.gbl --app myapp.s37 --bootloader mybootloader.s37 --seupgrade mySecureElement.seu`.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader Security Features

Gecko Bootloader Security Features

About Bootloader Image Security

Secure Boot and Secure Firmware Upgrade, discussed in the following sections, enable Gecko Bootloader to provide authenticity and integrity checks on the Application image, which provides a sufficient level of security for many applications. However, in systems without a hardware root of trust, no process checks the authenticity or integrity of the Gecko Bootloader itself. Its security is provided solely by the device hardware and the robustness of the software running on the device.

The native behavior of Firmware Upgrade will prevent accidental version rollback of Gecko Bootloader under normal usage conditions. However, without a hardware root of trust, intentional downgrade attacks may be feasible. If a higher level of security assurance is required by the application, using a Series 2 device with hardware root of trust and anti-rollback protection is recommended. For more information, refer to Series 2 Secure Boot with RTSL.

About Application Image Security

The Gecko Bootloader can enforce security on two levels:

- Secure Boot refers to the verification of the authenticity of the application image in main flash on every boot of the device.
- Secure Firmware Upgrade refers to the verification of the authenticity of an upgrade image before performing a bootload, and optionally enforcing that upgrade images are encrypted.

Secure Boot Procedure

When Secure Boot is enabled, the cryptographic signature of the application image in flash is verified on every boot before the application is allowed to run. Secure Boot is not enabled by default in the example configurations provided by Silicon Labs, but enabling it is highly recommended to ensure the validity and integrity of firmware images.

Signature Algorithms

The Gecko Bootloader supports the ECDSA-P256-SHA256 cryptographic signature algorithm. This is the ECDSA (elliptical curve digital signature algorithm) of the SHA-256 digest of the application firmware image, using the NIST P-256 (secp256r1) curve.

Summary of Operation

1. On boot, the bootloader checks the application image for information about whether it is signed.
2. The type of signature and signature location is determined.
3. If the type of signature does not match the requirements of the bootloader, the bootloader enters device firmware upgrade mode and prevents the application from running.
4. According to the chosen signature algorithm, the signature of the contents of flash from the beginning of the application to the location of the signature is compared to the signature at the signature location.
5. If the signatures do not match, the bootloader enters device firmware upgrade mode and prevents the application from running.

Secure Boot using ECDSA-P256-SHA256

For an image to be signed for Secure Boot, the application needs to contain a copy of the **ApplicationProperties_t** struct. This struct contains information about which signature algorithm is used, and where to find the signature.

On every boot, the bootloader calculates the SHA-256 digest of the application image, from the beginning of the application to the start of the signature. The signature of the SHA-256 digest is then verified using ECDSA-P256.

Copyright © 2025 Silicon Laboratories. All rights reserved.

If the signature is valid, the application is allowed to boot. Else, the bootloader is entered, and an application upgrade is attempted if one is available.

The public key used for signature verification is stored as a manufacturing token on Series 1 and EFR32xG22 devices. OnEFR32xG21 devices, the public key can either be stored as a manufacturing token or stored in the Secure Engine One-time Programmable memory (OTP). Simplicity Commander can be used to generate a key pair and write the public key to the device. See *UG162: Simplicity Commander Reference Guide* for more information.

Secure Boot with Application Rollback Protection

On every boot, the application version in the **ApplicationData_t** struct is stored at the end of the bootloader area in flash, which is used to prevent applications from being downgraded. The application version can remain the same for upgrades. The bootloader will only allow applications to increment its version 6 times by default. The bootloader can be upgraded after that to reset the stored application versions and to allow application upgrades again. The allowed number of upgrades can be increased by modifying linkerfiles to reserve an extra 4 bytes for each version and by increasing SL_GBL_APPLICATION_VERSION_STORAGE_CAPACITY from *btl_bootload.c*. **For Series 2 devices, this option is not applicable on devices with Secure Engine configured to perform full page lock**. If the bootloader area in flash is locked by Secure Engine, the bootloader will not be able to store the application versions in flash, and it would continue without performing that function.

When the application upgrade version check option is enabled in the **Bootloader Core** component, the bootloader checks where App properties points into flash. If app properties struct is present, then it will check compatibility of the application properties struct and check if the upgrade version is greater than the installed app; otherwise, it will check if product ID is correct before applying the upgrade image.

If app properties struct is absent at the app properties location, it is assumed that the previous upgrade was interrupted, and allows the upgrade to continue.

If application rollback prevention component is installed in the bootloader project, then before applying the upgrade image, the bootloader will check the application version in the application properties structure that resides inside the signed/encrypted GBL file and will only apply the OTA image to application area if the application version in ApplicationData structure is equal or higher than the highest application version last seen.

The application rollback prevention feature can be enabled in the **Bootloader Core** component by selecting the **Enable application rollback protection** option. The **Minimum application version allowed** option can be used to configure the minimum application version that should be allowed to boot.

The application versions are stored in the bootloader area in flash. To properly protect the stored application versions, it is recommended to lock the bootloader flash by selecting the **Prevent bootloader write/erase** option in the **Bootloader Core** component.

Secure Boot Using a Certificate

On Series 2 devices, a certificate-based secure boot operation is supported. The Certificate contains:

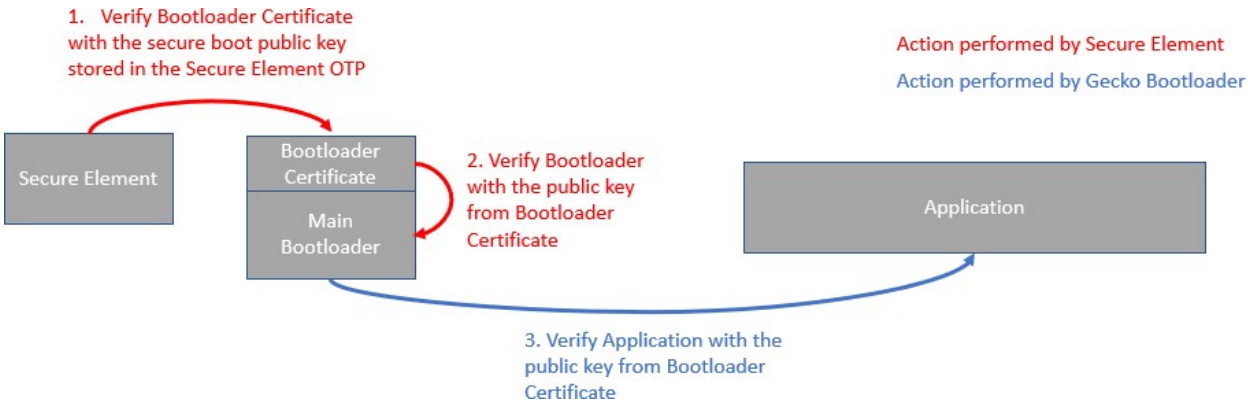
- Struct version: The version of the certificate structure.
- Public key: ECDSA-P256 public key, X and Y coordinates concatenated, used to validate the image.
- Certificate version: The version of the running certificate.
- Signature: ECDSA-P256 signature, used for the authentication of the public key and the certificate version.

The definition of the certificate struct can be found in api/application_properties.h.

To utilize certificate-based secure boot, configure Secure Engine to authenticate the bootloader image by configuring the certificate- based secure boot option in the Secure Engine OTP. Configure the Gecko Bootloader to enable certificate-based secure boot in the **Bootloader Core** component by selecting the **Enable certificate support** option. The Gecko Bootloader certificate must be signed by the private key pair of the public key stored in the Secure Engine OTP. For more information on the key storage, see section Key Storage.

The certificate-based secure boot procedure is illustrated in the following figure.

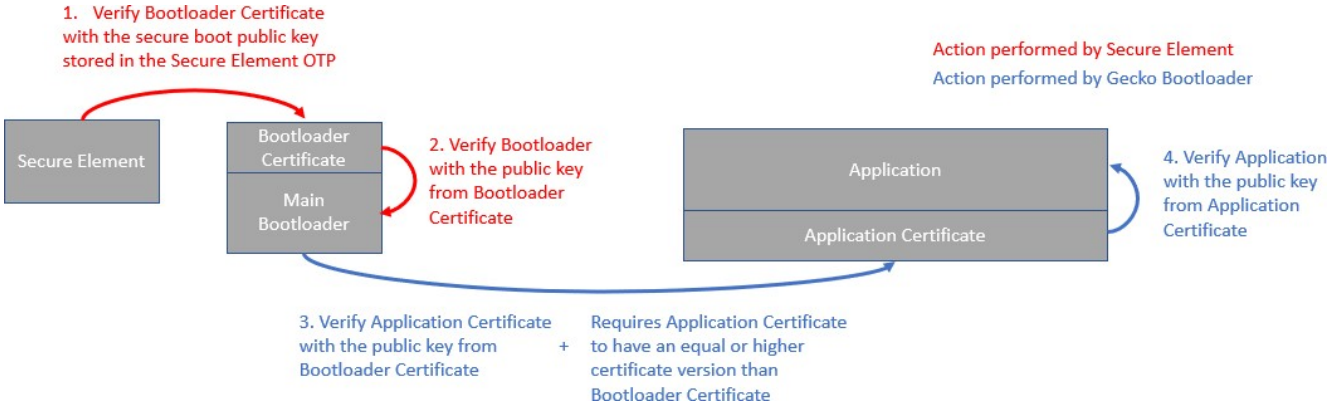
Copyright © 2025 Silicon Laboratories. All rights reserved.



Once the certificate-based secure boot option on Secure Engine is turned on, Secure Engine verifies the Gecko Bootloader certificate. The public key stored in the certificate is used to validate the signature of the Gecko Bootloader. Secure Engine will not accept bootloader images without a certificate.

If only the secure boot option is enabled (not certificate-based) on Secure Engine, and Secure Engine identifies a certificate, the certificate will be used to validate the bootloader image. If the certificate version from the bootloader image is higher than 0 and it gets verified once, Secure Engine will never again accept direct signed bootloader images without a certificate.

The Gecko Bootloader will authenticate the direct signed application using the public key stored in the Gecko Bootloader certificate. If the application contains a certificate, Gecko Bootloader will authenticate it. The procedure is illustrated in the following figure.



After authentication of the application certificate, Gecko Bootloader verifies the signature of the application using the public key from the application certificate. In addition, Gecko Bootloader compares the Gecko Bootloader certificate version against the application certificate version. All application images with a certificate version lower than the certificate version of the Gecko Bootloader will be rejected. Gecko Bootloader can be configured to only allow applications with certificate to boot by configuring the **Bootloader Core** component by selecting the **Reject direct signed images** option.

The **ApplicationProperties_t** struct contains the certificate struct **ApplicationCertificate_t**. The certificate struct can be injected to images that contain an **ApplicationProperties_t** with **ApplicationCertificate_t**. To inject a certificate to an image, issue the following command from Simplicity Commander:

```
commander convert <image file> --secureboot --keyfile <signing key> --certificate <certificate>--outfile <signed image file with certificate>
```

Secure Firmware Upgrade

The Gecko Bootloader supports a secure firmware upgrade process. This is achieved by using symmetric encryption to encrypt the upgrade image, and asymmetric cryptography to sign the upgrade image. Symmetric encryption provides

Copyright © 2025 Silicon Laboratories. All rights reserved.

confidentiality and asymmetric cryptography provides integrity and authenticity. Note that encryption alone is not enough to provide authenticity.

Encryption Algorithms

The Gecko Bootloader supports the AES-CTR-128 encryption algorithm. The GBL upgrade file is encrypted using 128-bit AES in Counter mode with a random nonce as the initial counter value. Through configuration, the GBL file's decryption duration can be improved by increasing the counter's block size. The block size is configurable with options 1, 2, 4, or 8 blocks. The configuration is available as part of the AES CTR Stream Block Config component.

The AES-128 key used for decryption, more commonly called the OTA decryption key, is stored as a manufacturing token on Series 1 and EFR32xG22 devices. On EFR32xG21 devices, the OTA decryption key can either be stored as a manufacturing token or stored in the Secure Engine OTP. To make use of the OTA decryption key stored in the Secure Engine OTP, the **Use symmetric key stored in Secure Engine storage** option in the **Bootloader Core** component must be selected. Simplicity Commander can be used to generate an OTA decryption key and write the key to the device. See *UG162: Simplicity Commander Reference Guide* for more information. For more information on storing the OTA decryption key on Series 2 devices, see *AN1222: Production Programming of Series 2 Devices*.

The Secure Engine OTP key support depends on the SE Manager component, which is enabled by default.

Signature Algorithms

The Gecko Bootloader supports the ECDSA-P256-SHA256 cryptographic signature algorithm. This is the ECDSA signature of the SHA-256 digest of the GBL upgrade file, using the NIST P-256 (secp256r1) curve.

Summary of Operation

Before starting a firmware upgrade process, the application can verify an image in storage by calling into the bootloader verification functions. For bootloaders with Communication Interface, the host device should verify the image before sending it to the NCP or RCP.

During firmware upgrade, the GBL file is parsed, and if encrypted, decrypted on-the-fly. A GBL Signature Tag in the GBL file indicates to the bootloader that the file is signed, and the signature is verified. If signature verification fails, the firmware upgrade process is aborted.

On Series 2 devices, Gecko Bootloader will authenticate the GBL signature tag using the public key stored in the bootloader certificate if the **Enable certificate support** option is selected in the **Bootloader Core** component. A GBL Certificate tag in the GBL file indicates to the bootloader that the GBL certificate tag needs to be authenticated using the public key stored in the bootloader certificate. The certificate version in the GBL certificate tag is compared with the bootloader certificate and only a version equal or higher than the bootloader certificate is accepted. Once the GBL certificate tag is authenticated, the GBL file's signature is verified using the authenticated public key from the GBL certificate tag. On EFRxG21 devices, the GBL certificate tag can also be signed by the private key pair of the public key stored in the Secure Engine OTP.

Using Application Image Security Features

This example assumes that a bootloader called **bootloader-uart-xmodem** has been built in Simplicity Studio. For a Series 1 device, three files of interest have been generated in the output directory:

- **bootloader-uart-xmodem.s37** – This file contains the main bootloader. Can be used for bootloader upgrade.
- **bootloader-uart-xmodem-crc.s37** - This file contains the main bootloader with CRC32 checksum.
- **bootloader-uart-xmodem-combined.s37** – This file contains the first stage and main bootloader with a CRC32 checksum in a single image. Can be used for manufacturing and initial deployment of the bootloader.

The relevant version can be flashed to the EFR32 using the Flash Programmer in Simplicity Studio or using Simplicity Commander. For Series 2 devices, only the main bootloader binary is generated.

This example provides two ways of signing the upgrade images. The first option uses Simplicity Commander to generate key material and sign data. This is suitable for development. The second option uses an external signer, such as a dedicated Hardware Security Module (HSM) to protect private key material and perform signing operations. Silicon Labs recommends using an HSM to safeguard private keys.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Generating Keys

To use the security features of the Gecko Bootloader, encryption and signing keys need to be generated. These keys must then be written to the EFR32 device. The encryption key is used with the GBL file for secure firmware upgrade. The signing keys are used both with the GBL file for secure firmware upgrade and to sign the application image for Secure Boot.

Generating a Signing Key Using Simplicity Commander

```
commander util genkey --type ecc-p256 --privkey signing-key --pubkey signing-key.pub --tokenfile signing-key- tokens.txt
```

This creates an ECDSA-P256 key pair for signing; **signing-key** contains the private key in PEM format and **must be kept secret from third parties**. This key will later be used to sign images and GBL files. **signing-key.pub** contains the public key in PEM format and can be used to verify GBL files using commander gbl parse. **signing-key-tokens.txt** contains the public key in token format, suitable for writing to the EFR32 device.

Generating a Signing Key Using a Hardware Security Module

When using a Hardware Security Module, the private key is kept secret inside the HSM. According to the instructions from your HSM vendor, have it generate an ECDSA-P256 key pair and export the public key in PEM format to the file **signing-key.pub**. Then use Simplicity Commander to convert the key to token format, suitable for writing to the EFR32 device.

```
commander gbl keyconvert --type ecc-p256 signing-key.pub --outfile signing-key-tokens.txt
```

Generating an Encryption Key

```
commander util genkey --type aes-ccm --outfile encryption-key
```

This creates an AES-128 key for encryption in the file **encryption-key**. The file has token format, making it suitable to write to the EFR32 device using `commander flash --tokenfile` .

Writing Keys to the Device

To write the two token files containing the encryption key and public key as manufacturing tokens to the device, issue the following command:

```
commander flash --tokengroup znet --tokenfile encryption-key --tokenfile signing-key-tokens.txt
```

Signing an Application Image for Secure Boot

If the bootloader enforces Secure Boot, the application needs to be signed to pass verification. On every boot, a SHA-256 digest of the application is calculated. The signature is verified using ECDSA-P256, with the same public key as for the GBL file signing. Signature verification failure prevents the application from booting.

Application images should contain an **ApplicationProperties_t** struct declaring the application version, capabilities, and other metadata. If **ApplicationProperties_t** is missing, the application image cannot be signed. For more details on adding **ApplicationProperties_t**, see section Application Properties.

Using Simplicity Commander

Signing the application can be done with the command:

```
commander convert myapp.s37 --secureboot --keyfile signing-key --outfile myapp-signed.s37
```

Using a Hardware Security Module

The application can be prepared for signing by issuing the command:

```
commander convert myapp.s37 --secureboot --extsign --outfile myapp-for-signing.s37
```

Creating a Signed and Encrypted GBL Upgrade Image File from an Application

To create a GBL file from an application, use `commander gbl create` .

Copyright © 2025 Silicon Laboratories. All rights reserved.

Note that, as of this writing, secure application images can only be constructed through Simplicity Commander, not through the configuration options available through Simplicity Studio.

Application images should contain an **ApplicationProperties_t** struct declaring the application version, capabilities, and other metadata. If **ApplicationProperties_t** is missing, the application image cannot be signed. For more details on adding **ApplicationProperties_t**, see section Application Properties.

Using Simplicity Commander to Sign

For an application called **myapp.s37**, use:

```
commander gbl create myapp.gbl --app myapp.s37 --sign signing-key --encrypt  encryption-key
```

This single command performs three actions:

- Creates a GBL file
- Encrypts the GBL file
- Signs the GBL file

If Secure Boot is also desired, the application must be signed using `commander convert --secureboot` prior to creating the GBL.

Using a Hardware Security Module to Sign

For an application called **myapp-signed.s37**, which has previously been signed for Secure Boot, use:

```
commander gbl create myapp-for-signing.gbl --app myapp-signed.s37 --extsign --encrypt  encryption-key
```

This command performs the following actions:

- Creates a partial GBL file
- Encrypts the partial GBL file
- Prepares the partial GBL file for signing by an external signer

Using an HSM, sign the output file **myapp-for-signing.gbl.extsign**, and supply the resulting DER-formatted signature file **signature.der** back to Simplicity Commander:

```
commander gbl sign myapp-for-signing.gbl.extsign --signature signature.der --verify signing-key.pub --outfile myapp.gbl
```

The GBL file is not valid until the signature has been applied using gbl sign.

System Security Considerations

The Gecko bootloader security features can be used to create a secure device, but do not create a secure system by themselves. This section goes over considerations that need to be taken when designing a secure system where the Gecko Bootloader is a component.

Key Storage

On Series 1 devices, the decryption and public sign keys used by the Gecko bootloader are stored in the Lockbits page in flash. To prevent a rogue application from being able to change or wipe the keys, the Lockbits page should be write protected after the keys have been written in manufacturing.

On Series 2 devices, the decryption key and the sign key used by the Gecko Bootloader can either be stored in the topmost page of the main flash or in the Secure Engine OTP. The decryption key can be provisioned in the Secure Engine OTP using Simplicity Commander or using the Secure Engine Mailbox interface. The keys are not accessible from the user applications on EFR32xG22 devices, which means that the keys used by Gecko Bootloader need to be stored in the topmost page of the main flash. Once a key value has been programmed into the Secure Engine OTP, it cannot be changed. For more details, refer to Series 2 Secure Boot with RTSL and Simplicity Commander Reference Guide.

The default behavior of the Gecko Bootloader is to use the sign key stored in the Secure Engine OTP to perform the signature verification. If the sign key is not provisioned, the Gecko Bootloader will try to use the public sign key stored in

Copyright © 2025 Silicon Laboratories. All rights reserved.

the topmost page of the main flash. This fallback mechanism can be disabled by disabling the “Allow use of public key from manufacturing token storage” component option from the **Bootloader Core** component in Simplicity Studio.

Write-Protecting the Bootloader

By default, the region in flash used by the Gecko bootloader is readable and writeable. The region needs to stay readable for the Application Interface to be able to interact with the bootloader. Immediately after reset, the region also needs to be writable to be able to perform bootloader upgrades.

Series 1

It is possible to write-protect the bootloader region on EFR32xG12 and newer. This is done by setting the write-once MSC_BOOTLOADERCTRL_BLWDIS bit on every bootup on Series 1 devices. The pages are write-protected until the next reboot. This is done by the Gecko bootloader main stage if the “Prevent bootloader write/erase” component option is enabled in the **Bootloader Core** component in Simplicity Studio.

On EFR32xG1, where the bootloader resides in main flash rather than the information block, the BLWDIS option does not exist. On EFR32xG1, the first flash page containing the first stage bootloader can be write-protected with a Page Lock word, using commander `device protect --write --range 0x0:+0x800`. If bootloader upgrades are to be supported, the pages containing the main bootloader need to stay writeable.

> **Note**: Setting MSC_BOOTLOADERCTRL_BLWDIS will also prevent debuggers from flashing a new bootloader. This means that debug tools that do not completely halt and reset the target device before re-flashing might fail to flash the new bootloader, as the flash is write-protected. If you are unsure whether your debug tools will handle this gracefully, Silicon Labs recommends keeping this setting disabled during development, and enabling it before going into production. If your debug tools halt and reset the target device before flashing, this is not an issue, and Silicon Labs recommends enabling this setting early in the development cycle.

Series 2

On Series 2 devices, the write-once MSC_PAGELOCKx register is used to write-protect the pages used by the bootloader if the “Prevent bootloader write/erase” component option is enabled in the **Bootloader Core** component in Simplicity Studio. The pages are write-protected until the next reboot.

Write-Protecting the Application

On Series 2 devices, it is also possible to write-protect the application. When the component option “Prevent write/erase of verified application” is enabled, the bootloader will write-protect the pages used by the application after successfully verifying the application signature, before allowing the application to start. This option is only available when Secure Boot of the application is enabled. The MSC_PAGE- LOCKx register is used to protect the pages, and the write protection lasts until the next reboot.

Debug Access

Series 1

To prevent debugger access to the Series 1 devices, the Debug Lock word needs to be written. Device recovery after enabling the Debug Lock is possible but will erase the main flash and the Lockbits page. This will erase the main application and the key material stored in the Lockbits page. The Userdata page and bootloader area will survive Debug Unlock, so secrets should not be stored in these locations. To debug lock the device, issue `commander device lock --debug enable`. The AAP Lock can be used to permanently lock the debug port. This also prevents Debug Unlock. To AAP lock the device, please see the reference manual for your device for the address location of the AAP lock word, and use `commander flash --patch` to write the appropriate value to this address.

Series 2

On Series 2 devices, debugger access can be prevented through Secure Engine. Debugger access can be re-enabled after enabling the Debug Lock by issuing device erase. To debug-unlock the device, issue commander device lock --debug

Copyright © 2025 Silicon Laboratories. All rights reserved.

disable from Simplicity Commander. This also erases the main flash, which results in the top page of main flash that stores the encryption and signing keys being erased. To permanently lock the debug port, the device erase disable command can first be issued through Secure Engine. Thereafter, the debug lock command can be issued to the Secure Engine. For more details, refer to Series2 Secure Debug.

Disabling Debug Access from Software

The software APIs `DBG_DisableDebugAccess(dbgLockModeAllowErase)` and `DBG_DisableDebugAccess (dbgLockModePermanent)` from the EMLIB DBG module can be used from the application to lock the debug interface.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Gecko Bootloader and Trustzone

Gecko Bootloader and TrustZone

With GSDK v4.2.2, Gecko Bootloader comes with TrustZone support to build TrustZone aware solutions. This section outlines the major points that should be considered when using a TrustZone aware bootloader.

Gecko Bootloader Operation

A TrustZone aware application can have the following configurations:

- Secure app only
- Non-secure app only
- Secure and Non-secure pair

With TrustZone enabled, the application bootloaders are entirely configured as Secure applications whereas the communication bootloaders are split into Secure and Non-secure application pairs. The communication interfaces are typically in the Non-secure part of the application and the GBL parser, and other functionalities are in the Secure part of the application. TrustZone aware applications are generated and built using pre-defined workspace applications. Refer to Series 2 TrustZone and Software Project Generation and Configuration with SLC-CLI for more details regarding TrustZone workspaces and the SLC CLI tool.

For the communication bootloaders to function properly, a combined bootloader binary image (containing both the Secure and Non-secure binary images) needs to be programmed onto the device. A combined bootloader binary can be obtained by either using the post-build steps defined for the workspace or using the Commander tool. Refer to *UG162: Simplicity Commander Reference Guide* for more details on how to use the Commander tool to combine multiple binaries into a single binary. Application bootloaders need to be built with the **Bootloader TrustZone Secure** component when building a TrustZone aware solution. The **Bootloader TrustZone Secure** component is required to set up the necessary configuration for the application to function correctly within a TrustZone aware solution.

To create a bootloader upgrade file, the combined bootloader binary (Secure and Non-secure) should be used. Refer to Series 2 TrustZone for more details on upgrading an existing application to TrustZone. Special care must be taken while building the TrustZone aware communication bootloaders for the xG21 family as the bootloader size increases from 16kB (non TrustZone aware solution) to 24kB (TrustZone aware solution). Similarly, the Bootloader – SoC Bluetooth Apploader OTA DFU application's size increases from 72kB to 80kB with TrustZone enabled.

Gecko Bootloader Configuration

The following table provides three of the configuration options available as part of the Bootloader Interface with TrustZone enabled.

Configuration Option	Description
BOOTLOADER_DISABLE_OLD_BOOTLOADER_MITIGATION	Disables multi-tiered fallback logic. The fault handling logic as well as the USART auto-detection logic will be disabled. Default value is set to 0.
BOOTLOADER_DISABLE_NVM3_FAULT_HANDLING	Disables peripheral access fault handling. The fault handling triggered by an erroneous access of peripherals will be disabled. This should be disabled if all the peripherals that are in use by the bootloader have been properly configured by the Manually override security state of peripherals option. Default value is set to 0.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configuration Option	Description
BOOTLOADER_MANUAL_OVERRIDE_SECURITY_STATE	Manually override the security state of peripherals in use by the bootloader. This option can be used to manually set a peripheral's access state before calling into the bootloader. Default value is set to 0.

Copyright © 2025 Silicon Laboratories. All rights reserved.

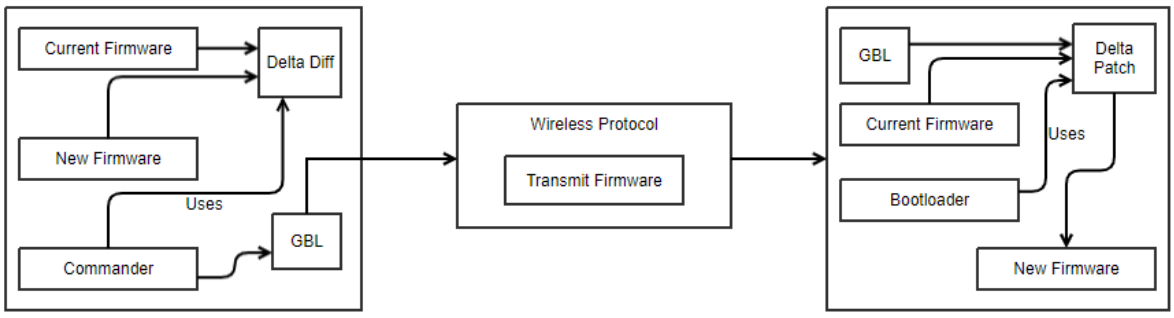
Gecko Bootloader and Delta DFU Upgrades

With GSDK v2024.6.1, Gecko Bootloader support comes with support for delta DFU upgrades. This section provides an introduction to Delta DFU upgrades and describes the major points that should be considered while using the Delta DFU component.

Introduction to Delta DFU

The main function of the image delta DFU is to calculate the difference between the current firmware stored on the device and the new firmware to be updated. This difference is used to create a patch file that can be applied to the current firmware, resulting in the new firmware. The patch file is created using the Simplicity Commander tool, transferred to the device with any wireless protocol as a GBL, and applied on top of the current firmware by the Gecko Bootloader. The patch file creation and application are both implemented in a library that is shared between these tools.

The following diagram illustrates the complete process.



Gecko Bootloader Operation

The Delta DFU upgrades can be enabled by including the Delta DFU component in the Gecko Bootloader. Simplicity Commander supports the creation of a Delta DFU GBL file.

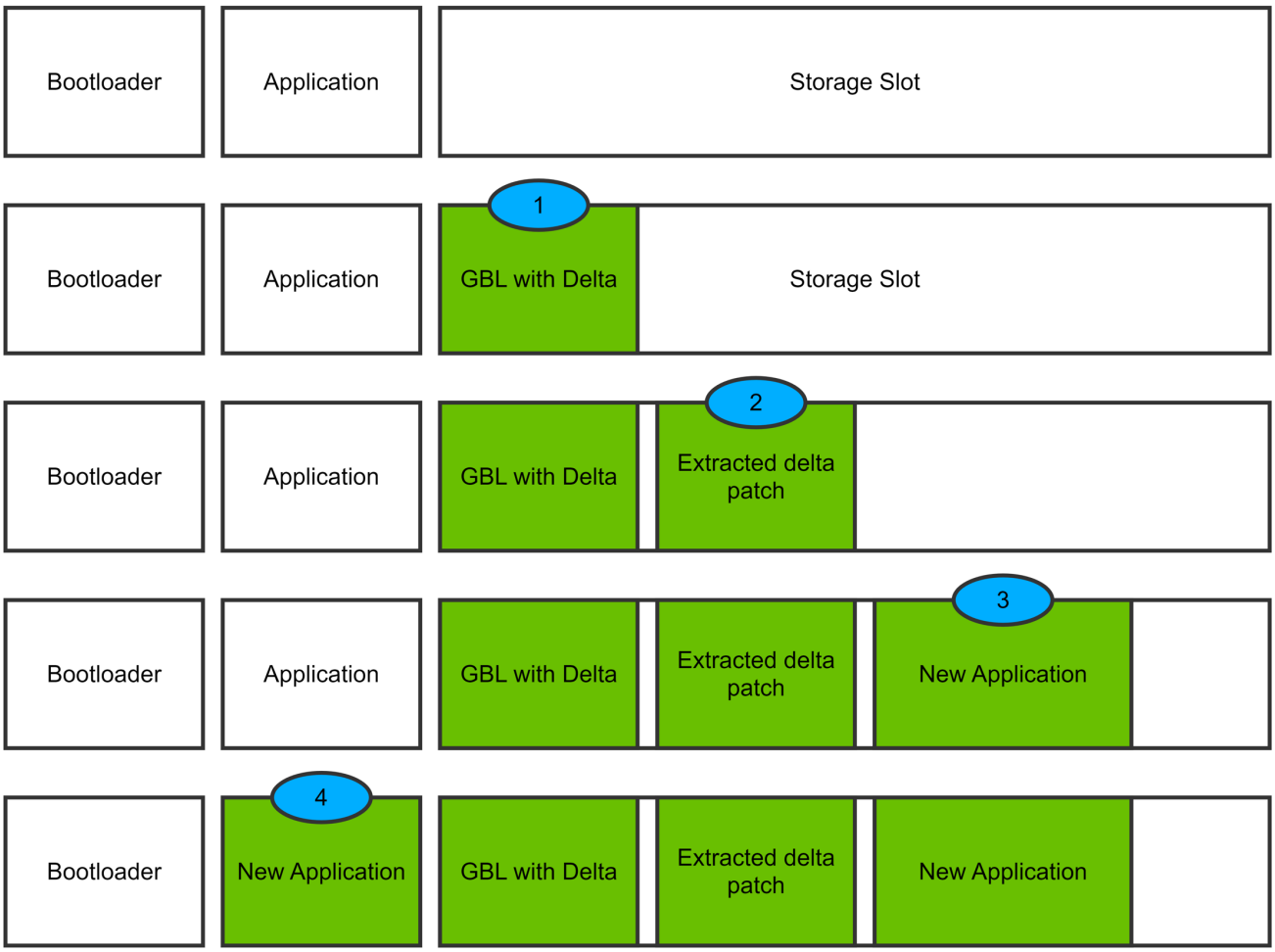
With Delta DFU enabled, the application bootloaders are equipped to accept Delta DFU GBL files. These GBL files contain a patch file that is extracted into the slot. The Gecko Bootloader creates the new firmware using this patch file and the currently running firmware within the slot. The slot size should have enough number of pages to hold the GBL file, patch file, and the new firmware binary in the slot. Each file starts on a new page.

The Delta DFU upgrades are triggered in the same way as in an application bootloader. Once the bootloader finds a GBL that contains Delta Patch, it extracts the patch file and then re-constructs the new firmware file within the slot. If the new firmware is created correctly, then the bootloader copies this new firmware to the application area, completing the upgrade.

To create the GBL file, Commander needs two input files, one of the firmware that is current and one that is the upgrade. For Delta DFU upgrades to work, it is essential that the firmware on the device and the firmware used to create the GBL are the same. Also, the version dependency tag is included in Delta DFU and should be correctly set by the user.

Delta DFU operation is illustrated in the following figure:

Copyright © 2025 Silicon Laboratories. All rights reserved.



Copyright © 2025 Silicon Laboratories. All rights reserved.

Application Interface

Application Interface

The bootloader has an application interface exposed through a function table in the bootloader. The application interface provides APIs to use bootloader functions for storing and retrieving upgrade images and verifying their integrity. APIs to reboot into the bootloader are also provided. For details see the Gecko Bootloader API Reference, shipped with the SDK in the platform/bootloader/documentation folder.

If you are not using a protocol stack from Silicon Labs, the **api/btl_interface.h** header provides the bootloader application interface API. If you are using a protocol stack from Silicon Labs, the recommended bootloader interface API for the specific protocol stack should be used instead. The following files provide the implementation of the bootloader interface:

api/btl_interface.c (common interface)

api/btl_interface_storage.c (interface to storage functionality)

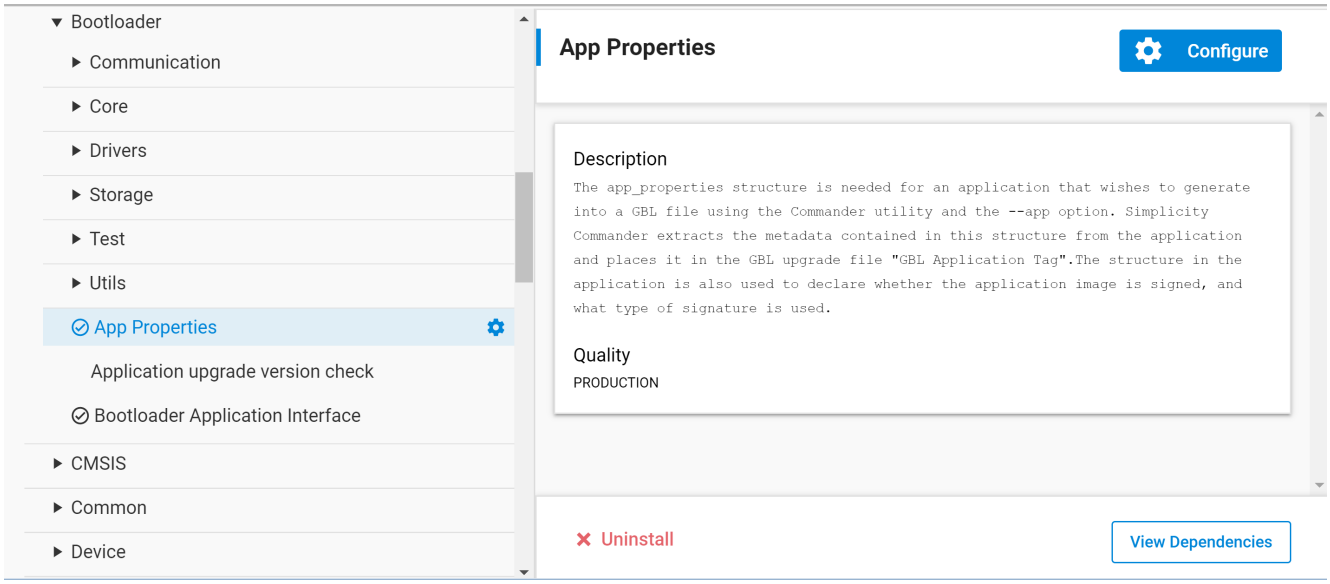
The application interface consists of functions that can be included into the customer application, and that communicate with the bootloader through the **MainBootloaderTable_t**. This table contains function pointers into the bootloader. The 10th word of the bootloader contains a pointer to this structure, allowing any application to easily locate it. Using the wrapper functions provided in the Bootloader Interface API is preferred over accessing the bootloader table directly. Modules include:

- **Application Parser Interface**: Application interface for interfacing with the bootloader image parser.
- **Application Storage Interface**: Application interface for interfacing with the bootloader storage. The Storage Interface is only available on bootloaders that support the storage interface.
- **Common Application Interface**: Generic application interface available on all versions of the bootloader, independently of which components are present.

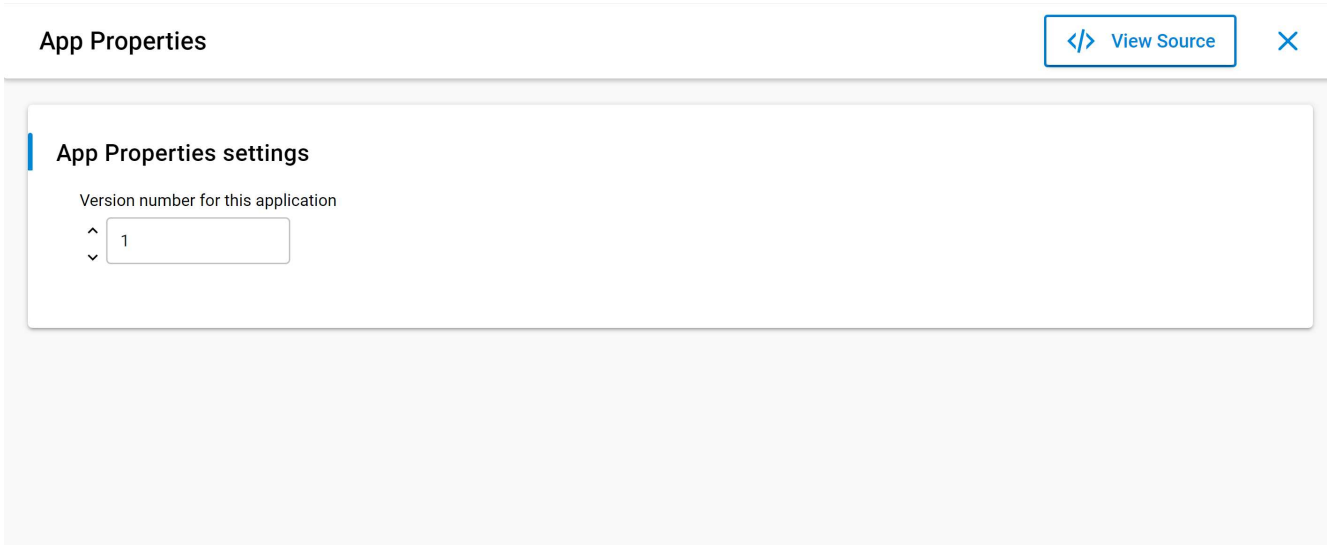
Application Properties

Application images should contain an **ApplicationProperties_t** struct declaring the application version, capabilities, and other metadata. Simplicity Commander extracts the metadata contained in this structure from the application and places it in the GBL upgrade file **GBL Application Tag**. If the structure is not present in the application, Simplicity Commander will raise an error. The **ApplicationProperties_t** struct is added to the application on installing the **bootloader_interface** component to the application. The **bootloader_interface** component installs **bootloader_app_properties** component which adds an instance of **ApplicationProperties_t** named **sl_app_properties** to the project. The component adds a source file named **app_properties.c** and a configuration file named **app_properties_config.h**. This configuration file allows users to configure the application version via Simplicity Studio's Component Editor. To open the Component Editor, locate the **App Properties** component under **Platform > Bootloader** as shown below.

Copyright © 2025 Silicon Laboratories. All rights reserved.



Click **Configure** to open the Component Editor.



The application type is automatically populated based on the wireless stack used in building the project. The value of the application type is automatically set during the project generation step and can be found in **autogen/sl_application_type.h** file.

The contents of the **GBL Application Tag** can be extracted from a GBL file by a running application using the Application Storage interface. Note that the **GBL Application Tag** will only be added if the GBL file contains an application image, not if the GBL file only contains a bootloader upgrade or metadata.

The structure in the application is also used to declare whether the application image is signed, and what type of signature is used. This information is added by Simplicity Commander when signing the image using `commander convert (--secureboot, --extsign or -- signature)`. For the bootloader to locate the **ApplicationProperties_t** struct, if not already done by the linker, Simplicity Commander modifies word 13 of the application to insert a pointer to the **ApplicationProperties_t** struct when signing the application image for Secure Boot.

Error Codes

Copyright © 2025 Silicon Laboratories. All rights reserved.

Most Gecko bootloader APIs return error codes. The following table lists the groups of error codes that may be returned. The full list of error codes within each group can be found in *api/btl_errorcode.h* in the platform/bootloader directory of the SDK, as well as in the API Reference.

ID	Description
0x0	OK
0x01xx	Initialization error
0x02xx	Image verification error
0x04xx	Storage error
0x05xx	Bootload error
0x06xx	Security error
0x07xx	Communication error
0x09xx	XMODEM parser error
0x10xx	GBL file parser error
0x11xx	SPI slave driver error
0x12xx	UART driver error
0x13xx	Compression error

Copyright © 2025 Silicon Laboratories. All rights reserved.

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: 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: Explains how NVM3 can be used as non-volatile data storage in various protocol implementations.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Non-Volatile Data Storage Fundamentals

Non-Volatile Data Storage Fundamentals

NOTE: This section replaces *UG103.07: Non-Volatile Data Storage Fundamentals*. Further updates to this user guide will be provided here.

This guide provides a general introduction to non-volatile data storage using flash, with a focus on the three different dynamic data storage implementations offered for Silicon Labs microcontrollers and radio SoCs (Systems on Chip). It offers a comparison of the three implementations and provides recommendations on when to use each. Additional detail on using the various data storage implementations may be found in the following documents:

- *AN1154: Using Tokens for Non-Volatile Data Storage*
- *AN703: Using Simulated EEPROM Version 1 and Version 2 for the EFR32 SoC Series 1 Platform*
- Using Third Generation Non-Volatile Memory (NVM3) Data Storage

Silicon Labs' *Fundamentals* series covers topics that project managers, application designers, and developers should understand before beginning to work on an embedded networking solution using Silicon Labs chips, networking stacks such as EmberZNet PRO or Silicon Labs Bluetooth, and associated development tools. These guides can be used as a starting place for anyone needing an introduction to developing wireless networking applications, or who is new to the Silicon Labs development environment.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Introduction

Introduction

Non-volatile memory (NVM) is memory that persists even when the device is power-cycled. On Silicon Labs microcontrollers and radio SoCs, the NVM is implemented as flash memory. In many applications the flash is not only used to store the application code but also to store data objects that are written and read frequently by the application. As flash memory can only be erased a limited number of times, several methods exist to efficiently read and write non-volatile data without wearing out the flash.

Some data is considered manufacturing data that is written only once at manufacturing time. This document is concerned with dynamic data that changes frequently over the life of the product.

This document provides an introduction to the main design options for dynamic data storage in microcontrollers and radio SoCs, along with guidelines on what factors affect flash lifetime. In addition it introduces the main flash data storage implementations offered by Silicon Labs:

- NVM3
- Simulated EEPROM version 1 (SimEEv1) and version 2 (SimEEv2)
- Persistent Store (PS Store)

Copyright © 2025 Silicon Laboratories. All rights reserved.

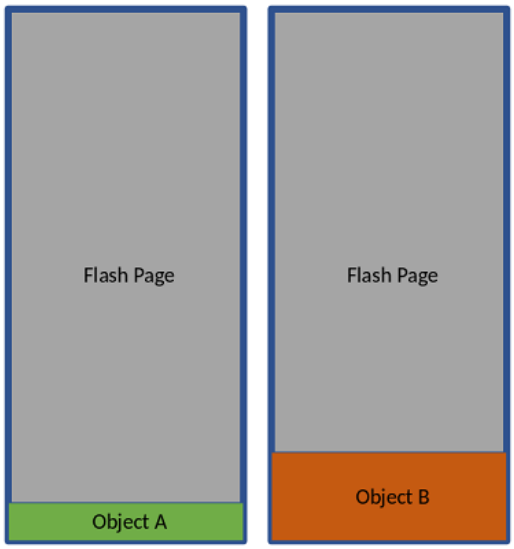
Implementations of Non-Volatile Data Storage

Implementations of Non-Volatile Data Storage

This chapter introduces some of the challenges and design options when implementing non-volatile data storage in flash memory. It describes at a high level how non-volatile data storage is implemented in flash memory in PS Store, SimEEv1/v2, and NVM3.

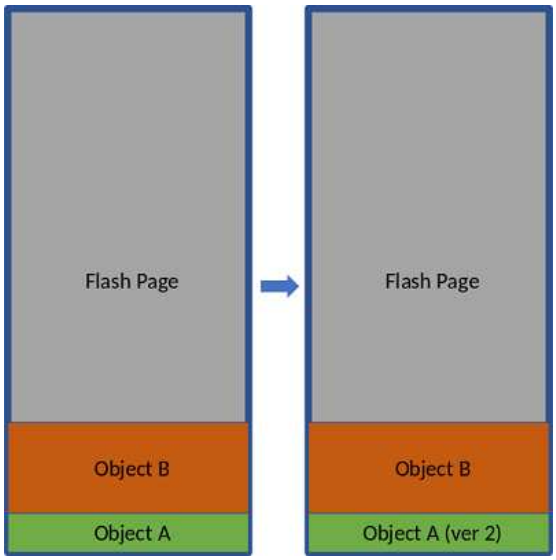
Basic Implementations

One of the characteristics of flash memory is that it is writable in smaller pieces, usually 32-bit words, while it can only be erased in larger chunks, usually pages of several kilobytes. When using flash as data storage, the most straightforward implementation option would be to store each data object in its own flash page, as shown in the following figure. This way each object can be erased and re-written without influencing the other data objects. Usually the data objects are much smaller than the page size, and this solution is not an effective way of using the available flash space.

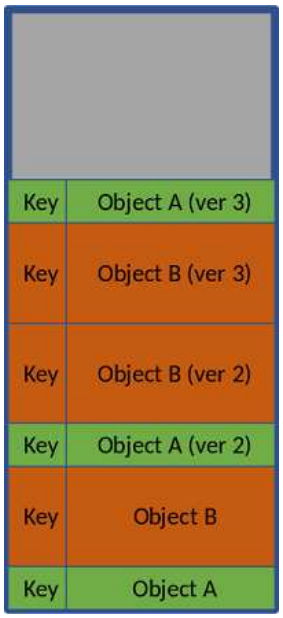


To avoid wasting flash space we can store several data objects in one flash page, as shown in the following figure. This solution then introduces a challenge when we want to write a new value to one of the data objects. In that case the page must be erased before all objects are written back to the page again, including the object we changed. As flash memory can only endure a limited amount of flash erases before the flash cells are worn out, this solution results in a very limited device lifetime.

Copyright © 2025 Silicon Laboratories. All rights reserved.



To avoid erasing the flash page for every object write, we can instead write new versions of each object to new empty locations in the flash page. This is a simple form of wear-levelling that reduces the number of page erases. However, this requires that we store some identification information along with the object data that tells us what object the data belongs to, so we know how to find the latest version of the data object. This is illustrated in the following figure, where a key is added to each version of the object data to identify what object the data belongs to. When accessing an object we then need to search through the flash page for the most recent version of the object. In this case the newest version is the one with the highest address, as we start writing from the lowest address of the page.

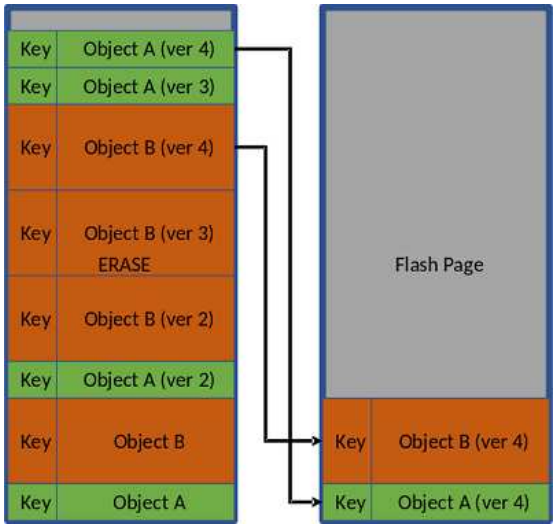


Handling Resets and Power Failures

As we fill up the flash page with new versions of the data objects, eventually no room is left to write new object data. At this point we need to erase the page and start over by writing only the latest versions of each object to the flash page. In many applications, however, power failures or resets can happen at any time, and we should not risk losing data if this occurs. If a reset occurs after the flash page is being erased, but before the data objects are written back, then we will lose this data. To handle this case we introduce a second page, to which we copy the latest version of the data objects, before erasing the original page, as shown in the following figure Then we can start filling the second page with data. When

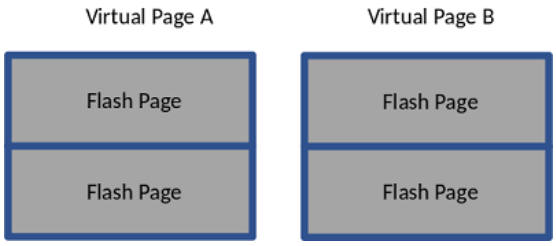
Copyright © 2025 Silicon Laboratories. All rights reserved.

the second page is full we move the latest data back to the first page and so on. This mechanism, where the storage is alternated between two flash pages, is how PS Store operates.



Introducing Virtual Pages

In some applications we write to data objects frequently, and the flash pages therefore also need to be erased frequently. As the data objects in the implementation so far are only spread across two flash pages, each page will frequently get erased and the flash lifetime will be limited. To increase the lifetime we can use more flash pages to store the data objects. In this example, instead of two physical pages, we operate with two virtual pages (A and B) that each consist of several physical flash pages. The virtual pages are erased and written to as if they were one larger flash page. The difference is simply that each virtual page is bigger and we can write more data before we need to erase the virtual page, hence the lifetime is extended. In addition to increasing flash lifetime, using several flash pages per virtual page allows you to store more or larger objects. SimEEv1 uses this design, with each virtual page consisting of two flash pages, A and B, as shown in the following figure.



In some applications the time it takes to write a non-volatile data object must be minimized so as to not interfere with the timing of other critical operations. If an object write is triggered when a virtual page is full, the latest version of all objects must first be copied to the new virtual page before writing the new object version in question to the new page. All objects must be copied over immediately to allow the first page to be quickly erased so we can move data there in case of a failure. Copying all objects at once increases the worst-case object write time.

To reduce the write times, a third virtual page can be introduced that is always kept erased. Instead of copying over the latest version of every object when the first page is full, we can instead copy over only some of the objects. The rest of the objects are copied over as a part of subsequent write operations. This way we spread the copy process to the new page over more write operations, hence each write operation takes less time to complete. With this approach we have live object data spread out across two virtual pages and the third page is always kept erased, so we have somewhere to move the data to in case of a failure. SimEEv2 uses this implementation with 3 virtual pages where each virtual page consists of 6 flash pages.

Copyright © 2025 Silicon Laboratories. All rights reserved.

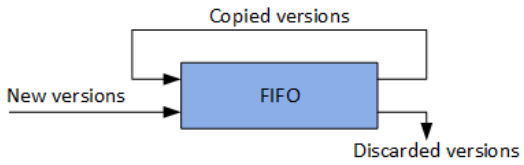
Basic Storage

Basic storage is defined as the size of all the latest version of all objects, including any overhead stored with them. Except for NVM3, every time a flash page or virtual page is erased we must first move the basic storage over to a new page. The basic storage size is important, as it determines how much flash space is left over in a page to store any new versions of the object data. If the basic storage takes up almost the entire page, we can only write a few new object versions before we need to move to a new page and erase the old one. This leads to frequent page erases and a short flash lifetime.

NVM3 only copies unique objects that are stored in the page that will be erased. If newer versions of the object exist in other pages, the object will not be copied.

FIFO Model

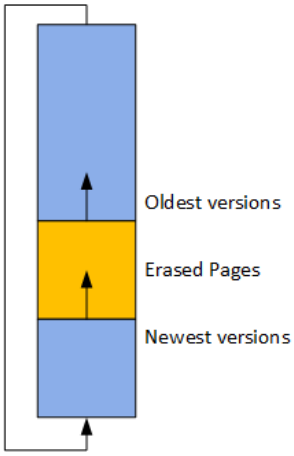
The flash data storage implementations can be modelled as a First-In First-Out (FIFO) buffer (see the following figure), where we write new versions to the input of the FIFO. As the FIFO fills up, we need to free up space by erasing one or more pages at the end of the FIFO. Before we erase a page, we need to copy any object versions for which no newer version of the same object exists in the rest of the flash pages. Other object data can be discarded, because newer versions exist. To maximize flash lifetime we want to copy as few object versions as possible, so that most of the writes to the flash memory are new versions of the objects. If the FIFO is implemented over a large flash space, it is more likely that the new version of the object has been written and that object versions at the end of the FIFO can be discarded. In this case the flash page can be erased with few or no object versions copied.



A drawback of using a few virtual pages is that the available memory for the FIFO is limited to only the virtual pages that can hold live data. For implementations using two virtual pages, like SimEEv1, this means only half the storage space is used for the FIFO, while for SimEEv2 two thirds of the storage space is used for the FIFO.

To allow a higher portion of the storage space to be used for the FIFO, we can instead implement the FIFO as a circular buffer over the entire flash storage space allocated, as shown in the following figure. In this implementation we always need to keep enough erased pages in front of the leading edge of the buffer to write the largest-sized object in case of a failure. When the FIFO fills up to reach the critical number of erased flash pages, we copy over any object versions that have not been superseded and erase the page at the back of the FIFO. This means that, instead of keeping a full virtual page erased, we only need to keep enough space erased to fit the largest-sized object. We can use the rest of the space for the FIFO. NVM3 is implemented as a circular buffer implemented over the entire storage space, thus increasing flash lifetime compared to implementations using smaller virtual pages.

Copyright © 2025 Silicon Laboratories. All rights reserved.



Counter Objects

For some types of data, the storage format can be optimized for the flash medium. EFR32 Series 0 and 1 can write each word two times, allowing for half word updates. Series 2 devices allow just one write per word.

For example, counter values are usually incremented by 1 or some other low value every time they are written. Normally this means that we would have to write the entire counter value in addition to identification bytes every time the counter is incremented. We can optimize this by only storing a start value in addition to the identification value the first time a counter is written. Then we reserve a number of the words following the initial value for writing increments. The flash words in Silicon Labs devices EFR32 Series 0 and 1 can write words twice between each erase by keeping a 1 in the bits that are not to be changed.

As an example, assume we are writing two 16 bit values, 0xAAAA and 0x5555. To safely write them in the same flash word this method can be used:

- Write 0xFFFFAAAA (word in flash becomes 0xFFFFAAAA)
- Write 0x5555FFFF (word in flash becomes 0x5555AAAA)

Note that there is a maximum of two writes to the same word between each erase due to a physical limitation of the flash.

To find the current counter value, we start with the initial value and add the increment values in the halfwords following the initial value. This means that we only need to write one halfword for each increment, instead of the whole counter value and identification value. NVM3 and SimEEv1/v2 support counter objects.

Indexed Objects

When storing data such as arrays in flash, we often only update one index at a time. If we store the whole array as one regular object, we must write the whole array to flash even if just one index is updated. Instead of storing the entire array in one object, we can divide each data array over multiple objects and only update the objects holding the changed array indexes. While it is possible to split arrays into multiple objects manually for all the Silicon Labs storage implementations, SimEEv1/v2 allow all of the indexes to share one object key. The index entry into the array to be looked up is then provided as a separate parameter.

NVM3 does not support reading and writing parts of an object. If the application wants to access indexes individually, each index must be accessed using a unique key.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Dynamic Data Storage Implementations

Dynamic Data Storage Implementations

This chapter introduces some of the dynamic data storage implementations offered by Silicon Labs, including SimEEv1 and SimEEv2, PS Store, and NVM3.

SimEEv1/v2

SimEEv1/v2 can be used with EmberZNet PRO and Silicon Labs Connect (installed with the Flex SDK). SimEEv1 uses two virtual pages with a fixed total size of 8 kB, while SimEEv2 uses three virtual pages with a fixed total size of 36 kB.

One characteristic of SimEEv1/v2 is that all objects are defined with size and type at compile time, hence a new object cannot be created or deleted at runtime.

Silicon Labs provides a plugin for upgrading SimEEv1 data to SimEEv2.

Information about the SimEEv1/v2 implementations is found in *AN703: Using Simulated EEPROM Version 1 and Version 2 for the EFR32 Series 1 SoC Platform*.

PS Store

PS Store can be used with Bluetooth devices on all platforms except for EFR32 Series 2. PS Store API commands are used to manage user data in PS keys in the flash memory of the Bluetooth device. User data stored within the flash memory is persistent across reset and power cycling of the device. The persistent store size is 2048 bytes and uses two flash pages for storage. As Bluetooth bondings are also stored in this area, the space available for user data also depends on the number of bondings the device has at the time. The size of a Bluetooth bonding is around 150 bytes. With its simple implementation and few storage flash pages, PS Store is the smallest of Silicon Labs' non-volatile storage options. PS Store allows objects to be created and deleted at runtime.

Information about the PS Store APIs may be found in the *Bluetooth API Software Reference Manual*.

NVM3

The third generation Non-Volatile Memory (NVM3) data storage driver is an alternative to SimEEv1/v2 and PS Store. The NVM3 driver provides a means to write and read data objects (key/value pairs) stored in flash. Wear-leveling is applied to reduce erase and write cycles and maximize flash lifetime. The driver is resilient to power loss and reset events, ensuring that objects retrieved from the driver are always in a valid state. Because NVM3 can be used with several protocols, such as Bluetooth and EmberZNet PRO, it allows a single data storage instance to be shared in Dynamic Multiprotocol (DMP) applications.

Some of the main features of NVM3 are as follows:

- Key/value pair data storage in flash
- Runtime creation and deletion of objects
- Persistence across power loss and reset events
- Wear-leveling to maximize flash lifetime
- Object sizes from 0 to 4096 bytes
- Configurable flash storage size (minimum 3 flash pages)
- Cache with configurable size for fast object access
- Data and counter object types
- Compatibility layers with token and PS Store APIs provided
- Single shared storage instance in multiprotocol applications
- Repack API to allow application to run clean-up page erases during periods with low CPU load

Detailed information on NVM3 is documented in the EMDRV->NVM3 section of the Gecko HAL & Driver API Reference Guide. Users who are accessing NVM3 through its native API should refer to this API reference guide for information. Users

Copyright © 2025 Silicon Laboratories. All rights reserved.

who are developing dynamic multiprotocol applications should refer to Using Third Generation Non-Volatile Memory (NVM3) Data Storage.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Comparing Non-Volatile Data Storage Implementations

Comparing Non-Volatile Data Storage Implementations

The following table presents an overview of the main features of the various Silicon Labs Non-Volatile Data Storage implementations.

Feature	NVM3	SimEEv1	SimEEv2	PS Store
Compatible devices	EFM32, EFR32	EFR32 Series 1	EFR32 Series 1	EFR32 Series 1
Compatible radio protocols	EmberZNet, Connect, Z-Wave, OpenThread, Bluetooth	EmberZNet, Connect	EmberZNet, Connect	Bluetooth
Flash used for storage	3 or more flash pages	8 KB	36 KB	4 KB
Virtual pages	NA	2	3	NA
Max basic storage (Sum of all object data with overhead)	Variable (see Using Third Generation Non-Volatile Memory (NVM3) Data Storage for details)	2 KB	8 KB	2 KB
Max number of objects	Limited by basic storage	254	254	Limited by basic storage
Max object size (bytes)	Configurable 208-4096	254	254	56
Object creation/deletion	Runtime	Compile time	Compile time	Runtime
Compiled flash size excluding data storage	9.1 KB	3.5 KB	5.4 KB	1.6 KB
Counter object support	Yes	Yes	Yes	No
Indexed object support	Partial (note 1)	Yes	Yes	No
Overhead per object (bytes)	4 (size <= 128 bytes); 8 (size > 128 bytes)	2	6	10
Counter object size including overhead (bytes)	212	60	56	NA
Counter increments before rewrite	100 (EFR32 Series 1); 50 (EFR32 Series 2)	25	25	NA

Note 1: When using NVM3, indexed objects are implemented by storing each index in a separate NVM3 object.

Flash Lifetime

All Silicon Labs Flash Data Storage implementations use some form of wear-levelling to prolong flash lifetime. The effectiveness of the wear-levelling depends on the implementation, the type of data stored, and how often it is updated. The main factors that affect wear-levelling and thereby flash lifetime are:

- Size of flash used for data storage: More flash area gives longer flash lifetime. For NVM3, the number of flash pages used for data storage can be configured, while the rest of the implementations use fixed storage sizes.
- Stored overhead per object: When writing data to the object storage, some overhead bytes are added to identify the data. Implementation with less overhead means the data objects take up less space in flash, and gives longer flash lifetime.
- Alignment to minimum object size: Objects are stored in multiples of the smallest object size. If the data size does not align with this size, padding bytes are added, which adds to the stored data and reduces flash lifetime. For instance, when storing

Copyright © 2025 Silicon Laboratories. All rights reserved.

16-bit objects, NVM3 and PS Store add two extra bytes of padding in addition to the overhead bytes. SimEEv1/v2 are able to store 16-bit data objects without padding.

- Remaining storage after basic storage: For implementations using virtual pages, when switching to a new virtual page one instance of each object is written to the page. The rest of the virtual page can then be used to store new writes of the objects. If a lot of space is used to store one instance of each object, little space is left in the virtual page to use for wear-levelling the subsequent object writes. The flash lifetime will therefore be reduced when the total amount of object data is large relative to the virtual page size. Even for NVM3, where virtual pages are not used, the flash lifetime is limited by the available space of the total NVM3 storage.

To help monitor the actual flash wear, NVM3 and SimEEv1/v2 include function calls for reporting the number of page erases of the data storage flash pages. These erase counters can be read during accelerated lifetime testing of a product to verify if the flash wears at an acceptable rate.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Using Third Generation Non-Volatile Memory (NVM3) Data Storage

Using Third Generation Non-Volatile Memory (NVM3) Data Storage

NOTE: This section replaces *AN1135: Using Third Generation Non-Volatile Memory (NVM3) Data Storage*. Further updates to this application note will be provided here.

The NVM3 driver provides a means to write and read data objects (key/value pairs) stored in flash. Wear-leveling is applied to reduce erase and write cycles and maximize flash lifetime. The driver is resilient to power loss and reset events, ensuring that objects retrieved from the driver are always in a valid state. A single NVM3 instance can be shared among several wireless stacks and application code, making it well-suited for multiprotocol applications. This application note explains how NVM3 can be used as non-volatile data storage in Zigbee (EmberZNet), Open Thread, Z-Wave, Bluetooth, Connect, and WiseConnect applications.

Version 3.x of the Gecko SDK Suite, used with Simplicity Studio 5, introduced a new component-based project architecture that replaced AppBuilder. Eventually all wireless protocols will move to the component-based architecture. This document addresses both this new approach to NVM3 configuration as well as the AppBuilder configuration still in use.

Key Points

- Key/value pair data object storage in flash
- Wear-leveling to maximize flash lifetime
- Resilient to power and reset events
- Shared by Zigbee, Connect, OpenThread, Z-Wave, and Bluetooth stacks
- Compatible with PS Store and Token APIs through wrappers
- Data upgradable from Simulated EEPROM version 2 to NVM3

Copyright © 2025 Silicon Laboratories. All rights reserved.

Overview

Overview

The third generation Non-Volatile Memory (NVM3) data storage driver is for storing persistent data in flash.

NVM3 is designed to work on all Silicon Labs wireless stacks running on EFR32, Series-3, and Si91x devices, as well as MCU applications running on EFM32.

Some of the main features of NVM3 are:

- Key/value pair data storage in flash
- Runtime object creation and deletion
- Persistence across power loss and reset events
- Wear-leveling to maximize flash lifetime
- Object sizes configurable up to 4096 bytes
- Configurable flash storage size (minimum 3 flash pages)
- Cache with configurable size for fast object access
- Data and counter object types
- Compatibility layers included for several Silicon Labs persistent storage APIs
- Single shared storage instance in multiprotocol applications
- Repack API to allow application to run clean-up page erases during periods with low CPU load

NVM3 is described in detail in the NVM3 Documentation on https://docs.silabs.com/. The rest of this document assumes you are familiar with that content.

While the NVM3 API can be used directly, NVM3 is also used as the underlying storage mechanism for several other persistent storage APIs provided by Silicon Labs:

- Token API used in Zigbee EmberZNet and Connect applications
- Persistent Storage API used in Silicon Labs Bluetooth applications

Copyright © 2025 Silicon Laboratories. All rights reserved.

NVM3 Default Instance

NVM3 Default Instance

Several NVM3 instances can be created on a device and live independently of each other, but to save memory it is usually desirable to use only one NVM3 instance as each instance adds some overhead. For applications based on Silicon Labs wireless stacks, a common default instance is used. This allows Dynamic Multiprotocol (DMP) applications that combine several wireless stacks to share the same NVM3 instance.

The number of flash pages used for the NVM3 default instance is configurable, but this setting must match if an application includes several stacks that all use the default instance.

> **IMPORTANT**: When creating an application that includes an NVM3 instance for a device that already contains an NVM3 instance in flash, the number of flash pages configured for the NVM3 instance must match the number of flash pages for the NVM3 instance already found on the device. Therefore it is not possible to change the size of an NVM3 instance once it has been installed on a device, without first erasing the flash pages holding the NVM3 instance and the NVM3 objects stored there.

NVM3 has a cache to speed up access to NVM3 objects. The cache size must be set to a value greater than or equal to the number of objects found in NVM3. This includes the number of NVM3 objects created through the native NVM3 API and any objects created through higher level APIs such as the Token API. The cache must also be large enough to hold any deleted NVM3 objects. The `nvm3_countObjects()` and `nvm3_countDeletedObjects()` functions can be used to find the number of live and deleted objects in NVM3 at any given point. Silicon Labs recommends checking these functions after initialization of all NVM3 objects, both through the native NVM3 API and higher level APIs such as the token API, to figure out the correct size of the NVM3 default cache size.

NVM3 Default Instance Key Space

NVM3 uses a 20-bit key to identify each object. To avoid using the same key for more than one object, the NVM3 key space for the default NVM3 instance has been divided into several domains as outlined in the following table. For example, NVM3 objects defined in the Zigbee EmberZNet stack should use NVM3 keys in the range 0x10000 to 0x1FFFF, while user application tokens should use keys below 0x10000. Note that any user defined NVM3 objects should be placed below 0x10000.

Domain	NVM3 Key
User	0x00000 - 0x0FFFF
Zigbee EmberZNet stack	0x10000 - 0x1FFFF
OpenThread stack	0x20000 - 0x2FFFF
Connect stack	0x30000 - 0x3FFFF
Bluetooth stack	0x40000 - 0x4FFFF
Z-Wave stack	0x50000 - 0x5FFFF
Bluetooth mesh stack	0x60000 - 0x6FFFF
Reserved	0x70000 - 0x7FFFF
Apple HomeKit	0x80000 - 0x80FFF
Zigbee Cluster Library (ZCL)	0x81000 - 0x81FFF
dotdot	0x82000 - 0x82FFF

Copyright © 2025 Silicon Laboratories. All rights reserved.

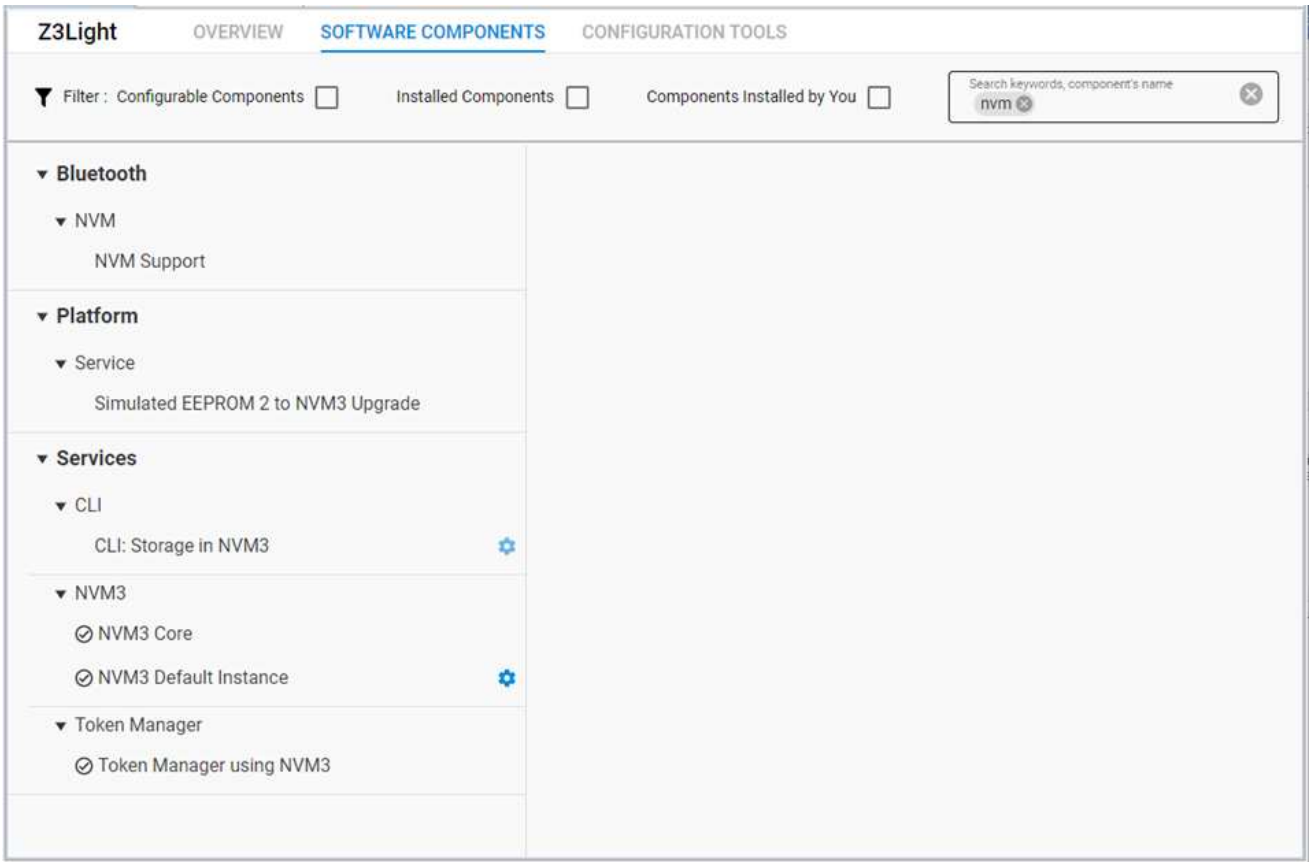
Domain	NVM3 Key
Platform CLI SM	0x83000 - 0x830FF
Platform Crypto	0x83100 - 0x870FF
Bootloader	0x87100 - 0x871FF
Matter	0x87200 - 0x87FFF
Reserved	0x88000 - 0x8DFFF
Static device tokens - override	0x8E000 - 0x8EFFF
Static secure tokens - override	0x8F000 - 0x8FFFF
Wi-SUN	0x90000 - 0x9FFFF
Reserved	0xA0000 - 0xFFFFF

Copyright © 2025 Silicon Laboratories. All rights reserved.

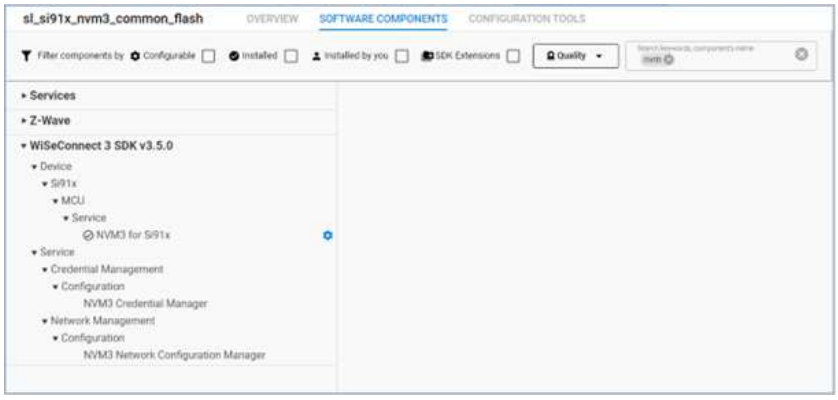
NVM3 in the Simplicity Studio 5 Project Configurator

NVM3 in the Simplicity Studio 5 Project Configurator

The Simplicity Studio 5 Project Configurator, used with Silicon Labs Bluetooth v3.x, Connect v3.x, EmberZNet Zigbee v7.x, and OpenThread SDK applications, includes an NVM3 Core component. A separate component is also provided for the NVM3 default instance, which will initialize this instance.



For Si91x devices, the NVM3 default instance can be configured under NVM3 for Si91x.



Copyright © 2025 Silicon Laboratories. All rights reserved.

The **NVM3 Default Instance** component provides the following configurations:

- **Cache Size**: Number of objects to cache. To reduce access times, this number should be equal to or higher than the number of live and deleted objects stored in NVM3 at any time.
- **Max Object Size**: Size of largest allowed NVM3 object in bytes. Must be between 208 and 4096 bytes.
- **User Repack Headroom**: Headroom determining how many bytes below the forced repack limit the user repack limit is placed. The default value is 0, which means that the forced and the user repack limits are the same.
- **Default Instance Size**: Size of the NVM3 storage region in flash. This must be set to match an integer number of flash pages, 3 pages at minimum.

If the **NVM3 Default Instance** component is included in a project, the default instance will be initialized automatically during `sl\_system\_init()` if the **System Init** component is included in the project.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Using NVM3 with Silicon Labs Connect

Using NVM3 with Silicon Labs Connect

This chapter applies to Connect in the Proprietary Flex SDK v3.x used with the Project Configurator in Simplicity Studio 5. Connect in Proprietary Flex SDK v2.x uses AppBuilder in Simplicity Studio 4. AppBuilder use is described in Using NVM3 in AppBuilder-Based Applications.

For Silicon Labs Connect v3.x applications, a **Token Manager** component is available under Platform > Driver in the Simplicity Studio Project Configurator Software Component view. While the token manager provides the Token API, an additional component must be selected for the token storage backend, either **Token Manager using Sim EEPROM 1**, **Token Manager using Sim EEPROM 2**, or **Token Manager using NVM3.** If NVM3 is chosen as the storage backend, **NVM3** and the **NVM3 Default Instance** components are included in the project.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Using NVM3 with Silicon Labs OpenThread Applications

Using NVM3 with Silicon Labs OpenThread Applications

This section applies to the Silicon Labs OpenThread SDK v1.x used with the Simplicity Project Configurator in Simplicity Studio 5. All OpenThread sample applications in the GSDK by default are configured to use NVM3 to store data in non-volatile memory. When doing so, these applications:

- Use the common default NVM3 instance.
- Include the NVM3 Core and the NVM3 Default Instance components in the project.
- Use the native NVM3 API to access the NVM3 object

The NVM3 key space used by the OpenThread stack is 0x20000 to 0x2FFFF.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Using NVM3 with Zigbee EmberZNet v.7x And Higher

Using NVM3 with Zigbee EmberZNet v7.x and Higher

This chapter applies to the Silicon Labs Zigbee EmberZNet SDK v7.x and higher used with the Simplicity Project Configurator in Simplicity Studio 5. Zigbee EmberZNet 6.x and lower use AppBuilder. AppBuilder use is described in Using NVM3 in AppBuilder-Based Applications.

All Zigbee sample applications in the GSDK by default are configured to use NVM3 to store data in non-volatile memory. When doing so, these applications:

- Use the common default NVM3 instance
- Include the NVM3 Core and the NVM3 Default Instance components in the project
- Use the native NVM3 API to access the NVM3 object

If you have non-volatile data stored in the older SimEE format and need to preserve the data when migrating your Zigbee application to EmberZNet v7.x or higher, then the Token Manager component is available to help manage the data storage. For more information, see the Token Manager component's description.

The NVM3 key space used by the Zigbee stack is 0x10000 to 0x1FFFF.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Using NVM3 with Silicon Labs Bluetooth Applications

Using NVM3 with Silicon Labs Bluetooth Applications

Traditionally the Bluetooth stack uses its own proprietary solution to store data in non-volatile memory, called **Persistent Store (PS Store)**. PS Store stores both data handled by the stack (such as temporary Bluetooth address, bonding keys, and so on) and user data (such as the device state) that has to be preserved on resetting the device. To learn more about PS Store, read the related section of the Bluetooth API Reference Guide.

Some of the sample applications in the Bluetooth SDK are still configured to use PS Store, while others are already configured to use NVM3. This means that:

- On Series 1 devices sample apps are configured to use PS Store, except Bluetooth Mesh NCP sample projects, where NVM3 is used by default.
- On Series 2 devices all sample apps are configured to use NVM3.

While Series 2 devices support NVM3 only, on Series 1 devices both PS Store and NVM3 can be used. This section describes how to configure NVM3 in the Bluetooth SDK, and how to switch between PS Store and NVM3 if needed.

Configuring NVM3 in the Bluetooth SDK

With Simplicity Studio 5 and Bluetooth SDK v3.x

NVM3 can be configured with the Project Configurator as described in NVM3 in the Simplicity Studio 5 Project Configurator.

With Simplicity Studio 4 and Bluetooth SDK v2.x

The Project Configurator is not available to configure NVM3 parameters. Therefore, the parameters have to be defined manually. To overwrite the default parameters:

1. Open the project settings.
2. Find the defined symbols:
    - If you use GCC as your compiler, go to C/C++ Build>Settings>GNU ARM C Compiler>Symbols>Defined Symbols
    - If you use IAR as your compiler, go to C/C++ Build>Settings>IAR C/C++ compiler for ARM>Preprocessor>Defined Symbols
3. Add any of the following defines to overwrite the default parameters:
    - NVM3_DEFAULT_NVM_SIZE
    - NVM3_DEFAULT_CACHE_SIZE
    - NVM3_DEFAULT_MAX_OBJECT_SIZE
    - NVM3_DEFAULT_REPACK_HEADROOM

You can find the description of each parameter in NVM3 in the Simplicity Studio 5 Project Configurator. Be careful when providing the size of NVM3, as it must be a multiple of the flash page size. Note: On Series 1 devices the flash page size is typically 2 kB, while on Series 2 devices the flash page size is typically 8 kB. Check the Wireless Gecko Reference Manual for your device. The NVM size must be at least 3 flash pages.

Switching from PS Store to NVM3

Beginning with Bluetooth SDK v2.13.0, both PS Store and NVM3 are supported as non-volatile memory solutions on Series 1 devices. Most sample applications are configured to use PS Store by default, but for some applications (where larger non-volatile memory is needed) NVM3 may be a better solution.

> **Note**: PS Store and NVM3 are not compatible with each other, therefore upgrading an already existing application from PS Store to NVM3 will result in losing all data stored on the device. If you have an application

Copyright © 2025 Silicon Laboratories. All rights reserved.

> running in the field, it may be wiser to stay with PS Store. If you still want to upgrade, see Using the Gecko Bootloader with the Silicon Labs Bluetooth Application for details.

> **Note**: PS Store uses only 2 flash pages (=4 kB on an EFR32BG1/12/13 device). Therefore, changing to NVM3 will affect the available space in flash. You must be particularly careful when you upgrade the firmware not to overwrite the NVM3 area with the application.

With Simplicity Studio 5 and Bluetooth SDK v3.x

To change your project configuration from PS Store to NVM3, simply install the **NVM3 Default Instance** component in the Project Configurator as discussed in NVM3 in the Simplicity Studio 5 Project Configurator. This will automatically uninstall the (otherwise hidden) PS Store component.

With Simplicity Studio 4 and Bluetooth SDK v2.x

To change your project configuration from PS Store to NVM3, use the following procedure.

1. Copy the following folder with all of its content:
   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\platform\emdrv\nvm3` under the `/platform/emdrv` folder of your project.
2. Remove the following files from the project:
   - `/platform/emdrv/nvm3/src/nvm3_hal_extflash.c`
   - `/platform/emdrv/nvm3/src/nvm3_default_extflash.c` (NVM3 use with external flash is deprecated)
3. If you use Apploader in your project, also copy the NVM3 version of Apploader from:
   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\lib\<device>\<compiler>\binapploader_nvm3.o`
   into the `/protocol/bluetooth/lib/<device>/<compiler>` folder of your project.
4. **If you use GCC as a compiler:**
   1. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Compiler > Includes`.
   2. Add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` to the include directory.
   3. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Linker > Miscellaneous`.
   4. Remove `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/libpsstore.a}`.
   5. Add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_gcc.a}`.

   If you use IAR as a compiler:
   1. Go to `Project > Properties > C/C++ Build > Settings > IAR C/C++ Compiler for ARM > Preprocessor`.
   2. Add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` to the include directory.
   3. Go to `Project > Properties > C/C++ Build > Settings > IAR Linker for ARM > Library`.
   4. Remove `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/\<device\>/\<compiler\>/libpsstore.a}`.
   5. Add `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_iar.a}`.
5. If you use Apploader, also modify:
   `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/\<device\>/\<compiler\>/binapploader.o}`
   to
   `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/\<device\>/\<compiler\>/binapploader_nvm3.o}`
6. Configure NVM3 as described in Configuring NVM3 in the Bluetooth SDK.

Switching from NVM3 to PS Store

It may happen that, for a reason such as backward compatibility, you have to change the configuration from NVM3 to PS Store.

With Simplicity Studio 5 and Bluetooth SDK v3.x

Copyright © 2025 Silicon Laboratories. All rights reserved.

To change your project configuration from NVM3 to PS Store, simply uninstall the NVM3 Default Instance component. This will automatically install the (otherwise hidden) PS Store component. Note, that this can only be done on series 1 devices, as series 2 devices do not support PS Store. **Also, it is not possible with Bluetooth Mesh Stack in use**.

With Simplicity Studio 4 and Bluetooth SDK v2.x

To change your project configuration from NVM3 to PS Store, use the following procedure:

1. Remove the /platform/emdrv/nvm3 folder from your project.
2. Copy the PS Store library from:
   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\lib\<device>\<compiler>\libpsstore.a`
   into the `/protocol/bluetooth/lib/<device>/<compiler>` folder of your project.
3. If you use Apploader in your project, also copy the PS Store version of Apploader from:
   `C:\SiliconLabs\SimplicityStudio\v4\developer\sdks\gecko_sdk_suite\<version>\protocol\bluetooth\lib\<device>\<compiler>\binapploader.o`
   into the `/protocol/bluetooth/lib/<device>/<compiler>` folder of your project.

> **Note**: PS Store is not supported on Series 2 devices (EFR32xG2x), therefore there is only an NVM3 version of the Apploader for these devices.

4. **If you use GCC as your compiler**
   1. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Compiler > Includes`.
   2. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` from the include directory.
   3. Go to `Project > Properties > C/C++ Build > Settings > GNU ARM C Linker > Miscellaneous`.
   4. Add `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/libpsstore.a}`.
   5. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_gcc.a}`.

   If you use IAR as your compiler
   1. Go to `Project > Properties > C/C++ Build > Settings > IAR C/C++ Compiler for ARM > Preprocessor`.
   2. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/inc}` from the include directories.
   3. Go to `Project > Properties > C/C++ Build > Settings > IAR Linker for ARM > Library`.
   4. Add `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/libpsstore.a}`.
   5. Remove `${workspace_loc:/${ProjName}/platform/emdrv/nvm3/lib/libnvm3_CM4_iar.a}`.
5. If you use Apploader, also change:
   `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/binapploader_nvm3.o}`
   to
   `${workspace_loc:/${ProjName}/protocol/bluetooth/lib/<device>/<compiler>/binapploader.o}`.

Copyright © 2025 Silicon Laboratories. All rights reserved.

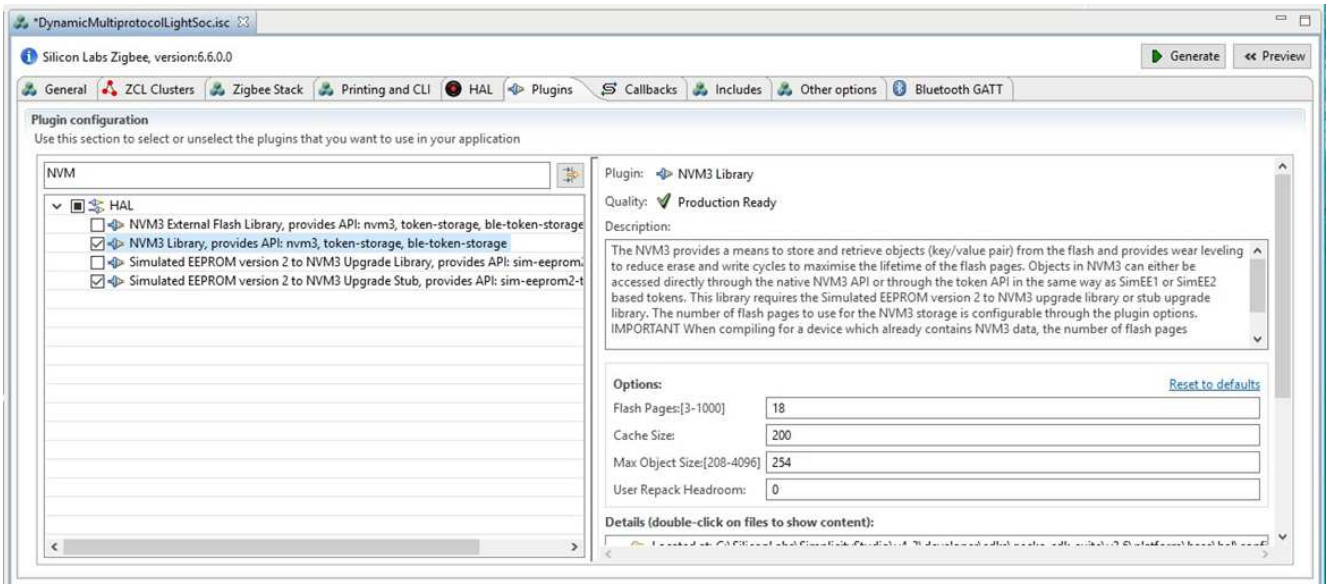
Using NVM3 In Appbuilder Based Applications

Using NVM3 in AppBuilder-Based Applications

This section explains how NVM3 can be used for non-volatile storage in AppBuilder-based applications like EmberZNet 6.10.x and lower, EmberZNet-based DMP applications in GSDK 3.2 and lower, and Connect applications in Gecko SDK Suite 2.7 and lower.

NVM Library Plugin

To use NVM3 with an AppBuilder-based example application, the NVM3 Library plugin should be included in the project. All PS Store and SimEE plugins should be deselected.



The **NVM3 Library** plugin offers four plugin options:

- **Flash Pages**: Number of flash pages to use for NVM3 data storage. Must be 3 or higher. Default is 18 (36KB) for EFR32 Series-1 devices and 4 (32KB) for EFR32 Series-2 devices
- **Cache Size**: Number of objects to cache. To reduce access times, this number should be equal to or higher than the number of objects, including tokens and deleted objects, stored in NVM3 at any time.
- **Max Object Size**: Size of largest allowed NVM3 object in bytes. Must be between 208 and 4096 bytes. Note that the token API can only be used to access objects of 254 bytes or smaller. When accessing larger objects, the native NVM3 API must be used.
- **User Repack Headroom**: Headroom determining how many bytes below the forced repack limit the user repack limit is placed. The default value is 0, which means that the forced and the user repack limits are the same.

When the NVM3 Library plugin is used the **Simulated EEPROM version 2 to NVM3 Upgrade Library** or **Simulated EEPROM version 2 to NVM3 Upgrade Stub Library** must be included, as described in SimEEv2 to NVM3 Upgrade Plugin.

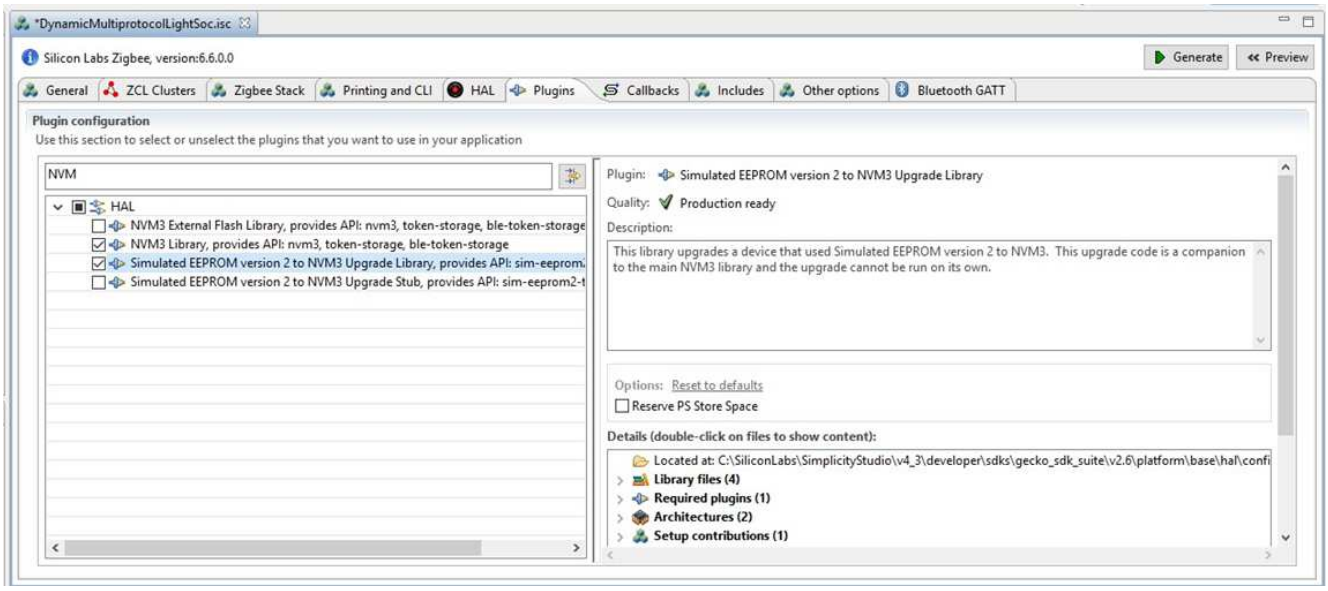
SimEEv2 to NVM3 Upgrade Plugin

An AppBuilder plugin (**Simulated EEPROM version 2 to NVM3 Upgrade Library**) is provided for EmberZNet applications that upgrade tokens stored in SimEEv2 to NVM3. For tokens to be successfully upgraded to NVM3, CREATOR_* and NVM3KEY_* defines must be added for all tokens as described in Token API. The upgrade plugin will replace the SimEEv2 storage in-place with an NVM3 storage instance. The plugin does this by compacting the SimEEv2 storage down to 12 kB,

Copyright © 2025 Silicon Laboratories. All rights reserved.

and then creates an NVM3 instance in the remaining 24 kB of the original 36 kB SimEEv2 storage space. After the token data has been copied over from SimEEv2 to NVM3, the SimEEv2 storage is erased and the NVM3 instance is resized to use the entire 36 kB storage space. Apart from the code space needed for the upgrade library code, the upgrade does not require any additional flash space to the 36 kB storage area. The upgrade plugin requires that the existing SimEEv2 storage space and new NVM3 storage space are located at the same address and have the same size.

The **Simulated EEPROM version 2 to NVM3 Upgrade Library** plugin should be included to enable the upgrade as shown in the figure below. If no SimEEv2 token data is found, the upgrade plugin will look for NVM3 data, and if neither is found it will create a new NVM3 instance with tokens set to their default values. For applications that do not need to upgrade any SimEEv2 tokens, the **Simulated EEPROM version 2 to NVM3 Upgrade Stub** plugin should be included instead.



Copyright © 2025 Silicon Laboratories. All rights reserved.

Using NVM3 with Z Wave Applications

Using NVM3 with Z-Wave Applications

For details on using NVM3 with Z-Wave applications, see https://www.silabs.com/documents/public/user-guides/INS14259.pdf, section 7.4.

Copyright © 2025 Silicon Laboratories. All rights reserved.

NVM3 API Options

NVM3 API Options

This chapter describes the three different APIs available to access NVM3 objects.

- Token API
- Persistent Store API
- Native NVM3 API

Token API

The token API is used to access data stored in SimEEv1 and SimEEv2 with the EmberZNet and Connect stacks, as well as multiprotocol applications. Information on how to define and access tokens can be found in *AN1154: Using Tokens for Non-Volatile Data Storage*, and users should read this document before using the token API. When selecting the NVM3 Library plugin instead of one the SimEE plugins, the NVM3 default instance is used to store the token data instead of SimEE. The same token API can still be used to access tokens stored in NVM3, but the token definition needs some modifications to work with NVM3, as described below.

When defining a token to be used with SimEE, a creator code must be defined as an identifier for the token. Similarly, when defining a token to be used with NVM3, an NVM3 key must be defined for the token. A token definition that is compatible with both NVM3 and SimEE would include both an NVM3 key and a creator code and look like this:

```
#define CREATOR_name 16bit_value
#define NVM3KEY_name 20bit_value
#ifdef DEFINETYPES
  typedef data_type type #endif
#ifdef DEFINETOKENS
DEFINE_*_TOKEN(name, type, ... ,defaults) #endif
```

Select a 20-bit NVM3 key for the token, according to the domains in NVM Default Instance Key Space. Each token must have a unique NVM3 key, except for indexed tokens, where more NVM3 keys must be reserved as outlined in Special Considerations for Indexed Tokens.

Deleting Tokens

As tokens are created at compile time, they cannot be created or deleted at run time. NVM3 objects are, however, created and deleted at run time, and the token initialization function creates NVM3 objects for each defined token if they do not already exist. The token initialization generally does not delete NVM3 objects found that do not have a corresponding token associated with them. Therefore, if a token is no longer included in an application, the application should manually delete the associated NVM3 object by using the NVM3 Native API described in Native NVM3 API. For indexed tokens, however, the token initialization checks if indexed tokens have more or less indexes than the number of NVM3 objects found in the indexed token's NVK3KEY range. If there are fewer indexes, the token initialization deletes the extra NVM3 objects. If the number of indexes has been increased, new NVM3 objects will be created to hold these indexes.

When NVM3 objects are deleted, the actual object data remains in NVM3 but is marked as deleted. The deleted object data remains in NVM3 and consumes cache space until NVM3 repacks have erased the page(s) holding all versions of these objects.

Special Considerations for Indexed Tokens

NVM3 does not have native support for indexed tokens. Therefore an extra requirement is imposed on the NVM3 key selection for indexed tokens. With NVM3, indexed tokens are implemented by storing each index in a separate object, starting with index 0 stored at the defined NVM3KEY_name key value and the last index (127) stored with key NVM3KEY_name + 127. Because of this implementation, 128 NVM3 keys must be reserved for each indexed token. The user still only defines one NVM3KEY_name key value, but no other tokens should be defined with key values in the 127

Copyright © 2025 Silicon Laboratories. All rights reserved.

values following this defined key. Even if the token is defined with fewer than 128 indices, all 128 indices should be reserved as the token might be expanded with more indices later on.

The example below shows two indexed tokens defined in the user key domain:

```
// This key is used for an indexed token and the subsequent 0×7F keys are also reserved #define
NVM3KEY_MY_INDEXED_TOKEN_A 0×00000
// This key is used for an indexed token and the subsequent 0×7F keys are also reserved #define
NVM3KEY_MY_INDEXED_TOKEN_B 0×00080
```

The table below provides indexed token NVM3 key selections in the example.

NVM3KEY	NVM3 Objects Contents
0x00000	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 0
0x00001	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 1
0x00002	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 2
...	
0x0007F	Reserved for TOKEN_MY_INDEXED_TOKEN_A index 127
0x00080	Reserved for TOKEN_MY_INDEXED_TOKEN_B index 0
0x00081	Reserved for TOKEN_MY_INDEXED_TOKEN_B index 1
...	
0x000FF	Reserved for TOKEN_MY_INDEXED_TOKEN_B index 127

Bluetooth NVM API

The Bluetooth NVM API that was originally designed for PS Store can be used in the same way when using NVM3 as when using PS Store. The Bluetooth stack will automatically translate its NVM API calls to PS Store API calls or to NVM3 API calls in the background, depending on what components the project uses. The same applies for Zigbee + Bluetooth DMP projects, where NVM3 is applied as the storage mechanism, but the Bluetooth NVM API can still be used. The Bluetooth API is documented in the Bluetooth API Reference Manual. In Bluetooth SDK v2.x it can be found under the *Flash* class (commands starting with `gecko_cmd_flash_`), while in Bluetooth SDK v3.x it can be found under the *NVM* class (commands starting with `sl_bt_nvm`).

16-bit keys are used with the Bluetooth NVM API, which are then mapped to a 20-bit NVM3 key when NVM3 is used as storage mechanism. The four most significant bits are set to 0x4 to place these objects in the Bluetooth domain of the NVM3 default instance key space. As the Bluetooth NVM API is fixed to use only the Bluetooth domain, any objects to be placed in other domains, for example the User domain, must be created and accessed using the native NVM3 API.

If you want to use the Bluetooth NVM API and the native NVM3 API in the same app, then:

1. Call `gecko_init(pconfig)` to initialize the Bluetooth stack and open its own NVM3 instance. This is done in all Bluetooth sample apps.
2. Open NVM3 by calling the `nvm3_open()` function with the default (!) parameters to open your NVM3 instance: `nvm3_open(nvm3_defaultHandle, nvm3_defaultInit);`

User data can now be saved to:

- PS key range 0x4000 - 0x407F. All other PS keys (0x0000-0xFFFF) are reserved for the stack (for example for storing bonding data).
- NVM3 key range 0x00000-0x0FFFF (NVM3 user data area), and 0x44000-0x4407F (PS Store user data area).

For example, the following API calls will have the same effect (writing to the same area):

- `nvm3_writeData(nvm3_defaultHandle,0x44000,(void\*)data,len);`
- `gecko_cmd_flash_ps_save(0x4000,len,data); //in Bluetooth SDK v2.x`
- `sl_bt_nvm_save(0x4000,len,data); //in Bluetooth SDK v3.x`

Similarly, you can read the same data with the following API calls:

Copyright © 2025 Silicon Laboratories. All rights reserved.

- `nvm3_readData(nvm3_defaultHandle,0x44000,(void\*)read_buffer,maxlen);`
- `gecko_cmd_flash_ps_load(0x4000); //in Bluetooth SDK v2.x`
- `sl_bt_nvm_load(0x4000,maxlen,&read_len,(uint8_t\*)read_buffer); //in Bluetooth SDK v3.x`

Native NVM3 API

For code accessing NVM3 objects that does not need to be compatible with the token or PS Store APIs, using the native NVM3 API to access NVM3 data is recommended to reduce code size and allow using the full feature set of NVM3. Any PS Store object or token can also be accessed through the native NVM3 API using the same NVM3 key. Complete documentation of this API is found online in the Drivers section of the Gecko Platform Developer Documentation. If you are using an earlier version of the GSDK, search that page's version history for the corresponding release.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Simplicity Commander and NVM3

Simplicity Commander and NVM3

Simplicity Commander is a single, all-purpose tool to be used in a production environment. It is invoked using a simple CLI (Command Line Interface) that is also scriptable. Simplicity Commander supports reading out the NVM3 data area from a device and parsing the NVM3 data to extract stored values. This can be useful in a debugging scenario where you may need to find out the stored state of an application that has been running for some time.

More information on how to use the Simplicity Commander with NVM3 can be found in Simplicity Commander Reference Guide.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Securing NVM3 Data on External Flash Devices

Securing NVM3 Data on External Flash Devices

Series-3 devices do not have on-die NVM. Since off-chip memory such as external flash is vulnerable to persistent attacks, data objects are securely stored and retrieved from the external flash using AES-GCM authenticated encryption. On Series-3 devices, the NVM3 AES-128 symmetric key is used to encrypt the data, ensuring it cannot be read in plain text from the external flash. The NVM3 AES-128 symmetric key is a device-generated, non-exportable key stored in the Secure Engine MTP. The NVM3 symmetric key is invalidated or erased during a device erase. Any attempt to load previously recorded NVM3 data (encrypted with the older key) into flash will result in invalid reads and must be avoided. In the case of a device erase, it is essential to reinitialize the NVM3 instance and add the objects again. For details about DFA and DPA countermeasures and the AES engines being used, refer to Secure Key Storage.

Impact of External Flash and Security on Repack Timings

For Series-3 devices using external flash, repack operations may take longer due to the increased latency of accessing external flash over the bus. Additionally, with security features such as authenticated encryption (AES-GCM) enabled, the repack timings may be further impacted due to the cryptographic overhead. It is recommended to schedule repack operations during periods of low CPU and bus activity to minimize the impact on application performance.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Best Practices and Optimizations For NVM3

Best Practices and Optimizations for NVM3

Right Sizing the NVM3 Storage Region in Flash

To minimize repacks and reduce flash wear, it is important to configure the NVM3 region size appropriately. The size of the NVM3 region should be based on:

- The number of objects stored in NVM3.
- The size of each object, including metadata overhead.
- The frequency of object updates and deletions.
- The flash page size of the device.

With the addition of security on Series 3 devices, storing a data object incurs a size overhead of 8 bytes. Ensure that the NVM3 region has sufficient space to accommodate active objects, deleted objects, and spare pages for repack operations. A larger NVM3 region reduces the frequency of repacks, thereby extending the flash lifetime. The "Maximum allowed basic storage" section at https://docs.silabs.com/gecko-platform/latest/platform-driver/nvm3#max-allowed-basic-storage-with-4-k-b-page-size can help with configuring the appropriate number of flash pages and right-sizing the NVM3 storage in flash.

Right Sizing the Cache

The NVM3 cache size should be configured to match the number of live and deleted objects in the NVM3 region. This ensures faster access to objects and reduces the overhead of searching for objects in flash. Users can query the cache size and memory information using the `nvm3_getMemInfo()` API. This API provides details about the current memory state, including available memory, cache status, and additional cache requirements. Properly sizing the cache improves performance and reduces object lookup latency.

Low Memory Notification Callback

Users can use the `nvm3_registerCallback()` API to register a callback function, which is triggered when the NVM3 instance detects low memory conditions or a cache overflow. The callback allows the application to handle conditions such as freeing up memory by triggering repack. It is recommended that the callback be registered before calling any NVM3 APIs. Additionally, users can deregister the callback using the `nvm3_deregisterCallback()` API.

NVM3 Optimizations

Enabling NVM3 optimization improves the NVM3 initialization and object lookup time. NVM3 optimization can be enabled or disabled from Simplicity Studio UC for Series-2 and Series-3 devices. The optimization support has implications on code size. With optimization support enabled, the code size increases by approximately 1248 bytes. NVM3 optimization is enabled by default for Series-3 devices and disabled for Series-2 devices.

By following these best practices and leveraging the available APIs and optimizations, users can maximize the efficiency and reliability of the NVM3 storage system while maintaining high performance for their applications.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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: 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

IoT Endpoint Security Fundamentals

IoT Endpoint Security Fundamentals

NOTE: This section replaces *UG103.05: IoT Endpoint Security Fundamentals*. Further updates to this user guide will be provided here.

This guide 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, this guide clearly delineates the solutions Silicon Labs provides to support endpoint security and what you must do outside of the Silicon Labs framework. Where appropriate, Silicon Labs' approach to our own security is offered as an example. This guide is designed for product developers and managers.

Silicon Labs' *Fundamentals* series covers topics that project managers, application designers, and developers should understand before beginning to work on an embedded networking solution using Silicon Labs chips, networking stacks such as EmberZNet PRO or Silicon Labs Bluetooth, and associated development tools. These guides can be used as a starting place for anyone needing an introduction to developing wireless networking applications, or who is new to the Silicon Labs development environment.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Overview

Overview

Securing the IoT is challenging. It is also mission-critical. Threats are continuously evolving, and the demand on product developers to keep up can be burdensome – particularly in low-cost, resource-constrained IoT products. Protecting your product in a connected world is a necessity, as customer data and modern online business models are increasingly targets for costly hacks that jeopardize end-user privacy and corporate brand damage. Silicon Labs is committed to working with the security community, customers, and other experts to bring state-of-the-art technology to help protect your connected portfolio.

Silicon Labs is a member of the ioXt (Internet of Secure Things) Alliance. The ioXt Alliance was formed to bring together wireless carriers, leading consumer product manufacturers, standards groups, compliance labs and government organizations to align baseline security requirements, to set the stage for testing and compatibility certification, and to work together building global standards for the IoT world.

The ioXt alliance has produced the ioXt Security Pledge (https://www.ioxtalliance.org/s/ioXt-SecurityPledge-booklet-final.pdf) The pledge covers eight principles in the areas of Security, Upgradability, and Transparency. Silicon Labs has adopted these principles in our own operations as well as in the products we provide. Our approach to these principles is described in this document.

Copyright © 2025 Silicon Laboratories. All rights reserved.

The above image and all pledge language is reproduced from *The ioXt Security Pledge: 8 Principles for Consumer Product Design and Manufacturing to Ensure Security, Upgradability & Transparency (2019)*.

Copyright © 2025 Silicon Laboratories. All rights reserved.

No Universal Passwords

No Universal Passwords

The product shall not have a universal password; unique security credentials will be required for operation. Universal passwords allow an attacker to easily gain access to any device. Therefore, products shall either have a unique password or require the user to enter a new password immediately upon first use.

It is your responsibility to ensure that your product enforces the creation of a unique password before activation.

Silicon Labs' products are designed to be configured by the manufacturer before being delivered to customers, and therefore passwords are outside of our scope. However, Silicon Labs tools are designed to support the various levels of security provided by the protocol in question. Most protocols offer different security levels, with tradeoffs between security level and other features such as ease of network formation. You need to review and decide on the level required by your application. For example:

- The EmberZNet Pro SDK supports a highly secure centralized trust-center-controlled method that replaces a device's factory-programmed link key with a key that is unique to each device on the network.
- Z-Wave 700 products come with a factory-programmed unique S2 keypair on first power-up, and support SmartStart commissioning through a package QR code containing the public key.
- Bluetooth options range from an unsecured "Just Works" approach to a LE Secure Connections Pairing model. Application designers can implement additional device authentication methods, such as through the companion smartphone app, to help ensure secure pairing even for devices without a user interface.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Secured Interfaces

Secured Interfaces

All product interfaces shall be appropriately secured by the manufacturer.

The interfaces to be secured will vary by product configuration. For example, in an NCP topology the NCP interface must be secured. Debug interfaces should always be locked. Wireless interfaces should be secured by using strong pairing and commissioning methods and by enabling encrypted and authenticated transmissions.

While securing the interfaces is in the end your responsibility, Silicon Labs provides the tools to enable that security.

Both Series 1 and Series 2 devices are designed to support securing debug access. For Series 1 devices, that functionality is provided through writing a Debug Lock word to the device. Unlocking the device erases the main application and the key material stored in the Lockbits page. For Series 2 devices, securing debug access is done through the device's Secure Engine. Both allow the developer to lock the debug port itself. See Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher, Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices), or *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.2 and Lower* for an overview of securing debug access, and Series 2 Secure Debug for details on the Series 2 implementation. *UG104: Testing and Debugging Applications for the Silicon Labs EFR32MG Platforms* provides an overview of the various application testing stages and the debug access (hardware and software) required in each.

For more information on Wireless interface security in the different protocols, see the following:

- Zigbee Security
- Bluetooth LE Fundamentals and relevant KBAs
- *AN1037: Apple HomeKit Over Bluetooth®*
- *UG235.03: Architecture of the Silicon Labs Connect Stack v2.x*
- *UG435.03: Architecture of the Silicon Labs Connect Stack v3.x*

Copyright © 2025 Silicon Laboratories. All rights reserved.

Proven Cryptography

Proven Cryptography

Product security shall use strong, proven, updatable cryptography using open, peer-reviewed methods and algorithms.

An important aspect of any IoT device is how secure the device is when it communicates with other devices, gateways, or the cloud. This standard mandates using proven cryptographic methods rather than attempting to implement your own.

Developers commonly secure communications such as TCP/IP connections, Bluetooth, Zigbee, or Z-Wave using the standardized and proven cryptographic methods native to the protocol. However, if a microcontroller sends sensitive information over a simple interface such as a UART to another microcontroller, it is important to realize that data should also be secured to prevent someone from snooping the UART line.

Silicon Labs offers a hardware CRYPTO module that provides an efficient acceleration of common cryptographic operations and allows these to be used efficiently with low CPU overhead. The CRYPTO module includes hardware accelerators for the Advanced Encryption Standard (AES), Secure Hash Algorithm SHA-1 and SHA-2 (SHA-224 and SHA-256), and modular multiplication used in ECC (Elliptic Curve Cryptography) and GCM (Galois Counter Mode). The CRYPTO module can autonomously execute and iterate a sequence of instructions to aid software and speed up complex cryptographic functions like ECC, GCM, and CCM (Counter with CBC-MAC).

In addition to the CRYPTO module, Silicon Labs includes mbed TLS as part of the Gecko Platform SDK. mbed TLS is open source software licensed by ARM Limited. It provides an SSL library that makes it easy to use cryptography and SSL/TLS in applications. mbed TLS supports software implementations of all crypto algorithms that are supported by TLS 1.2 as well as a build API that allows hardware drivers to replace the software implementations when cipher accelerators are supported by the platform. Its modular framework allows for subcomponents like the crypto libraries to be incorporated into a design independently of the SSL/TLS components, saving valuable code space and runtime RAM. mbed TLS supports SSLv3 up to TLSv1.2 communication by providing the following:

- TCP/IP communication functions: listen, connect, accept, read/write.
- SSL/TLS communication functions: init, handshake, read/write.
- X.509 functions: CRT, CRL and key handling
- Random number generation
- Hashing
- Encryption/decryption

These functions are split up into logical interfaces. They can be used separately to provide any of the above functions or to mix-and-match into an SSL server/client solution that utilizes a X.509 PKI. Examples of such implementations are provided with the source code. Components or plugins and APIs provide configuration interfaces accessible through the various SDK installations.

For more information, see the latest MCU and Peripheral Software Documentation for the target part at https://docs.silabs.com.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Security by Default

Security by Default

Product security shall be appropriately enabled by default by the manufacturer.

The state in which a product is shipped is up to the manufacturer. This standard mandates that any security features provided with the product be enabled before shipping. Customers should not have to turn security on; rather they should actively have to disable it. For example, Silicon Labs Z-Wave end-nodes and gateway SDKs ship with S2 cryptography and SmartStart network formation enabled by default.

Silicon Labs believes that product security should be considered during product design, and not as an afterthought. Within development environments, all Silicon Labs application security features may be enabled or disabled as appropriate during application development. Security must also be considered during device design and testing. Bringing Up Custom Devices for the EFR32MG and EFR32FG Families describes the security tokens (keys, certificates, and so on) that can be programmed into a custom device to support various types of security, including that provided by the Gecko Bootloader (see Signed Software Updates).

Copyright © 2025 Silicon Laboratories. All rights reserved.

Signed Software Updates

Signed Software Updates

The product shall only support signed software updates. While it is critical that all products be updatable, it is just as critical that these update images be secured. A manufacturer must cryptographically sign update images to prevent tampering during deployment. The product must not use unsigned updates, as they could be fraudulent.

Silicon Labs development tools support building signed upgrade images and securely updating devices in the field, through the Silicon Labs Gecko Bootloader. The Gecko Bootloader can be configured to perform a variety of functions, from device initialization to firmware upgrades. Key features of the bootloader are:

- Useable across Silicon Labs Gecko microcontroller and wireless microcontroller families
- In-field upgradeable
- Configurable
- Enhanced security features, including:
- Secure Boot: When Secure Boot is enabled, the bootloader enforces cryptographic signature verification of the application image on every boot, using asymmetric cryptography. This ensures that the application was created and signed by a trusted party.
- Signed upgrade image file: The Gecko Bootloader supports enforcing cryptographic signature verification of the upgrade image file. This allows the bootloader and application to verify that the application or bootloader upgrade comes from a trusted source before starting the upgrade process, ensuring that the image file was created and signed by a trusted party.
- Encrypted upgrade image file: The image file can also be encrypted to prevent eavesdroppers from acquiring the plaintext firmware image.

On Series 1 devices, the Gecko Bootloader has a two-stage design, first stage and main stage, where a minimal first stage bootloader is used to upgrade the main bootloader. The first stage bootloader only contains functionality to read from and write to fixed addresses in internal flash. To perform a main bootloader upgrade, the running main bootloader verifies the integrity and authenticity of the bootloader upgrade image file. The running main bootloader then writes the upgrade image to a fixed location in internal flash and issues a reboot into the first stage bootloader. The first stage bootloader verifies the integrity of the main bootloader firmware upgrade image, by computing a CRC32 checksum before copying the upgrade image to the main bootloader location.

On Series 2 devices, the Gecko Bootloader consists only of the main stage bootloader. The main bootloader is upgradable through the hardware peripheral Secure Engine. The Secure Engine provides functionality to install an image to address 0x0 in internal flash, by copying from a configurable location in internal flash. To perform a main bootloader upgrade, the running main bootloader verifies the integrity and authenticity of the bootloader upgrade image file. The running main bootloader then writes the upgrade image to the upgrade location in flash and requests that the Secure Engine install it. The Secure Engine is also capable of verifying the authenticity of the main bootloader update image against a root of trust. The Secure Engine itself is upgradable using the same mechanism.

In summary, Series 2 devices support a hardware root of trust and a Secure Boot process that verifies the authenticity and integrity of Gecko Bootloader, whereas in Series 1 devices, the authenticity and integrity of Gecko Bootloader are assumed trusted and are not explicitly checked.

The Gecko Bootloader can enforce application image security on two levels:

- Secure Boot refers to the verification of the authenticity of the application image in main flash on every boot of the device. When Secure Boot is enabled, the cryptographic signature of the application image in flash is verified on every boot, before the application is allowed to run. Secure Boot is not enabled by default in the example configurations provided by Silicon Labs, but enabling it is highly recommended to ensure the validity and integrity of firmware images.
- Secure Firmware Upgrade refers to the verification of the authenticity of an upgrade image before performing a bootload, and optionally enforcing that upgrade images are encrypted. The Secure Firmware Upgrade process uses symmetric

Copyright © 2025 Silicon Laboratories. All rights reserved.

encryption to encrypt the upgrade image, and asymmetric cryptography to sign the upgrade image in order to ensure its integrity and authenticity.

For more information on Silicon Labs' support for software update security, refer to the following:

Bootloaders in general: Bootloader Fundamentals

The Gecko Bootloader in general: Silicon Labs Gecko Bootloader User's Guide for Series 3 and Higher, Silicon Labs Gecko Bootloader User's Guide for GSDK 4.0 and Higher (series 1 and 2 devices), or *UG266: Silicon Labs Gecko Bootloader User's Guide for GSDK 3.2 and Lower*.

Using the Gecko Bootloader with specific protocols:

- Using the Gecko Bootloader with EmberZNet
- Using the Gecko Bootloader with Silicon Labs Connect
- Using the Gecko Bootloader with Silicon Labs Bluetooth Applications

Secure Boot on Series 2 devices: Series 2 Secure Boot with RTSL

Copyright © 2025 Silicon Laboratories. All rights reserved.

Automatically Applied Updates

Automatically Applied Updates

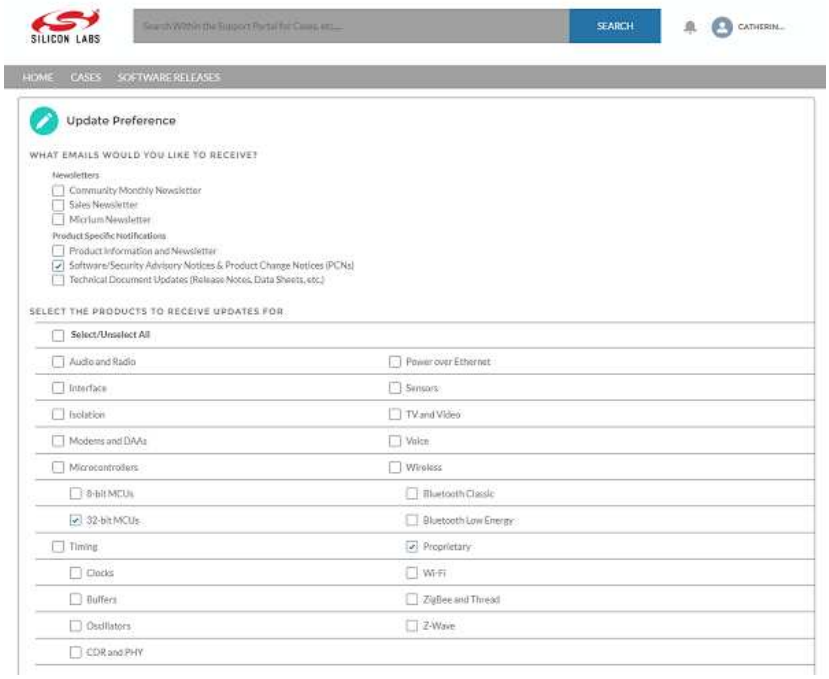
The manufacturer will act quickly to apply timely security updates. Whenever a security vulnerability is detected, the manufacturer will automatically apply a patch to the product. No user intervention will be required.

It is the manufacturer's responsibility to develop and implement automatic security updates. The design and methodology of such systems, for example through a Cloud-connected infrastructure or by direct intervention by a service representative, is up to you.

Silicon Labs will notify you of any security-related updates, as described in Vulnerability Reporting Program. Your responsibility is to evaluate the level of risk that vulnerability poses for your particular product and to integrate the update into your platform as appropriate so that your end users are protected. Updated components might include the protocol libraries, Secure Engine firmware inside the Series 2 family, or an SDK module such as the Gecko Bootloader that enforces secure OTA updates and secure boot functionality.

Silicon Labs recommends the following:

- Subscribe to security updates through our Salesforce portal. To review or change your subscriptions, log in to the portal, click **HOME** to go to the portal home page and then click the **Manage Notifications** tile. Make sure that **Software/Security Advisory Notices & Product Change Notices (PCNs)** is checked, and that you are subscribed at minimum for your platform and protocol. Click **Save** to save any changes.



- Do not turn off Simplicity Studio's update notification. Within Simplicity Studio, you can download updates and easily access product release notes.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Vulnerability Reporting Program

Vulnerability Reporting Program

The manufacturer shall implement a vulnerability reporting program, which will be addressed in a timely manner. All companies that offer Internet-connected devices and services shall provide a public point of contact as part of a vulnerability disclosure policy in order that security researchers and others are able to report issues. Disclosed vulnerabilities should be acted on in a timely manner.

Manufacturers are responsible for implementing their own program. For any individual vulnerability, you will need to weigh the value of transparency with your customers against the risk of malicious use of the information to exploit a vulnerability before it can be addressed. Silicon Labs makes similar decisions about how broadly to report security vulnerabilities discovered in our products.

Silicon Labs customers and security researchers can report security vulnerabilities in Silicon Labs hardware and software products on the Silicon Labs website: https://www.silabs.com/security/product-security.

Silicon Labs' Security Vulnerability Disclosure Policy may be found here:

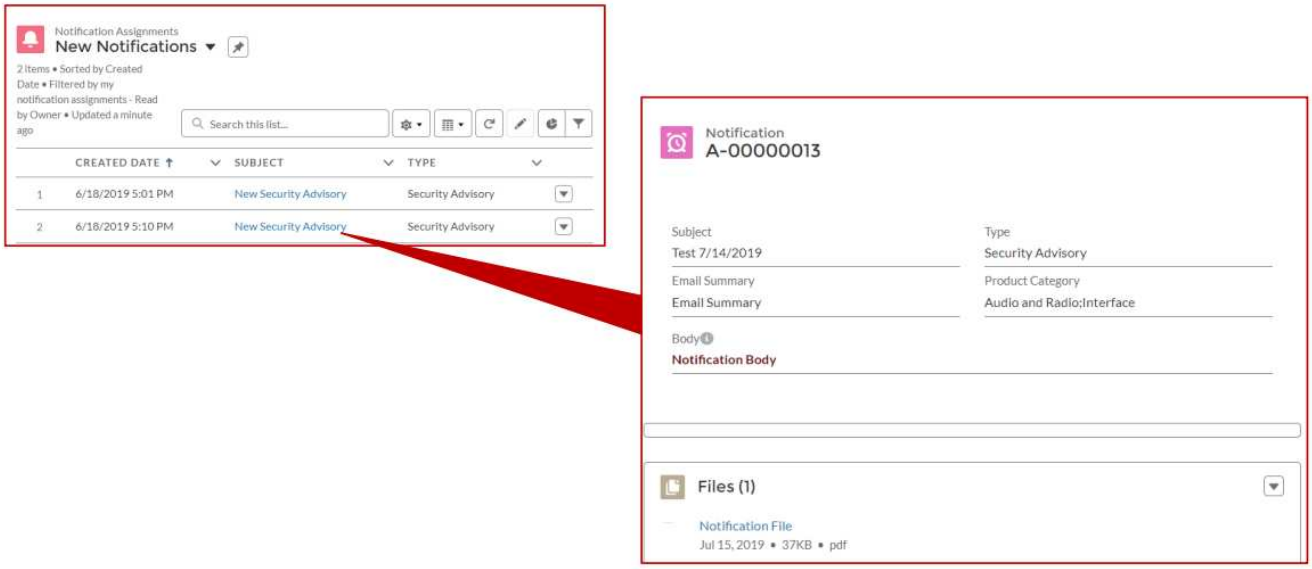
https://www.silabs.com/documents/public/miscellaneous/PS1012-Security_Vulnerability_Disclosure_Policy.pdf

Silicon Labs has a Product Security Incident Response Team (PSIRT) that is dedicated to the case management of reported security vulnerabilities. The PSIRT works with other Silicon Labs groups including Applications, Developers, Sales, and Marketing to assess reported vulnerabilities, perform technical analysis and determine an appropriate response. The key processes for addressing vulnerabilities include:

- **Triage**: Determines what is needed to reproduce the vulnerability.
- **Technical Analysis and Disposition**: Confirms the validity of the security vulnerability, its scope, and its impact, and provides a resolution or disposition decision. Silicon Labs scores incidents according to CVSS 3.1 (Common Vulnerability Scoring System): low, medium, high, critical.
- **Output**: Communicates with our customers. The level and method of disclosure beyond the reporting entity depends on the severity and scope of the vulnerability.

Silicon Labs' provides broad vulnerability reporting to customers subscribed through our Salesforce portal (see Automatically Applied Updates for information on how to subscribe). A subscribed customer will see Security Advisory notifications something like the following:

Copyright © 2025 Silicon Laboratories. All rights reserved.



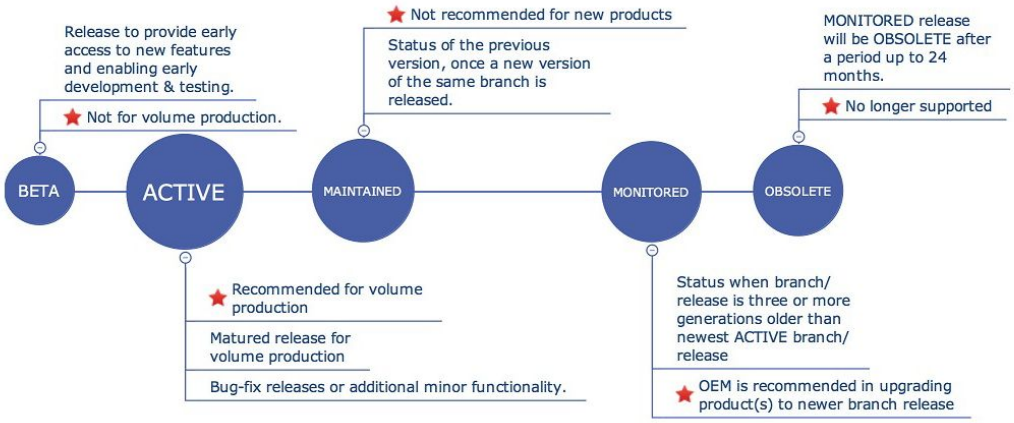
Copyright © 2025 Silicon Laboratories. All rights reserved.

Security Expiration Date

Security Expiration Date

The manufacturer shall be transparent about the period of time that security updates will be provided. Like a manufacturer's product warranty, there shall be transparency around the support period of security updates.

Manufacturers should provide details about product support at various stages and publish security expiration dates. Z-Wave's Protocol Lifecycle provides an example.



The Lifecycle details in what phases updates will be applied, and to what product branch. For details on the various phases and how the lifecycle is implemented for specific Z-Wave products, see:

https://www.silabs.com/products/development-tools/software/z-wave/embedded-sdk/life-cycle

Copyright © 2025 Silicon Laboratories. All rights reserved.

Next Steps

Next Steps

The Silicon Labs Security web page (https://www.silabs.com/security) contains links to a variety of general security-related resources. You may wish to bookmark the page, as it will be continually updated with new content, new tools, and new flows.

If you are already in development, we strongly recommend that you implement the standards described here as you develop, test, and release your product to customers.

If you are in the early stages of your product design and have not already selected a device or development environment, we recommend that you include security considerations in your decision. Silicon Labs provides information about the security features of our devices and development environments. Section EFR32 Series 2 Device Security Features highlights the features and their documentation references. In addition, protocol-specific security information is available in the following documents.

- Zigbee Security
- *AN1302: Bluetooth® Low Energy Application Security Design Considerations in SDK v3.x and Higher*
- *AN1329: Using Silicon Labs Secure Vault Features with OpenThread*
- Bluetooth LE Fundamentals and relevant Knowledge Base Articles (KBAs)
- *UG235.03: Architecture of the Silicon Labs Connect Stack v2.x*
- *UG435.03: Architecture of the Silicon Labs Connect Stack v3.x*

Copyright © 2025 Silicon Laboratories. All rights reserved.

Series 2 Device Security Features

Series 2 Device Security Features

Protecting IoT devices against security threats is central to a quality product. Silicon Labs offers several security options to help developers build secure devices, secure application software, and secure paths of communication to manage those devices. Silicon Labs' security offerings were significantly enhanced by the introduction of the Series 2 products that included a Secure Engine. The Secure Engine is a tamper-resistant component used to securely store sensitive data and keys, and to execute cryptographic functions and secure services.

On Series 1 devices, the security features are implemented by the TRNG (if available) and CRYPTO peripherals.

On Series 2 devices, the security features are implemented by the Secure Engine and CRYPTOACC (if available). The Secure Engine may be hardware-based, or virtual (software-based). Throughout this document, the following abbreviations are used:

- HSE: Hardware Secure Engine
- VSE: Virtual Secure Engine
- SE: Secure Engine (either HSE or VSE)

Additional security features are provided by Secure Vault. Three levels of Secure Vault feature support are available, depending on the part and SE implementation, as reflected in the following table:

Security Level (1)	SE Support	MCU	Wireless SoC (2)
Secure Vault Base (SVB)	N/A	EFM32JG1, EFM32PG1, EFM32JG12, EFM32PG12, EFM32GG11, EFM32GG12, EFM32TG11	EFR32xG1, EFR32xG12, EFR32xG13, EFR32xG14
Secure Vault Mid (SVM)	VSE (VSE-SVM)	EFM32PG22C	EFR32xG22C, EFR32xG27C (3)
Secure Vault Mid (SVM)	HSE (HSE-SVM)	-	EFR32xG21A, EFR32xG21A (Rev C), EFR32MR21A (Rev C), EFR32xG23A, EFR32xG24A, EFR32xG25A, EFR32xG28A
Secure Vault High (SVH)	HSE only (HSE-SVH)	EFM32PG23B, EFM32PG28B	EFR32xG21B, EFR32xG12B (Rev C) EFR32xG23B, EFR32xG24B, EFR32xG25B, EFR32xG28B

Notes:

1. The features of different Secure Vault levels can be found in https://www.silabs.com/security.
2. The x is a letter B, F, M, or Z.
3. Unlike other VSE-SVM parts, the EFR32xG27C device has a built-in PUF.

Secure Vault Mid consists of two core security functions:

- **Secure Boot**: Process where the initial boot phase is executed from an immutable memory (such as ROM) and where code is authenticated before being authorized for execution.
- **Secure Debug access control**: The ability to lock access to the debug ports for operational security, and to securely unlock them when access is required by an authorized entity.

Secure Vault High offers additional security options:

- **Secure Key Storage**: Protects cryptographic keys by "wrapping" or encrypting the keys using a root key known only to the HSE-SVH.
- **Anti-Tamper protection**: A configurable module to protect the device against tamper attacks.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Device authentication: Functionality that uses a secure device identity certificate along with digital signatures to verify the source or target of device communications.

A Secure Engine Manager and other tools allow users to configure and control their devices both in-house during testing and manufacturing, and after the device is in the field.

Silicon Labs strongly recommends installing the latest SE FW image on Series 2 devices and updating to the latest GSDK to mitigate security vulnerabilities. The latest SE FW image can be found in this Windows folder for GSDK v4.x:

C:\Users\<PC USER NAME>\SimplicityStudio\SDKs\gecko_sdk\util\se_release\public

Refer to *AN1222: Production Programming of Series 2 Device*s for guidance on the SE firmware upgrade procedure. The latest SE firmware shipped with Series 2 devices and modules (if available) at the time of this writing are listed in the following table:

:::custom-table{30%,}

Series 2 MCU and Wireless SoC VSE - SVM	Shipped SE Firmware Version
EFM32PG22C	1.2.12
EFR32xG22C	1.2.12
EFR32xG22C (Rev D)	1.2.14
EFR32xG27C	2.2.1
:::	

:::custom-table{30%,}

Series 2 Wireless SoC HSE - SVM	Shipped SE Firmware Version
EFR32xG21A	1.2.13
EFR32MR21A (Rev C)	1.2.16
EFR32xG21A (Rev C)	1.2.16
EFR32xG23A	2.1.7
EFR32xG24A	2.1.7
EFR32xG25A	2.2.1
EFR32xG28A	2.2.2
:::	

:::custom-table{30%,}

Series 2 MCU and Wireless SoC HSE - SVH	Shipped SE Firmware Version
EFR32xG21B	1.2.13
EFR32xG21B (Rev C)	1.2.16
EFM32PG23B	2.1.7
EFR32xG23B	2.1.7
EFR32xG24B	2.1.7
EFR32xG25B	2.2.1
EFM32PG28B	2.2.2
EFR32xG28B	2.2.2
:::	

In support of these products Silicon Labs offers whitepapers, webinars, and documentation. The following table summarizes the key security documents:

:::custom-table{30%,50%,20%}

Copyright © 2025 Silicon Laboratories. All rights reserved.

Document	Summary	Applicability
Document	Summary	Applicability
Series 2 Secure Debug	How to lock and unlock Series 2 debug access, including background information about the Secure Engine	Series 2
Series 2 Secure Boot with RTSL	Describes the secure boot process on Series 2 devices using Secure Engine.	Series 2
Anti-Tamper Protection Configuration and Use	How to program, provision, and configure the anti-tamper module	Series 2 with SVH
Authenticating Silicon Labs Devices using Device Certificates	How to authenticate a device using secure device certificates and signatures, at any time during the life of the product	Series 2 with SVH
Secure Key Storage	How to securely “wrap” keys so they can be stored in non-volatile storage	Series 2 with SVH
AN1222: Production Programming of Series 2 Devices	How to program, provision, and configure security information using Secure Engine during device production	Series 2
AN1303: Programming Series 2 Devices Using the Debug Challenge Interface (DCI) and Serial Wire Debug (SWD)	How to provision and configure Series 2 devices through the DCI and how to program their internal flash memory through the SWD	Series 2
AN1311: Integrating Crypto Functionality Using PSA Crypto Compared to Mbed TLS	How to integrate crypto functionality into applications using Silicon Labs implementation of PSA Crypto compared to Mbed TLS	Series 1 and Series 2
Series 2 TrustZone	Describes the basics of TrustZone, secure and privileged programming model, and shows how to upgrade existing application to TrustZone.	Series 2

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

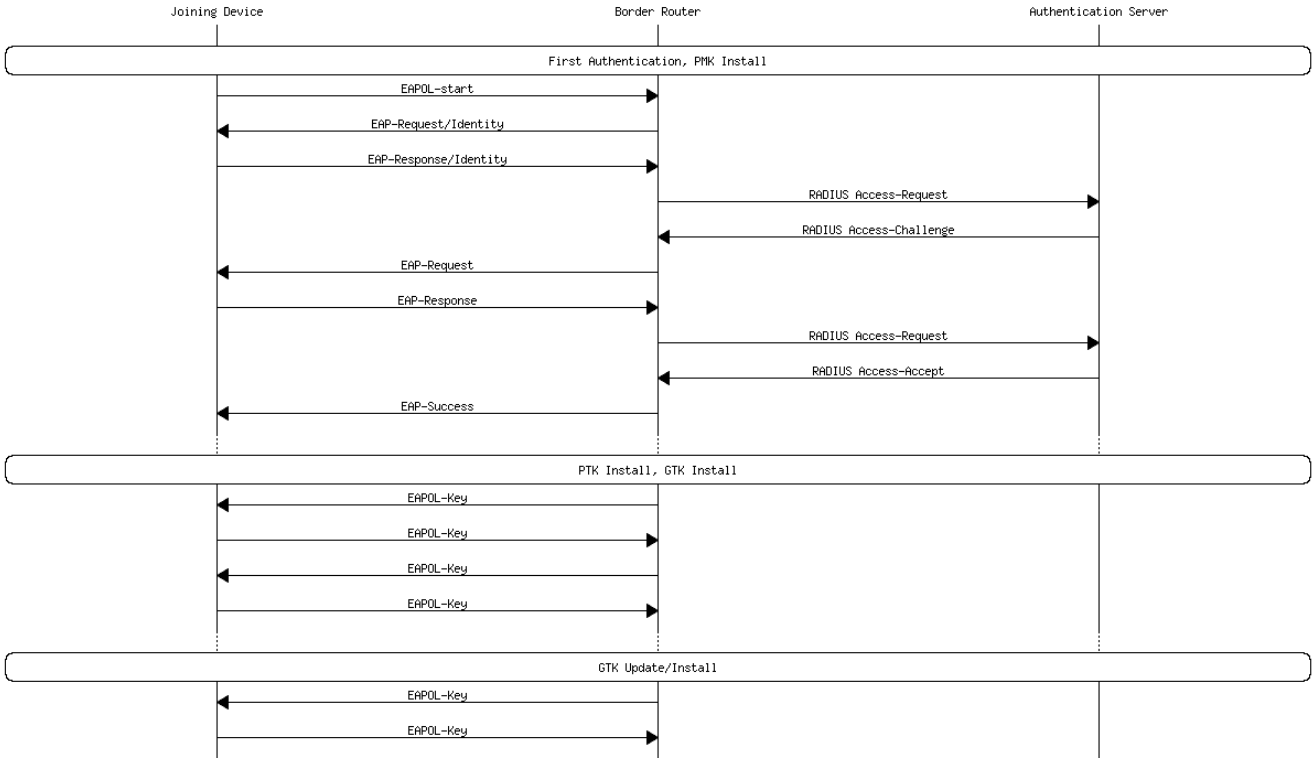
This page discusses security in terms of:

- Authentication
- Wi-SUN Router
- Wi-SUN Border Router
- Create your Own Wi-SUN Certificates

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.



Copyright © 2025 Silicon Laboratories. All rights reserved.

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
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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
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).

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

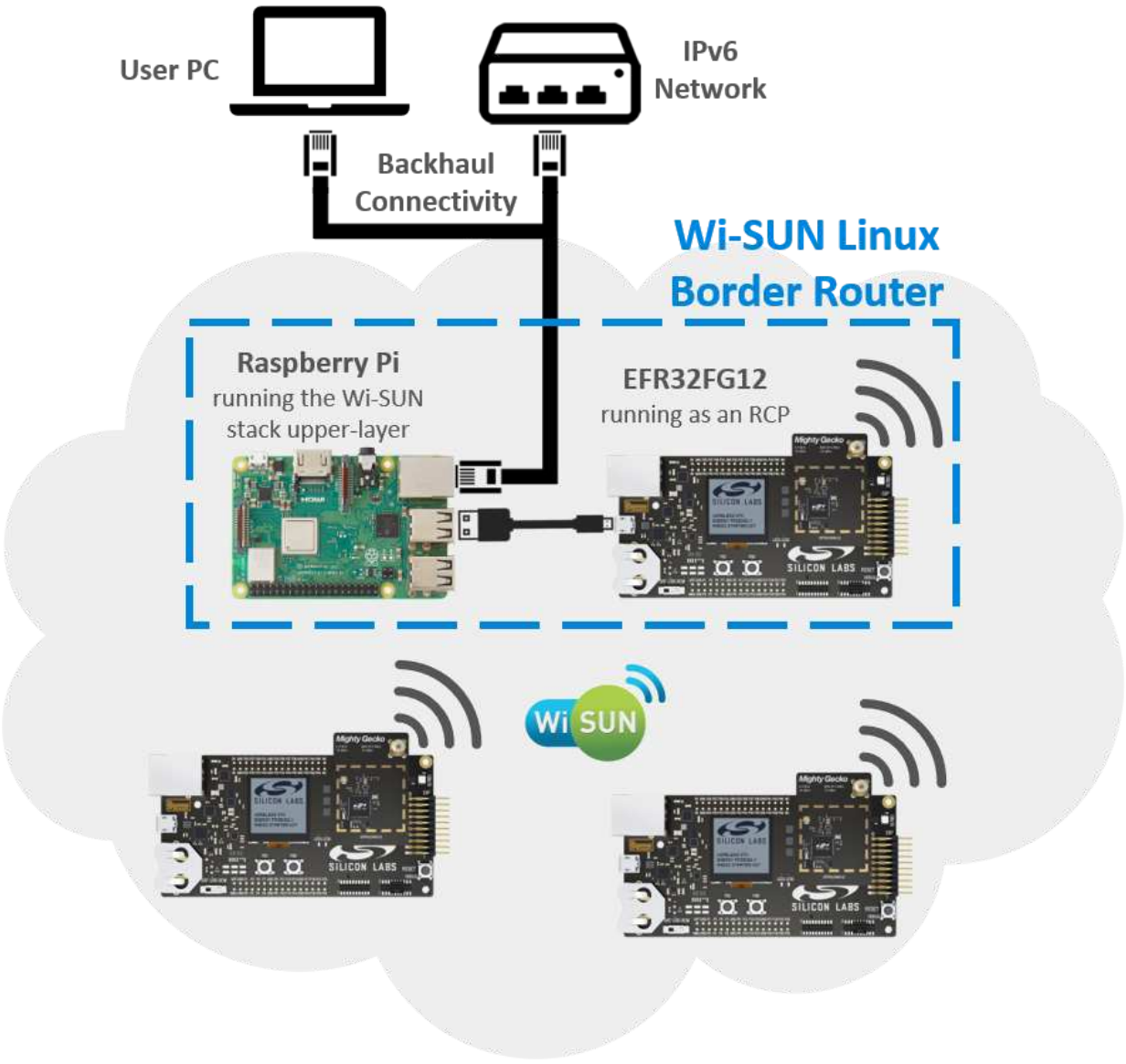
Network Configuration

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.



Copyright © 2025 Silicon Laboratories. All rights reserved.

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 on the Silicon Labs Wi-SUN Products page.

Software Requirements

You need the following software to complete the steps in this application note, you will need the following software:

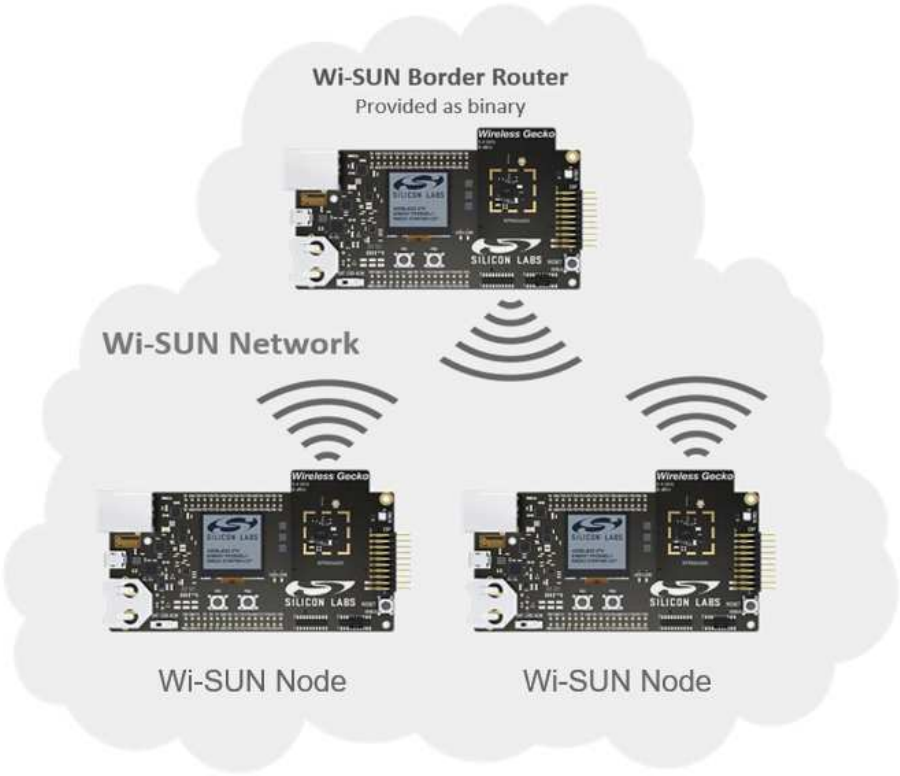
- Simplicity Studio 5
- Raspberry Pi imager
- wsbrd source code repository

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN SoC Border Router

Wi-SUN SoC Border Router

The Silicon Labs Wi-SUN SoC Border Router provides a production-grade Wi-SUN Border Router implementation running entirely on the EFR32. This solution enables an efficient and cost-effective way to evaluate and deploy the Silicon Labs Wi-SUN stack, eliminating the need for expensive and complex external hardware. A Command Line Interface (CLI) is available to facilitate easy configuration and management.



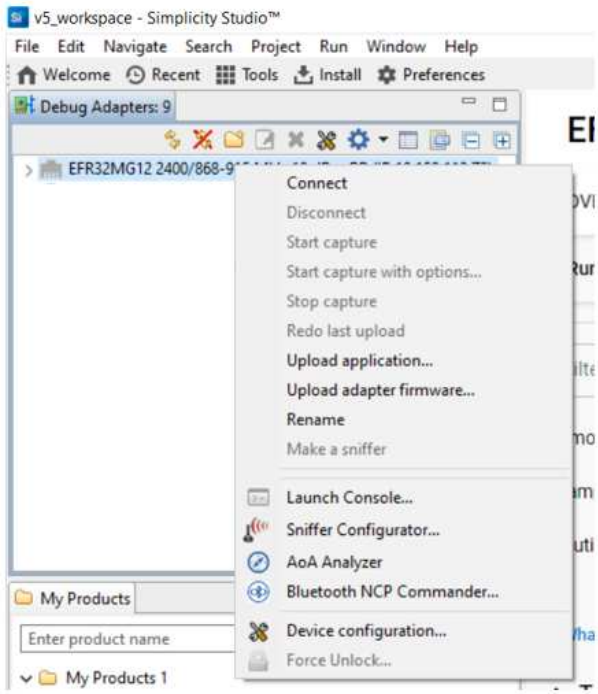
Getting Started with the Solution

The Wi-SUN SoC Border Router project creates a Wi-SUN network that the Wi-SUN nodes can connect to. For an out-of-the-box experience, use the **Wi-SUN – SoC Border Router** Demo instead of creating and building the **Wi-SUN – SoC Border Router** project.

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. The project will be flashed on the fly to your board.
4. In the Debug Adapter view, right click your chosen device, and click on **Launch Console**.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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 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 comes with a list of default PHYs. To add other PHYs to the application, create the **Wi-SUN – SoC Border Router** project and use the Wi-SUN Configurator.

Going Further

For more customization and to adapt the application to your specific use case, it is recommended to use the **Wi-SUN – SoC Border Router** project instead of the demo. This allows you to modify, extend, and tailor the features of the Wi-SUN SoC Border Router to meet your unique requirements effectively.

For an advanced SoC Border router solution with Wi-Fi backhaul connection support, see SoC Border Router Wi-Fi Backhaul.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN Linux Border Router

Wi-SUN Linux Border Router

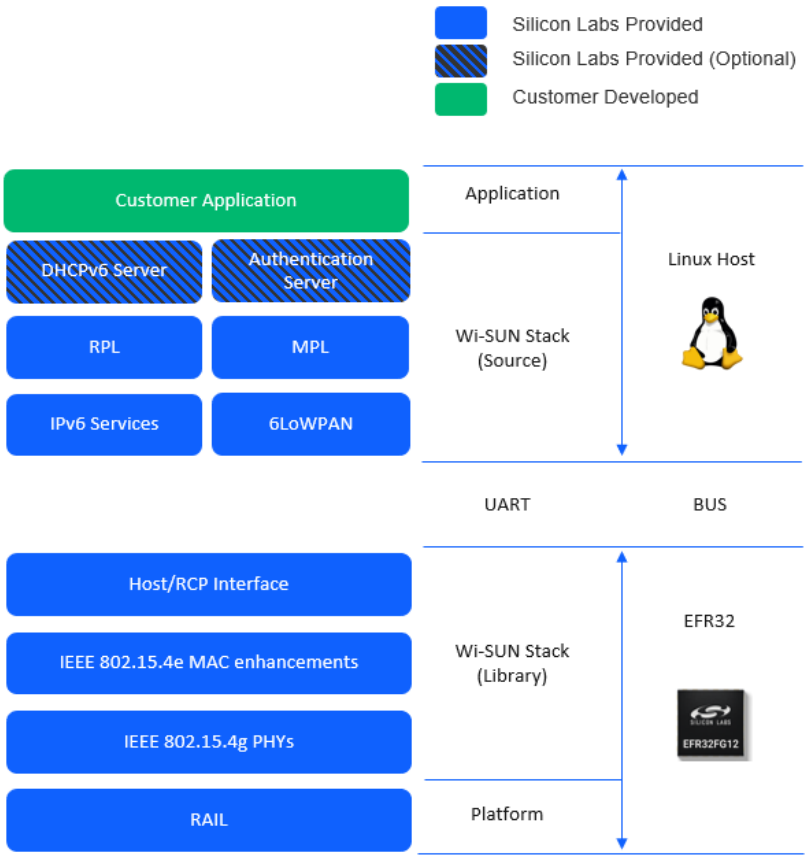
The Silicon Labs Wi-SUN Linux Border Router solution “wsbrd” provides a daemon that is responsible for running the Wi-SUN protocol high-level layers. It is paired with an RF device (RCP) handling the low-level layers and radio activities. The Silicon Labs RCP devices currently supported are listed under 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

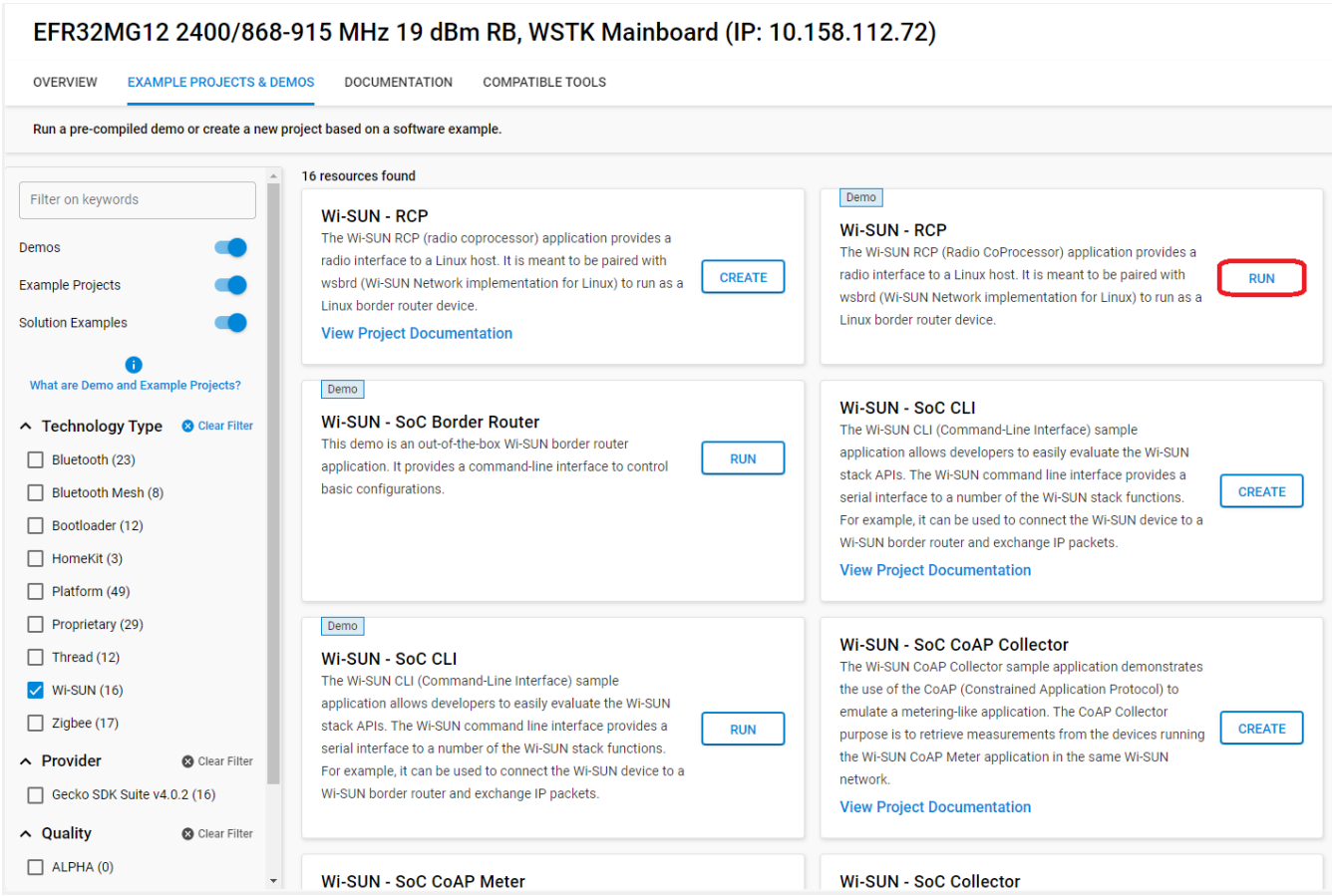
Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
$ 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”.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
$   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)

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

CPCD and wsbrd

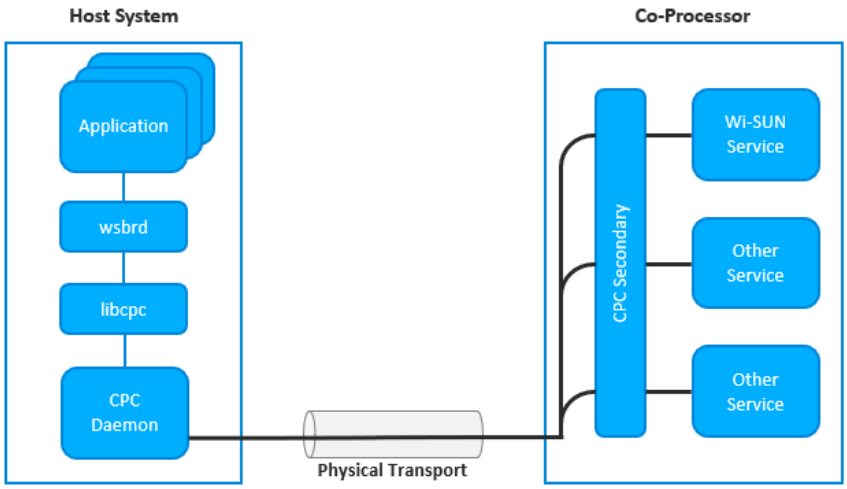
Using Co-Processor Communication Daemon with wsbrd

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. SPI communication using CPC is described in CPC SPI on Linux Border Router



Setting Up the RCP with CPC Secondary Service

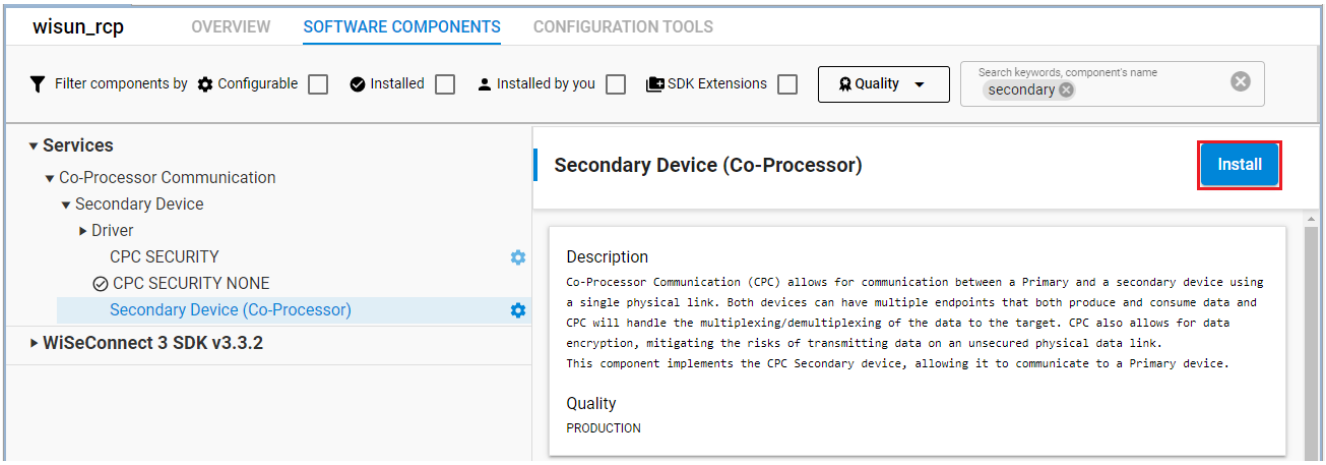
This part is the RCP project creation and configuration in Simplicity Studio.

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 *secondary* to filter the components and install the following components:

Copyright © 2025 Silicon Laboratories. All rights reserved.

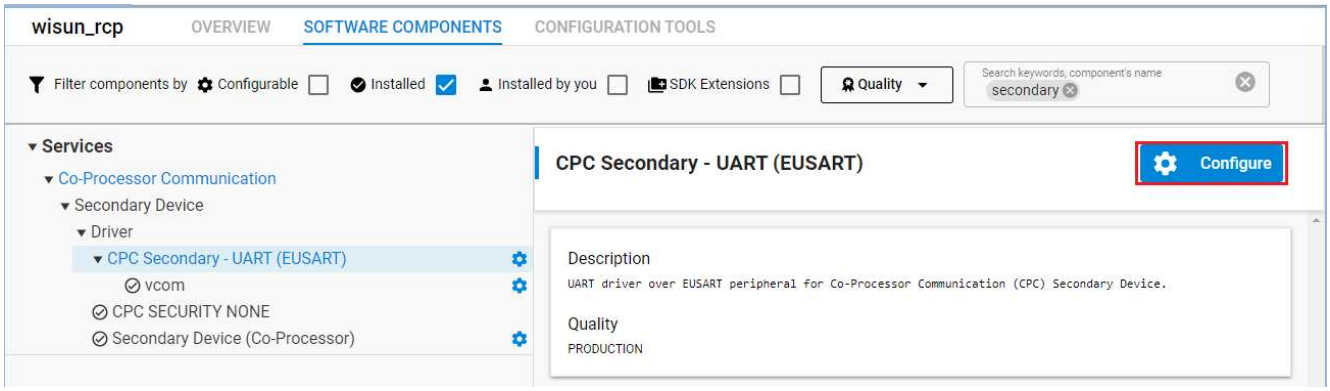


1. Install **Services/Co-Processor Communication/Secondary Device (Co-Processor)**.
2. By default, the **CPC SECURITY NONE** component will be installed automatically after installing the **Secondary Device (Co-Processor)** component. If you wish to enable security on the CPC link, install **CPC SECURITY**.
3. The **Wi-SUN/Application/Wi-SUN RCP/Wi-SUN Border Router RCP - CPC Interface** will be automatically added to your project.

Secondary Device Configuration

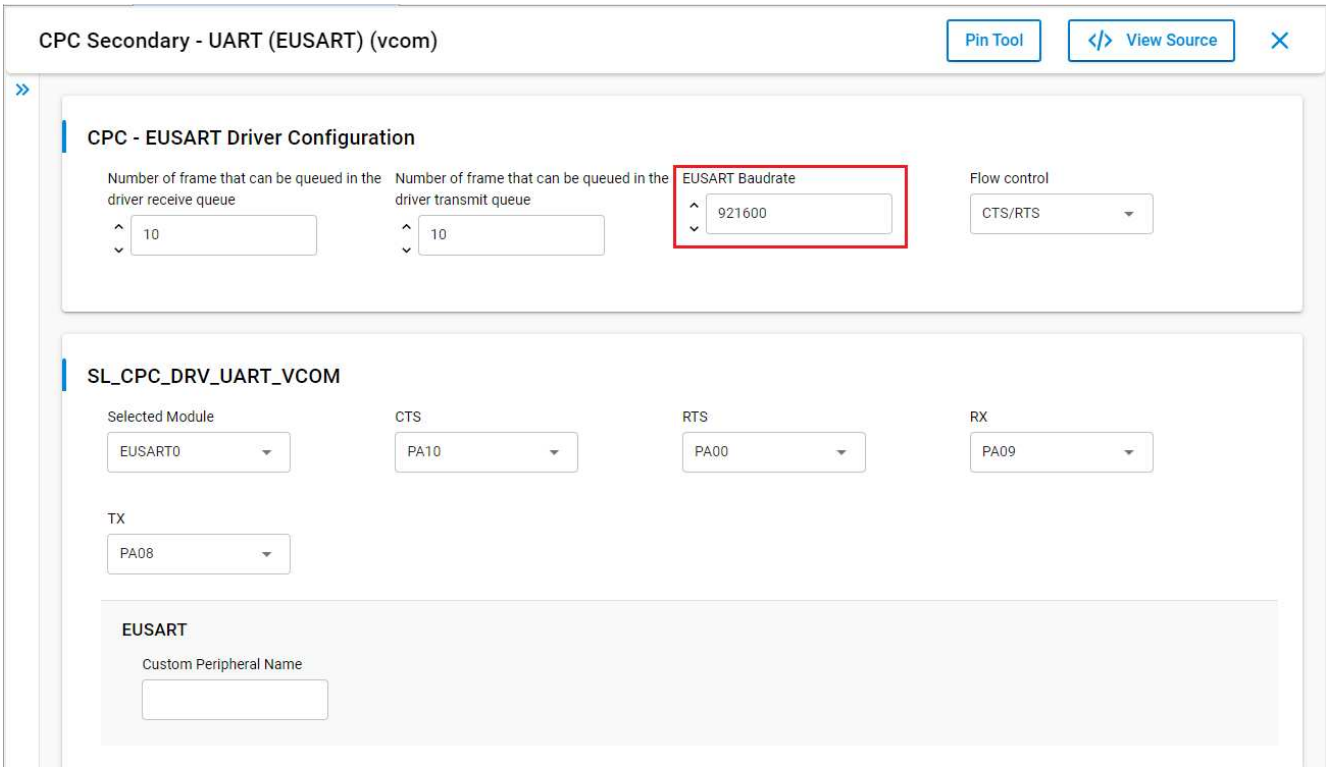
CPC Secondary - UART (EUSART) Component

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 921600. Concerning the **Flow Control**, it is set by default to **CTS/RTS**. Silicon Labs recommends this default; do not set it to **None**.

Copyright © 2025 Silicon Laboratories. All rights reserved.



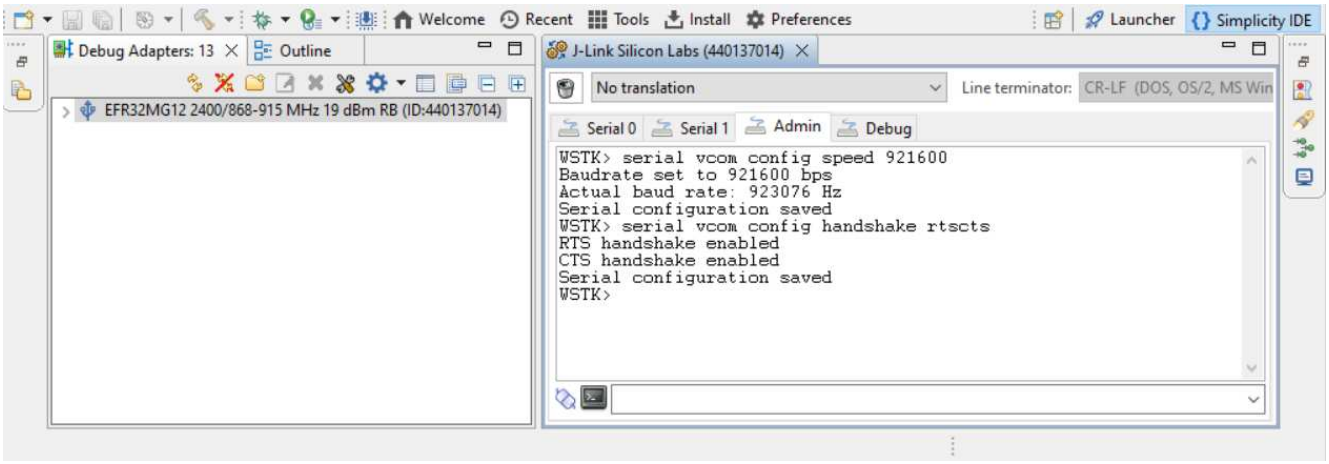
WSTK Configuration

The WSTK boards are factory programmed to support 115200bps with no flow control. To align the WSTK configuration with the chosen CPC EUSART Baudrate 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 speed 921600
serial vcom config handshake rtscts
```

The following figure shows the resulting console output.



After finishing these steps, you can **Build** your project and flash it to the device.

Copyright © 2025 Silicon Laboratories. All rights reserved.

CPCd Installation and Configuration

This part is to be executed on the linux host.

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.

Checking `cpcd` / `libcpc` Installs

```
$ ls -al /usr/local/bin | grep cpc
-rwxr-xr-x  1 root root   748024 May 29 11:01 cpcd
```

```
$ ls -al /usr/local/lib/arm-linux-gnueabihf/ | grep libcpc
lrwxrwxrwx 1 root root    11 May 29 11:01 libcpc.so -> libcpc.so.3
lrwxrwxrwx 1 root root    17 May 29 11:01 libcpc.so.3 -> libcpc.so.4.4.2.0
-rw-r--r-- 1 root root 87776 May 29 11:01 libcpc.so.4.4.2.0
```

```
$ cpcd --version
4.4.2.0
GIT commit: 37bc19f2dfffa8cd954162bb42b8f80755e31e99
GIT branch: refs/heads/main
Sources hash: bbf8593d1941a71988883e18e2bfeb85ec27aaffbb84aadbf12a50a7e0c2a226
```

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: true
```

Note that the `uart_device_file` and `uart_device_baud` values above are examples and can be changed to suit your use case. The `uart_device_file` must match what is visible in `dmesg` after plugging the WSTK/WPK on the host.

If security was enabled on the secondary device (the RCP) by installing the **CPC SECURITY** component, the value of **disable_encryption** must be set to **false**.

Refer to CPC SPI on Linux Border Router for CPC over SPI (SPI provides higher bandwidth than UART).

Copyright © 2025 Silicon Laboratories. All rights reserved.

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 `/wisun-br-linux/CMakeCache.txt` file.

In the **/etc/wsbrd.conf** file (copied from wisun-br-linux/examples/wsbrd.conf), comment-out `cpc_instance` and give it the same value as `instance_name` in **/usr/local/etc/cpcd.conf** and comment the `uart_device` option.

```
cpc_instance = cpcd_0
#uart_device = /dev/ttyACM0
```

Setup Cross-check

```
$ cat /etc/wsbrd.conf | grep cpc_instance
cpc_instance = cpcd_0
$ cat /usr/local/etc/cpcd.conf | grep instance_name
instance_name: cpcd_0
$ cat /usr/local/etc/cpcd.conf | grep uart_device_baud
uart_device_baud: 921600
```

The `uart_device_baud` value must match the WSTK `Stored port speed` when calling `serial vcom` in the 'Admin' console tab.

Launching CPDC and wsbrd

Starting `cpcd` before starting `wsbrd`

If `cpcd` is not running when `wsbrd` is launched, the following error will occur:

```
 $ sudo wsbrd -F /etc/wsbrd.conf
Silicon Labs Wi-SUN border router...
cpc_init: Connection refused
```

Manually starting cpcd

Before starting CPCd and wsbrd, connect the RCP to your Raspberry Pi. Then you can manually start the CPCd using the following command:

```
$sudo cpcd --conf /usr/local/etc/cpcd.conf
..
..
Info : Daemon startup was successful. Waiting for client connections
```

Manually Starting wsbrd

After the line indicating that the CPC deamon startup was successful appears in the logs, use a separate terminal to manually launch wsbrd using the command:

```
sudo wsbrd -F /etc/wsbrd.conf
```

Starting cpcd as a Service

Starting cpcd as a service is a better option once the system has been tested using a manual start. With the `Restart=on-failure`, cpcd will be restarted automatically in case of failure.

This requires:

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Creating a service file for cpcd

```
nano $ sudo nano /usr/local/lib/systemd/system/cpcd.service
```

```
# SPDX-License-Identifier: GPL-2.0 or APACHE-2.0
[Unit]
Description=Silicon Labs CPC Service
Documentation=file:////usr/local/etc/cpcd.conf
After=network.target
After=freeradius.service
# If you rely on an external DHCP server, don't forget to add a dependency here

[Service]
ExecStart=cpcd --conf /usr/local/etc/cpcd.conf
#Restart=on-failure

[Install]
WantedBy=multi-user.target
```

- reloading the system services

```
sudo systemctl daemon-reload
```

Checking `cpcd` Service Status

```
 $ sudo systemctl status cpcd.service
● cpcd.service - Silicon Labs CPC Service
     Loaded: loaded (/usr/local/lib/systemd/system/cpcd.service; disabled; vendor preset: enabled)
     Active: active (running) since Mon 2023-11-06 15:45:08 CET; 2min 48s ago
       Docs: file:////usr/local/etc/cpcd.conf
   Main PID: 7608 (cpcd)
      Tasks: 5 (limit: 1595)
        CPU: 62ms
     CGroup: /system.slice/cpcd.service
             └─7608 cpcd --conf /usr/local/etc/cpcd.conf

Nov 06 15:47:56 rns-md-wisun-br cpcd[7608]: [2023-11-06T14:47:56.882099Z] Core : Endpoint #10: rxd data frame queued
Nov 06 15:47:56 rns-md-wisun-br cpcd[7608]: [2023-11-06T14:47:56.882121Z] Core : Endpoint #10 sent ACK: 6
Nov 06 15:47:56 rns-md-wisun-br cpcd[7608]: [2023-11-06T14:47:56.882147Z] Core : Pushed frame to driver : 14:0A:00:00:86:B9:14
. . .
```

Starting wsbrd as a Service

Starting wsbrd as a service is a good option once the system has been tested using a manual start. With the `Restart=on-failure`, wsbrd will be restarted automatically in case of failure.

This requires:

- Creating a service file for wsbrd

```
nano $ sudo nano /usr/local/lib/systemd/system/wisun-borderrouter.service
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
# SPDX-License-Identifier: GPL-2.0 or Apache-2.0
[Unit]
Description=Wi-SUN Border Router Service
Documentation=file:///usr/local/share/doc/wsbrd/examples/wsbrd.conf
After=network.target
After=freeradius.service
# If you rely on an external DHCP server, don't forget to add a dependency here

[Service]
BusName=com.silabs.Wisun.BorderRouter
ExecStart=/usr/local/bin/wsbrd -F /etc/wsbrd.conf
Restart=on-failure
# Files created by wsbrd contain secrets, so remove read permissions
UMask=0066

[Install]
WantedBy=multi-user.target
```

- reloading the system services

```
sudo systemctl daemon-reload
```

CPC/wsbrd Communication

A log of CPC/wsbrd communication can be obtained adding `cpc` trace to `/etc/wsbrd.conf`. Once CPC is working, removing the `cpc` trace is recommended.

If the `Connected to RCP "x.y.z" (x.y.z), API t.u.v` message is present, everything is working as expected (the RCP version string comes from the RCP binary, not from `wsbrd`, so it validates proper communication between the host and the RCP over CPC).

Error Messages and Corrections

As a general rule, always check the traces from `cpcd` and `wsbrd` in case of issues. These can be obtained directly from the command line is the applications are manually started from a command line, and using `journalctl` if started as services:

```
tail -f sudo journalctl -u cpcd.service -f
tail -f sudo journalctl -u wisun-borderrouter.service.service -f
```

`'support for CPC is disabled'`

- `cpcd` has been installed after `wsbrd` had already been compiled.
- You need to rebuild `wsbrd` following `cpcd` installation.
- Make sure to delete the `/wisun-br-linux/CMakeCache.txt` file and the `/wisun-br-linux/CMakeFiles` folder before rebuilding `wsbrd`.

`'cpc_init: Connection refused'`

- `cpcd` is not running yet when starting `wsbrd`.
- Make sure `cpcd` is started first and the `cpcd` traces contain `Info : Daemon startup was successful. Waiting for client connections`
- You may need to disconnect/reconnect the RCP is some cases.

`'cpc_init: No such file or directory'`

See RCP not communicating

`Failed to connect, secondary seems unresponsive`

Copyright © 2025 Silicon Laboratories. All rights reserved.

See RCP not communicating

RCP Not Communicating

When communication issues appear with the RCP, check cpc status to know more:

```
sudo systemctl status cpcd.service
```

Possible causes are:

- There is no WSTK/WPK plugged in, if the setup has been running before. Unplug/Replug the WSTK/WPK. Check `dmesg` for the `USB ACM device` name.
- There is a mismatch between the `uart_device_file` in `/usr/local/etc/cpcd.conf`. Start `dmesg -w` then unplug/plug the WSTK/WPK to check the `USB ACM device` name and update the `uart_device_file` to match.
- There is an invalid binary running on the RCP. Make sure you flash it with a RCP project with CPC enabled and the correct UART baudrate.
- The serial vcom speed on the WSTK/WPK is incorrect. Refer to WSTK Configuration.

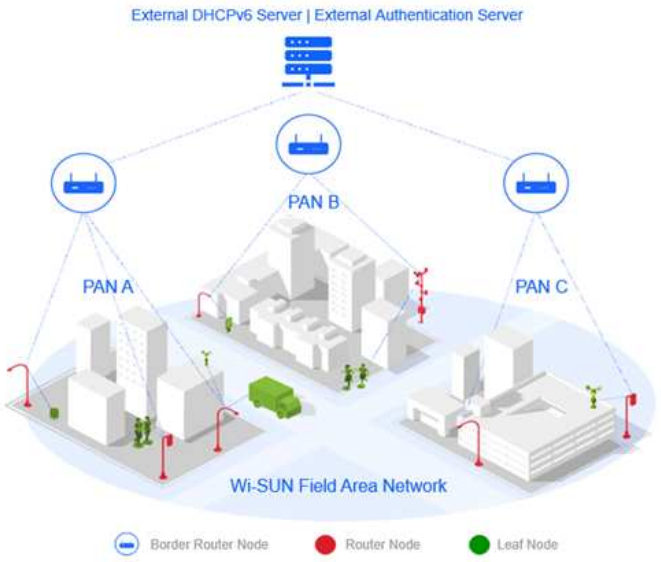
Copyright © 2025 Silicon Laboratories. All rights reserved.

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 the Wi-SUN Linux Border Router repository Readme and follow the guidelines to use the configuration examples.

Copyright © 2025 Silicon Laboratories. All rights reserved.

SoC Border Router Backhaul

SoC Border Router with Wi-Fi Backhaul

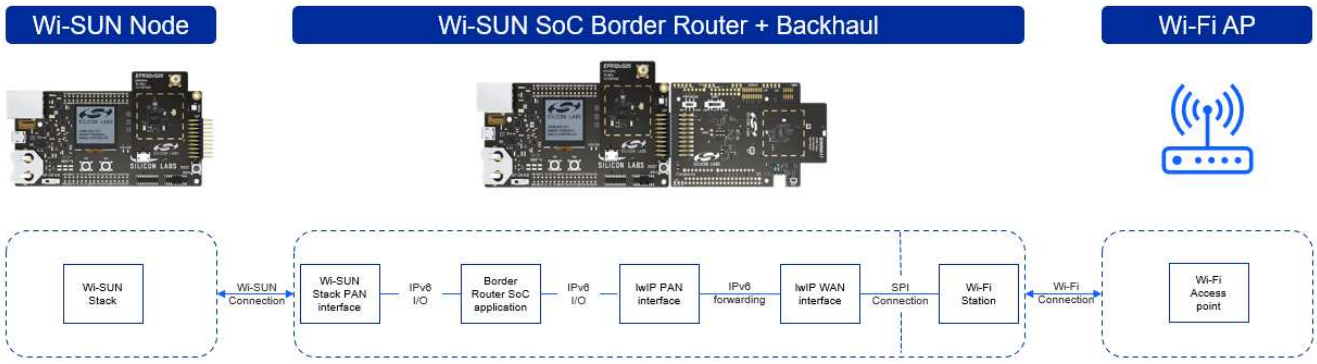
The Wi-SUN SoC Border router supports backhaul connection, using the two APIs `sl_wisun_br_set_ipv6_up_handler()` and `sl_wisun_br_ipv6_down()`. With `sl_wisun_br_set_ipv6_up_handler()` API, a host can register a callback to handle IPv6 packets from the Wi-SUN Network to forward them to internet. And with the `sl_wisun_br_ipv6_down()` API, the host can inject IPv6 packets to the Wi-SUN Network.

For the purpose of demonstrating this feature, a standalone project has been added to the list of Wi-SUN applications. The **Wi-SUN - SoC Border Router - with SiWx917 Wi-Fi backhaul** as its name indicates, relies on the **SiWx917** Wi-Fi board to open a backhaul interface on the Wi-SUN SoC Border Router. The LwIP stack is used as the host IP stack of this project because of its integration, ease-of-use, and most importantly, the IPv6 packet forwarding.

The SoC Border Router with backhaul connectivity enables the use of an external DHCPv6 server. So in addition to the support of the internal DHCPv6 server that makes from the SoC Border router an out-of-the-box solution that can be evaluated on the fly, it also becomes possible to scale the border to a production-grade solution that can fit in larger and complex networks, thanks to the external DHCPv6 server feature support.

The demonstration relies on having a Wi-Fi access point with IPv6 support and DHCPv6 server, configured to forward the Wi-SUN packets through the SiWx917 Wi-Fi interface and have the right IPv6 addresses range that will be assigned to the Wi-SUN nodes.

> **Note**: Silicon Labs considers the configuration of the access point out of scope for this demonstration.



Demo Prerequisites

This demonstration, as shown in the previous figure, relies on four main entities:

- A Wi-SUN Node running on one of the Wi-SUN Development Kits.
- The Border router running on an FG25-RB4270B or an FG25-RB4271A.
- The Wi-Fi module SiWx917-RB4346A plus the Si-EB8045A expansion board.
- An access point with IPv6 and DHCPv6 server support, which is a Raspberry Pi in this demo.

Software used in this demonstration:

- Simplicity Studio v5

Copyright © 2025 Silicon Laboratories. All rights reserved.

- SiSDK v2024.12.0 or later
- WiseConnect 3 Extension latest version

Setup Preparation

Wi-SUN Node

On the Wi-SUN node, flash the **Wi-SUN - CLI Example**. For details on how to create a project and flash it to your board using Simplicity Studio v5, refer to Creating a Project Based on an Example.

SiWx91x Connectivity Firmware

The SiWx91x has to be updated with the latest Connectivity Firmware. Refer to Update SiWx91x Connectivity Firmware for detailed steps.

Wi-SUN Border Router

First start by creating the project **Wi-SUN - SoC Border Router - with SiWx917 Wi-Fi backhaul** in Studio. The project supports, by default, the internal DHCPv6 server. In case of a setup that relies on the internal DHCPv6 server, you can directly build the project and flash it to the board. If the setup has an external DHCPv6 server, follow the next steps to install the **DHCPv6 Relay for Wi-SUN Border Router** component.

1. In the *wisun_soc_brcli_wifi.slcp*, click **[SOFTWARE COMPONENTS]**.
2. Type "DHCPv6" in the search bar, and click **DHCPv6 Relay for Wi-SUN Border Router**.
3. Click **[Install]** and, in the prompt window, choose the option to replace the internal DHCPv6 server with the DHCPv6 Relay for Wi-SUN Border Router.

Once the DHCPv6 Relay component is installed, you can build and flash the project.

Demo Configuration and Launch

After flashing all the binaries to the boards, launch the consoles of the Wi-SUN node board and the SoC Border router. In the **Debug Adapters** view, right-click the desired board and select **Launch Console**. To get a prompt on the Console, go to the Serial 1 tab and press **Enter**.

Wi-Fi Configuration

On the console of the Wi-SUN Border router, use the following commands to configure the access point SSID and passphrase:

```
> wisun set wifi.ssid "BRSoCDemo"
wifi.ssid = BRSoCDemo
> wisun set wifi.passphrase "Silabs_BRSoC"
wifi.passphrase = Silabs_BRSoC
```

After configuring the right ssid and passphrase, connect to the access point using the following command:

```
> wifi connect
[wifi: connecting to BRSoCDemo]
[wifi: connection successful]
```

The IPv6 address of the Wi-Fi interface and other details can be checked by the command `wisun get wifi` :

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
>wisun get wifi
wifi.ssid = "BRSoCDemo"
wifi.security = WPA2_CCMP (3)
wifi.passphrase = "Silabs_BRSoC"
wifi.wlan_state = connected
wifi.channel_number = 7
wifi.mac_address = ec:f6:4c:a0:43:20:02:08
wifi.ipv6_address = fd00:1234::eef6:4cff:fea0:4320
```

SoC Border router Configuration

The Border Router configuration differs depending on whether it is the internal DHCPv6 server that is being used, or if the setup has an external DHCPv6 server.

With the two configurations, it is essential to set the Wi-SUN Network IPv6 prefix that is recommended to be different from the WAN IPv6 prefix to ease the routing configuration on the access point. Use the following command to set the IPv6 prefix:

```
> wisun set wisun.ipv6_prefix "fdc0:114f:d751:1::/64"
wisun.ipv6_prefix = fdc0:114f:d751:1::/64
```

> Now that the IPv6 prefix of the Wi-SUN Network is known, the access point can be configured to route all the packets destined to the wisun.ipv6_prefix via the wifi.ipv6_address. This configuration is essential to have a working backhaul.

DHCPv6 Relay Configuration

In case of using the DHCPv6 relay, the address of the DHCPv6 server should be provided to the SoC Border Router. To set the DHCPv6 server address, use the following command:

```
> wisun set wisun.dhcpv6_server "fd00:1234::1"
wisun.dhcpv6_server = fd00:1234::1
```

> The IPv6 addresses range with the wisun.ipv6_prefix should be configured on the external DHCPv6 server. It is important that the external DHCPv6 Server generates IPv6 addresses for Wi-SUN nodes with the same IPv6 prefix that was configured on the Border Router side. If not, IPv6 routing protocol (RPL) will not work within the Wi-SUN Network.

Start The SoC Border Router

The SoC Border Router can be started with the default PHY configuration. In case of changing the default PHY configuration, make sure to have the same configuration on the node side. To start the SoC Border Router use the following command:

```
> wisun start_fan11
[Using built-in trusted CA #0]
[Using built-in device credentials]
[Border router started]
```

Node Configuration

If the Node Radio Board is on the same frequency band of the Border Router SoC Radio Board, the node can be started with the default PHY configuration. In case the boards have different frequency bands, check that the node PHY

Copyright © 2025 Silicon Laboratories. All rights reserved.

configuration matches the SoC Border Router PHY configuration and make the necessary adjustment in case the configurations are different before starting the node.

```
> wisun join_fan11
[Using built-in trusted CA #0]
[Using built-in device credentials]
[Connecting to "Wi-SUN Network"]
[Join state 1: Select PAN]
[Join state 2: Authenticate]
[Join state 3: Acquire PAN Config]
[Join state 4: Configure Routing - parent selection]
[Join state 4: Configure Routing - DHCP]
[Join state 4: Configure Routing - address registration]
[Join state 4: Configure Routing - DAO registration]
[Join state 5: Operational]
[Connected: 173 s]
[IPv6 address: fdc0:114f:d751:1::6c9b]
```

Local Ping Test

To test that the backhaul functions correctly, try to ping the node from a Smartphone connected to the access point.

On the phone side, PingTools is used to perform the ping. The following screenshot shows a successful ping of the node IP address `fdc0:114f:d751:1::6c9b` .

From the node, use the following command to ping the phone. Make sure to change the IP address in the example.

```
> wisun ping fd00:1234::b473:7aff:fe9b:a804
PING fd00:1234::b473:7aff:fe9b:a804: 40 data bytes
40 bytes from fd00:1234::b473:7aff:fe9b:a804: icmp_seq=1 time=350 ms
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Using CPC SPI on Linux Border Router

Using CPC SPI on Linux Border Router

This document refers to a setup build using:

- BRD4271A as the Silicon Labs Radio Board
- BRD8016A as the Silicon Labs Raspberry Pi Adapter Board
- A Raspberry Pi as the host

> **Note**: Pin mapping will need to be adapted to any custom hardware.

RCP SPI Connections

SPI signals need to be allocated to pins connected to the host via the Radio Board connectors and the BRD8016A Raspberry Pi connector shown below.

xG25 Software Configuration

SPI signal	sl_cpc_drv_secondary_spi_eusart_exp_config.h	RCP values	xG25 GPIO
IRQ	SL_CPC_DRV_SPI_EXP_IRQ_PORT / SL_CPC_DRV_SPI_EXP_IRQ_PIN	SL_GPIO_PORT_A / 5	EFR32_PA5
CS	SL_CPC_DRV_SPI_EXP_CS_PORT / SL_CPC_DRV_SPI_EXP_CS_PIN	SL_GPIO_PORT_B / 4	EFR32_PB4

Copyright © 2025 Silicon Laboratories. All rights reserved.

SPI signal	sl_cpc_drv_secondary_spi_eusart_exp_config.h	RCP values	xG25 GPIO
(MISO)CIPO/RX	SL_CPC_DRV_SPI_EXP_CIPO_PORT / SL_CPC_DRV_SPI_EXP_CIPO_PIN	SL_GPIO_PORT_C / 1	EFR32_PC1
(MOSI)COPI/TX	SL_CPC_DRV_SPI_EXP_COPI_PORT / SL_CPC_DRV_SPI_EXP_COPI_PIN	SL_GPIO_PORT_C / 0	EFR32_PC0
SCLK	SL_CPC_DRV_SPI_EXP_SCLK_PORT / SL_CPC_DRV_SPI_EXP_SCLK_PIN	SL_GPIO_PORT_C / 2	EFR32_PC2

xG25 Pins

SPI signal	xG25 GPIO	xG25 Pin	WSTK signal
IRQ	EFR32_PA5	34	WSTK_P4
CS	EFR32_PB4	24	WSTK_P7
(MISO)CIPO/RX	EFR32_PC1	2	WSTK_P3
(MOSI)COPI/TX	EFR32_PC0	1	WSTK_P1
SCLK	EFR32_PC2	3	WSTK_P5

BRD4271A Pins

SPI signal	BRD4271A signal	Radio Board Pin
IRQ	WSTK_P4	P201_7
CS	WSTK_P7	P201_10
(MISO)CIPO/RX	WSTK_P3	P201_6
(MOSI)COPI/TX	WSTK_P1	P201_4
SCLK	WSTK_P5	P201_8

BRD8016A Pins

SPI signal	Radio Board Pin	BRD8016A signals	RPi GPIO
IRQ	P201_7	RADIO_P4	RPI_GPIO22
CS	P201_10	RADIO_P7	RPI_GPIO8
(MISO)CIPO/RX	P201_6	RADIO_P3	RPI_GPIO9
(MOSI)COPI/TX	P201_4	RADIO_P1	RPI_GPIO10
SCLK	P201_8	RADIO_P5	RPI_GPIO11

Raspberry Header Pins

SPI signal	RPi GPIO	RPi Header pin	RPi SPI signal
IRQ	RPI_GPIO22	15	
CS	RPI_GPIO8	24	RPI_SPI_CS
(MISO)CIPO/RX	RPI_GPIO9	21	RPI_SPI_MISO
(MOSI)COPI/TX	RPI_GPIO10	19	RPI_SPI_MOSI
SCLK	RPI_GPIO11	23	RPI_SPI_SCLK

Raspberry Software Settings

Copyright © 2025 Silicon Laboratories. All rights reserved.

SPI signal	RPi SPI signal	RPi Software settings	RPi Value
IRQ		cpcd.conf spi_rx_irq_gpio	22
CS	RPI_SPI_CS	device tree spi0_cs_pins[0]	0x08
(MISO)CIPO/RX	RPI_SPI_MISO	device tree spi0_pins[0]	0x09
(MOSI)COPI/TX	RPI_SPI_MOSI	device tree spi0_pins[1]	0x0a
SCLK	RPI_SPI_SCLK	device tree spi0_pins[2]	0x0b

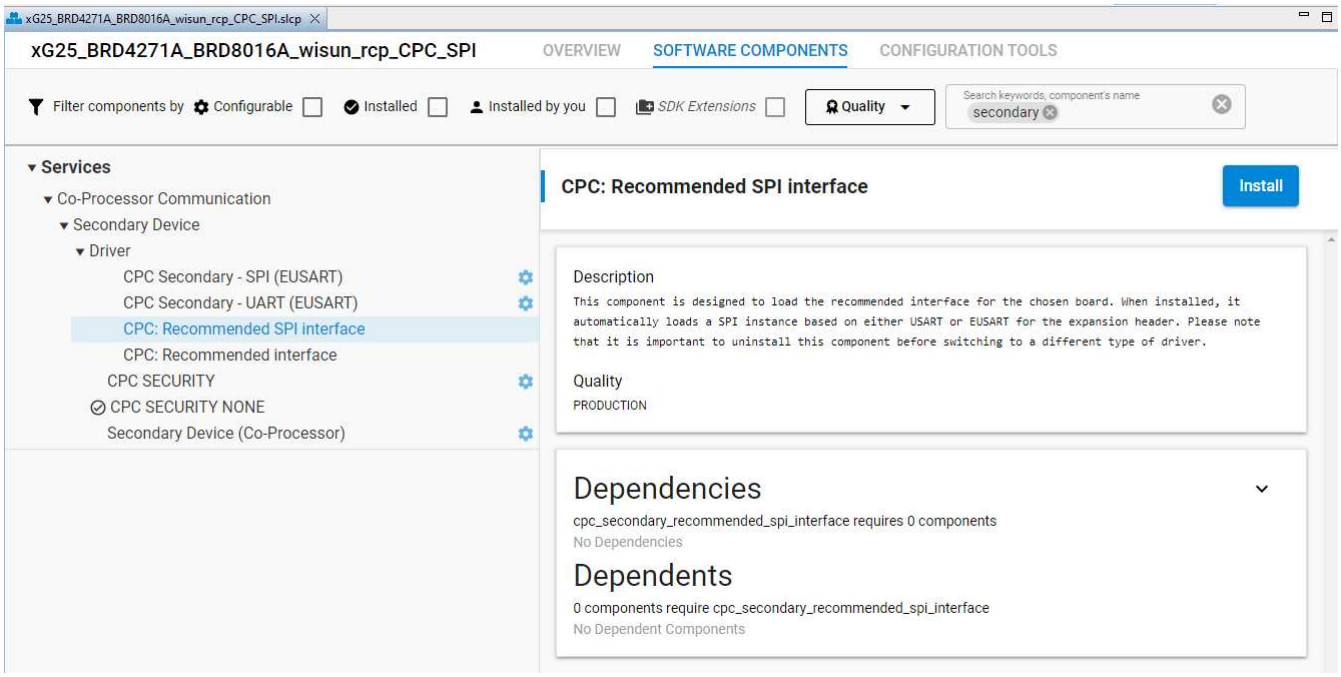
RCP Project

CPC Addition to Wi-SUN RCP Project

Add Wi-SUN / Application / Wi-SUN RCP / Wi-SUN Border Router RCP - CPC Interface.

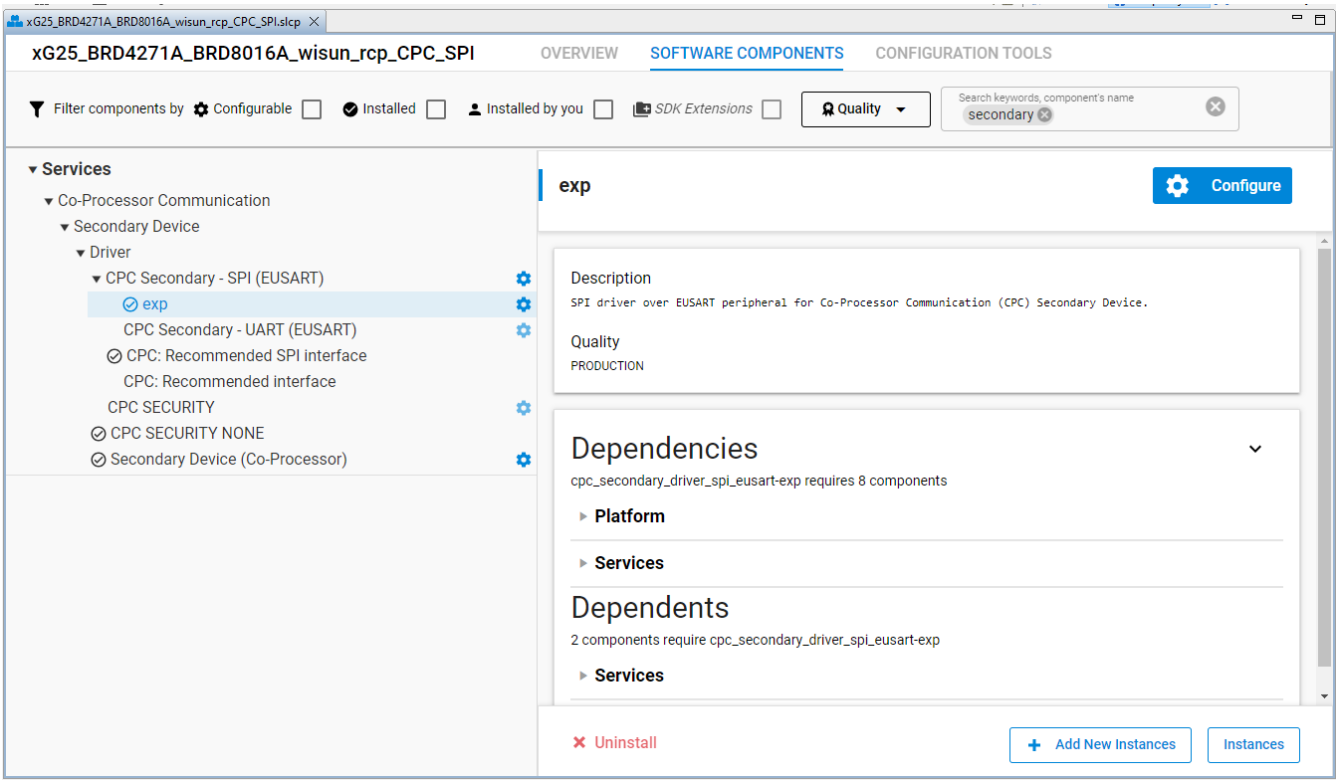
By default, this component uses UART. Follow the steps below to add CPC to your Wi-SUN project.

1. Install the Services / Co-Processor Communication / Secondary Device / Driver / CPC Secondary - SPI: Recommended SPI interface.

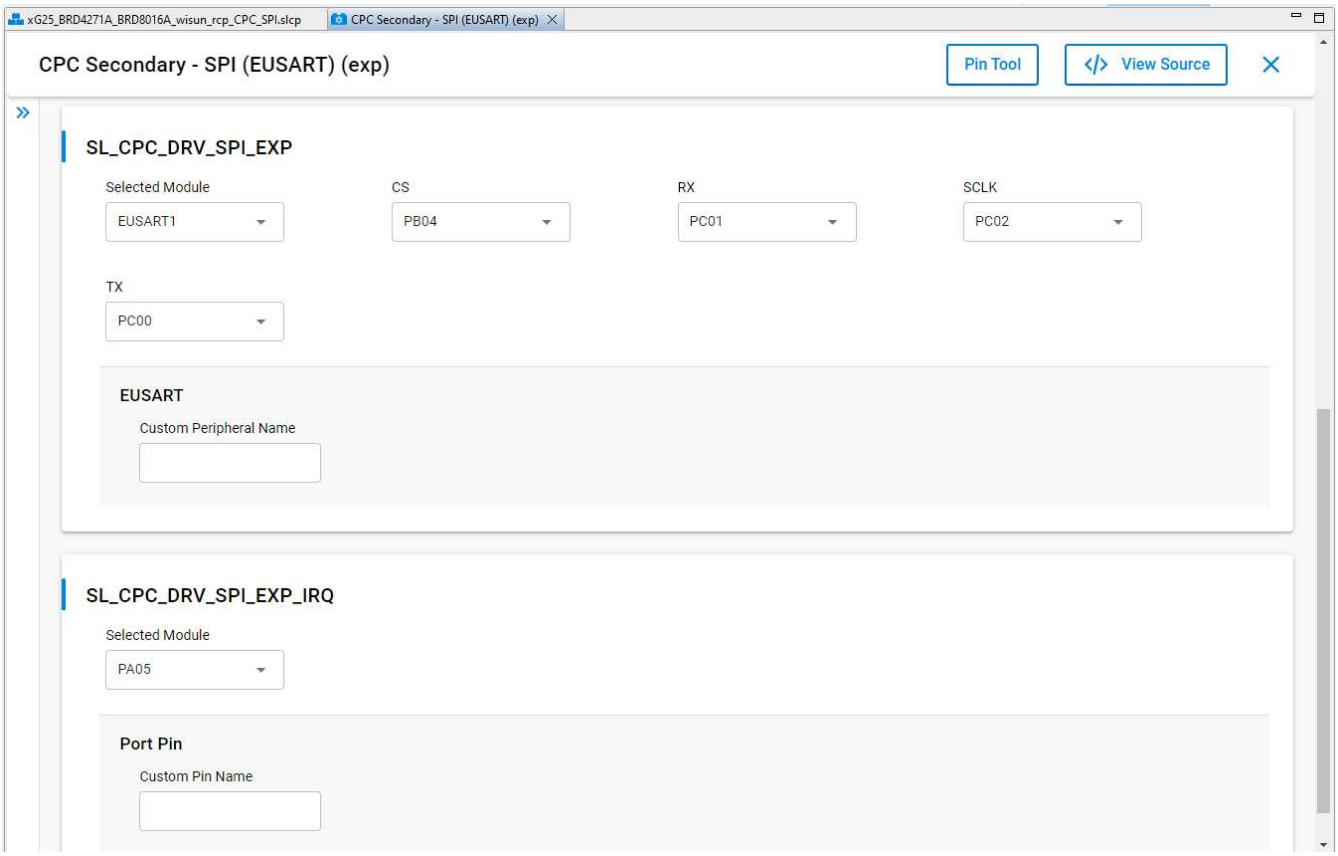


This creates the default `exp` CPC instance under Services / Co-Processor Communication / Secondary Device / Driver / CPC Secondary - SPI (EUSART) / exp.

Copyright © 2025 Silicon Laboratories. All rights reserved.

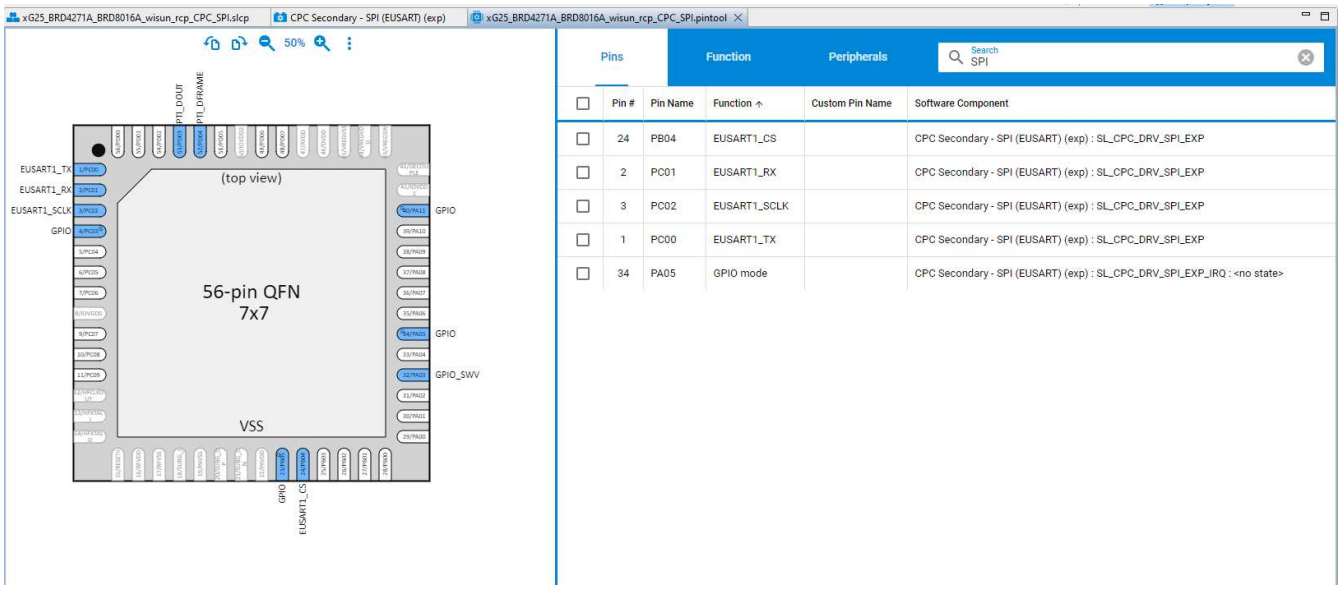


2. Click **Configure** to check the SPI pin configuration, making sure it matches the RCP SPI connections for BRD4271A/BRD8016A/RPi hardware.



Copyright © 2025 Silicon Laboratories. All rights reserved.

Press the **Pin Tool** button and filter on 'SPI' to get a pin view of SPI signals.



4. Build your CPC project.
5. Flash it on the RCP, clearing the flash before flashing (to get rid of any remaining NVM data).
6. Insert the RCP Radio Board on the BRD8016A Adapter Board.
7. Make sure that the expansion board switch is set to the **High Power (LDO)** mode.
8. Insert the BRD8016A on the Raspberry Pi header.

Border Router

> **Note**: As from 23Q4/GSDK V4.3.2, the CPC component in the GSDK will use the **CPC Protocol v5**, so the `cpcd` installation on the Border Router also needs to use this version.

First of all CPC needs to be able to communicate with the RCP, independently from wsbrd.

To achieve this, `cpcd` needs to be compiled with the **CPC Protocol v5**, as follows:

Removing previous `cpcd` / `libcpc` Installs

```
sudo rm -rf /usr/local/lib/arm-linux-gnueabihf/*
```

Installing cpcd with Protocol V6

```
git clone https://github.com/SiliconLabs/cpc_daemon.git cpc_daemon
cd cpc_daemon/
cat CMakeLists.txt | head
mkdir build
cd build/
cmake ../
make
sudo make install
sudo ldconfig
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Checking `cpcd` Protocol Version

```
$ cat ~/cpc_daemon*/CMakeLists.txt | grep CPC_PROTOCOL_VERSION
set(CPC_PROTOCOL_VERSION "6")
```

Checking `cpcd` / `libcpc` Installs

```
$ ls -al /usr/local/lib/arm-linux-gnueabihf/
total 188
drwxr-xr-x 3 root root  4096 Jul  9 11:35 .
drwxr-xr-x 6 root root  4096 Apr 18  2024 ..
lrwxrwxrwx 1 root root    11 Apr 19  2024 libcpc.so -> libcpc.so.3
lrwxrwxrwx 1 root root    17 Jul  9 11:35 libcpc.so.3 -> libcpc.so.4.7.0.0
-rw-r--r-- 1 root root 87776 Jun  7  2024 libcpc.so.4.4.2.0
-rw-r--r-- 1 root root 88160 Jul  9 11:34 libcpc.so.4.7.0.0
drwxr-xr-x 2 root root  4096 Jul  9 11:35 pkgconfig
```

```
 $ cpcd --version
4.7.0.0
GIT commit: 4ff183d9d1ec888812bff3c3562e4961594e5bb5
GIT branch: refs/heads/main
```

Enabling SPI via the Device Tree

In `/boot/config.txt`, comment the UART line and add the SPI overlay as follows:

```
[all]
## Disable WiFi and Bluetooth, both potentially using UART
dtoverlay=disable-bt
dtoverlay=disable-wifi

# Disable UART
#enable_uart=1

# Enable SPI for CPC host/RCP communication
dtoverlay=spi0-1cs

# Generate a reset on GIPO 23, keep the level high after reset
gpio=23=op,dh
```

> Note: A reboot is required after modifying `/boot/config.txt`.

- Reboot the Raspberry Pi

`cpcd` SPI configuration file

(only active lines are shown)

The default configuration file is `/usr/local/etc/cpcd.conf`. To use another, use the `--conf` option.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
$ cat /usr/local/etc/cpcd.conf | grep -v ^$ | grep -v ^#
instance_name: cpcd_0
bus_type: SPI
spi_device_file: /dev/spidev0.0
spi_rx_irq_gpio_chip: gpiochip0
spi_rx_irq_gpio: 22
spi_device_bitrate: 1000000
bootloader_recovery_pins_enabled: false
bootloader_wake_gpio_chip: gpiochip0
bootloader_wake_gpio: 24
bootloader_reset_gpio_chip: gpiochip0
bootloader_reset_gpio: 23
stdout_trace: false
trace_to_file: false
traces_folder: /tmp/
enable_frame_trace: false
rlimit_nofile: 2000
disable_encryption: true
binding_key_file: ~/.cpcd/binding.key
```

`wsbrd` Configuration File for CPC

The changes are similar for UART and SPI mode, since the bus mode is selected in the `cpcd` configuration file (shown here with traces enabled).

```
$ cat ~/wsbrd.conf | grep -v ^$ | grep -v ^#
user = wsbrd
group = wsbrd
cpc_instance = cpcd_0
ipv6_prefix = fd12:3456::/64
network_name = Linux_spi_EU_33_5
domain = EU
chan_plan_id = 33
phy_mode_id = 5
phy_operating_modes = auto
key = /home/pi/wisun-br-linux/examples/br_key.pem
certificate = /home/pi/wisun-br-linux/examples/br_cert.pem
authority = /home/pi/wisun-br-linux/examples/ca_cert.pem
color_output = no
pcap_file = /tmp/dump.pcapng
trace = hif
```

The only line relevant to using CPC is (on top of commenting out UART-related lines):

```
cpc_instance = cpcd_0
```

Setup Cross-Check

Using the BRD8016A Wireless Expansion board: you will need to mount the expansion board on the Raspberry Pi and make sure that the expansion board switch is set to the high power mode.

```
$ cat /etc/wsbrd.conf | grep cpc_instance
cpc_instance = cpcd_0
$ cat /usr/local/etc/cpcd.conf | grep instance_name
instance_name: cpcd_0
```

SPI Signals Allocation Checks

Raspberry Pi Device Tree Check

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
dtc -I fs /sys/firmware/devicetree/base
```

gives (abstracts) `spi0` information (comment lines added for clarity):

```
spi0_pins {
        brcm,function = <0x04>;
        phandle = <0x0e>;
        #           MISO MOSI SCLK
        brcm,pins = <0x09 0x0a 0x0b>;
};
```

```
spi0_cs_pins {
        brcm,function = <0x01>;
        phandle = <0x0f>;
        #           CS0
        brcm,pins = <0x08>;
};
```

Raspberry Pi GPIOs Check

These are controlled by the device tree.

> **Note**: Remember that changes to the device tree require a reboot to be taken into account

```
cat /sys/kernel/debug/gpio | grep SPI

 gpio-7   (SPI_CE1_N          )
 gpio-8   (SPI_CE0_N          |spi0 CS0          ) out hi ACTIVE LOW
 gpio-9   (SPI_MISO           )
 gpio-10  (SPI_MOSI           )
 gpio-11  (SPI_SCLK           )
 gpio-43  (SPIFLASH_CE_N      )
```

Communication Checks

`cpcd` /RCP communication (without wsbrd)

> **Note**: A reboot is required after modifying `/boot/config.txt` , so reboot if you haven't done so yet.

`cpcd` manual start

Use `ps -C cpcd` to make sure no cpcd process is running. Use `sudo kill <cpcd PID>` to kill it if it's running.

A full log (with traces enabled) is provided below.

Click to expand and view the full log

Copyright © 2025 Silicon Laboratories. All rights reserved.

▶ Details

An abstract is:

```
 $ sudo cpcd --conf /usr/local/etc/cpcd.conf
WARNING in function 'main' in file /home/pi/cpc_daemon/main.c at line #68 : Running CPCd as 'root' is not recommended. Proceed at your own risk.
[2025-07-09T13:10:44.177112Z] Info : [CPCd v4.7.0.0] [Library API v3] [RCP Protocol v6]
. . .
[2025-07-09T13:10:44.177487Z] WARNING : In function 'main' in file /home/pi/cpc_daemon/main.c at line #68 : Running CPCd as 'root' is not
recommended. Proceed at your own risk.
[2025-07-09T13:10:44.177578Z] Info : Reading cli arguments
[2025-07-09T13:10:44.177657Z] Info : cpcd --conf /usr/local/etc/cpcd.conf
[2025-07-09T13:10:44.178256Z] Info : Reading configuration
[2025-07-09T13:10:44.178275Z] Info :   file_path = /usr/local/etc/cpcd.conf
[2025-07-09T13:10:44.178291Z] Info :   instance_name = cpcd_0
. . .
[2025-07-09T13:10:44.178430Z] Info :   bus = SPI
[2025-07-09T13:10:44.178445Z] Info :   spi_file = /dev/spidev0.0
[2025-07-09T13:10:44.178458Z] Info :   spi_bitrate = 10000000
[2025-07-09T13:10:44.178473Z] Info :   spi_irq_chip = gpiochip0
[2025-07-09T13:10:44.178487Z] Info :   spi_irq_pin = 22
. . .
[2025-07-09T13:10:44.178597Z] Info :   reset_sequence = true
. . .
[2025-07-09T13:10:44.178638Z] Info : ENCRYPTION IS DISABLED
. . .
[2025-07-09T13:10:44.181352Z] Driver : Opening spi file /dev/spidev0.0
[2025-07-09T13:10:44.181371Z] Driver : Init done
. . .
[2025-07-09T13:10:44.182031Z] Info : Connecting to Secondary...
. . .
[2025-07-09T13:10:44.182436Z] Reset Sequence : Reboot mode sent
. . .
[2025-07-09T13:10:44.183996Z] Reset Sequence : Reboot mode reply received, reset request sent
. . .
[2025-07-09T13:10:44.184735Z] Driver : Secondary signaled a frame to send
. . .
[2025-07-09T13:10:44.185199Z] Core : rxd frame : 14:00:0A:00:C4:55:D3:01:01:04:00:00:00:00:00:13:F9
. . .
[2025-07-09T13:10:44.185284Z] System : on_final_reset()
[2025-07-09T13:10:44.185300Z] Reset Sequence : Reset request response received : 0
. . .
[2025-07-09T13:10:44.185344Z] Reset Sequence : Reset request acknowledged, waiting for device to reset
[2025-07-09T13:10:44.479121Z] Driver : Init of a transaction caused by the IRQ line
[2025-07-09T13:10:44.479174Z] Driver : Secondary signaled a frame to send
. . .
[2025-07-09T13:10:44.481315Z] Reset Sequence : Received RX capability of 2038 bytes
[2025-07-09T13:10:44.481331Z] Info : RX capability is 2038 bytes
. . .
[2025-07-09T13:10:44.481375Z] Reset Sequence : Obtained RX capability
[2025-07-09T13:10:44.481388Z] Info : Connected to Secondary
. . .
[2025-07-09T13:10:44.492150Z] Security : Encryption is disabled
[2025-07-09T13:10:44.492169Z] Core : Security changed state: 0 -> 1
[2025-07-09T13:10:44.492185Z] Info : Daemon startup was successful. Waiting for client connections
```

When you get the `Daemon startup was successful. Waiting for client connections` message, you're ready to start `wsbrd`.

Starting `cpcd` before starting `wsbrd`

If `cpcd` is not running when `wsbrd` is launched, the following error will occur:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
 $ sudo wsbrd --conf /etc/wsbrd.conf
Silicon Labs Wi-SUN border router v2.2
cpc_init: Connection refused
```

Starting `cpcd` as a Service

This is an option. Probably the best, with `Restart=on-failure`.

```
nano $ sudo nano /usr/local/lib/systemd/system/cpcd.service
```

```
$ cat /usr/local/lib/systemd/system/cpcd.service
# SPDX-License-Identifier: GPL-2.0 or APACHE-2.0
[Unit]
Description=Silicon Labs CPC Service
Documentation=file:////usr/local/etc/cpcd.conf
After=network.target
After=freeradius.service
# If you rely on an external DHCP server, don't forget to add a dependency here

[Service]
ExecStart=cpcd --conf /usr/local/etc/cpcd.conf
Restart=on-failure

[Install]
WantedBy=multi-user.target
```

Checking `cpcd` Service Status

```
 $ sudo systemctl status cpcd.service
● cpcd.service - Silicon Labs CPC Service
     Loaded: loaded (/usr/local/lib/systemd/system/cpcd.service; enabled; vendor preset: enabled)
     Active: active (running) since Wed 2025-07-09 15:15:24 CEST; 2s ago
       Docs: file:////usr/local/etc/cpcd.conf
   Main PID: 2110 (cpcd)
      Tasks: 4 (limit: 1595)
        CPU: 19ms
     CGroup: /system.slice/cpcd.service
             └─2110 cpcd --conf /usr/local/etc/cpcd.conf

Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.668762Z] Core : rxd frame :
14:00:11:00:C4:C7:60:06:09:0B:00:04:00:00:00:32:2E:31:30:2E:30:00:18:4D
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.668788Z] Core : Endpoint #0: rxd unnumbered frame
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.668808Z] System : Processing command seq#9 of type 3
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.668872Z] Info : Secondary APP v2.10.0
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.668892Z] Core : Endpoint #0: unnumbered frame processed
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.668906Z] Core : rxd uframe with valid header checksum
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.668921Z] Reset Sequence : Obtained Secondary APP version
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.669204Z] Security : Encryption is disabled
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.669224Z] Core : Security changed state: 0 -> 1
Jul 09 15:15:25 rns-md-wisun-direct-connect cpcd[2110]: [2025-07-09T13:15:24.669239Z] Info : Daemon startup was successful. Waiting for client
connections
```

CPC/wsbrd Communication

A full log is provided below, obtained with `sudo wsbrd --conf /etc/wsbrd.conf -T cpc`.

Click to expand and view the full log

Copyright © 2025 Silicon Laboratories. All rights reserved.

▶ Details

> **Note**: Once CPC is working, removing `-T cpc` is recommended.

If the `Connected to RCP "x.y.z" (x.y.z), API t.u.v` message is present, everything is working as expected (the RCP version string comes from the RCP binary, not from `wsbrd`).

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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 the 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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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`.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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 libcoap2

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

> On more recent linux kernels, use libcoap3 and the commands with the `-notls` suffix to keep it simple (without TLS authentication), considering that your Wi-SUN network already supports encryption.

1. From the node cli, make sure that you are connected to the network.

```
> wisun get wisun.join_state

wisun.connection_state = Operational (5)
```

2. 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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
$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.

Get Help on CoAP Meter Capabilities

```
$coap-client -t text -m put -N -B 1 coap://[fd12:3456::be33:acff:fef6:3161]:5683/meter/help
```

```
Command example:
coap-client -m get -N -B 10 -t text coap://[$METER_IPV6_ADDR]:5683/$URI_PATH -e "$PAYLOAD"

Requests:
  [PUT/POST/GET] sensor/all:
    - no payload or "async": GET request to get measurement data
    - "register": POST/PUT request to register collector
    - "remove": POST/PUT request to remove collector
  [GET]      sensor/temperature: Request to get temperature
  [GET]      sensor/humidity: Request to get humidity
  [GET]      sensor/light: Request to get light
  [PUT/POST]  gpio/led: Toggle LED0/LED1. Payload is index of the LED: "0" or "1"
```

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 "0"
```

LED1 can also be toggled with changing the payload to “1”.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

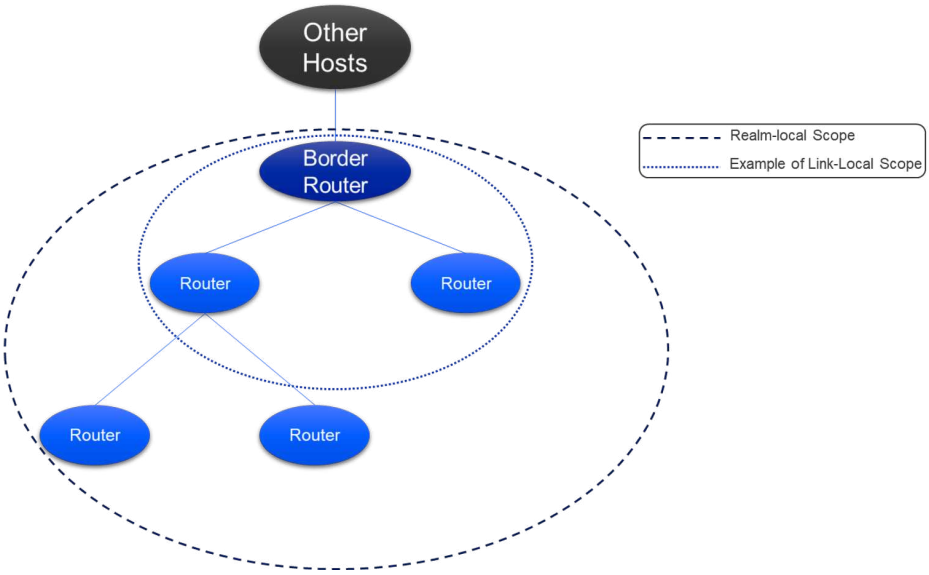
A **scope zone** as described by [RFC 4007] is a topological span within which the address may be used as a unique identifier for an interface or set of interfaces.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Allow multicast on `tun0`

```
sudo ip link set tun0 multicast on
```

Route multicast traffic to `tun0` for common multicast scopes

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
# Link local Nodes
sudo ip -6 route add ff02::1 dev tun0 table local

# Link local Routers
sudo ip -6 route add ff02::2 dev tun0 table local

# Realm Local Nodes
sudo ip -6 route add ff03::1 dev tun0 table local

# Realm Local Routers
sudo ip -6 route add ff03::2 dev tun0 table local
```

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')
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

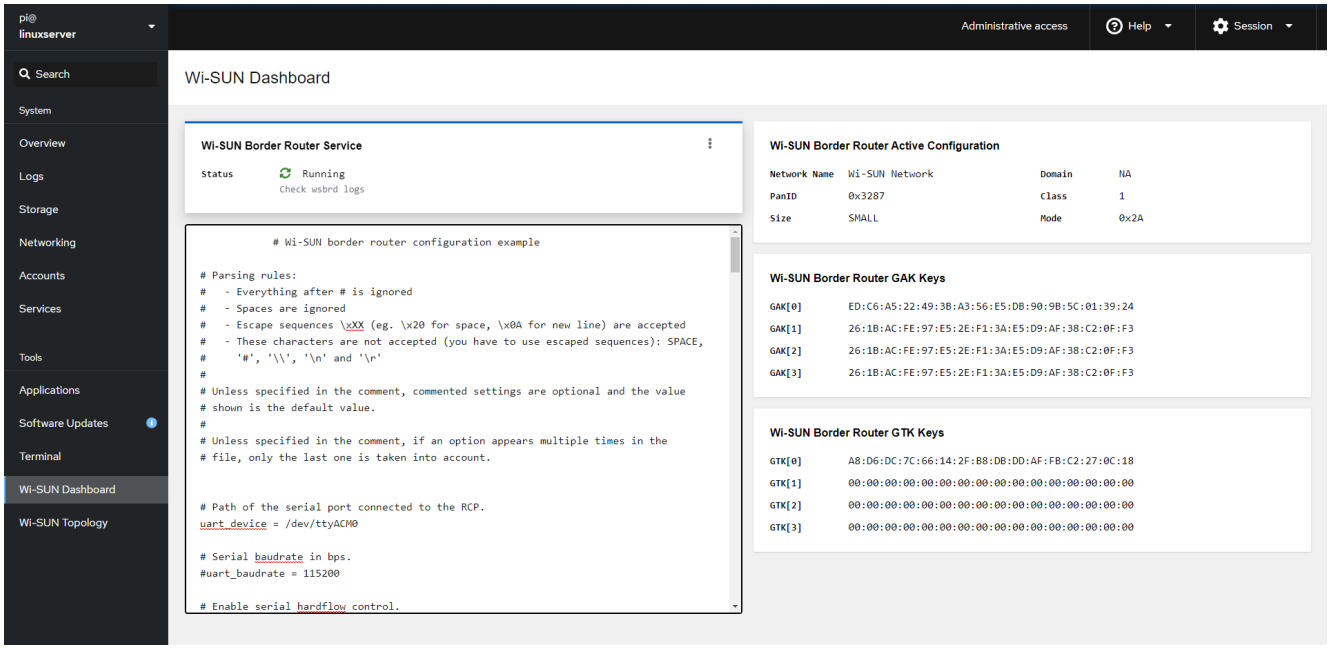
Border Router GUI

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

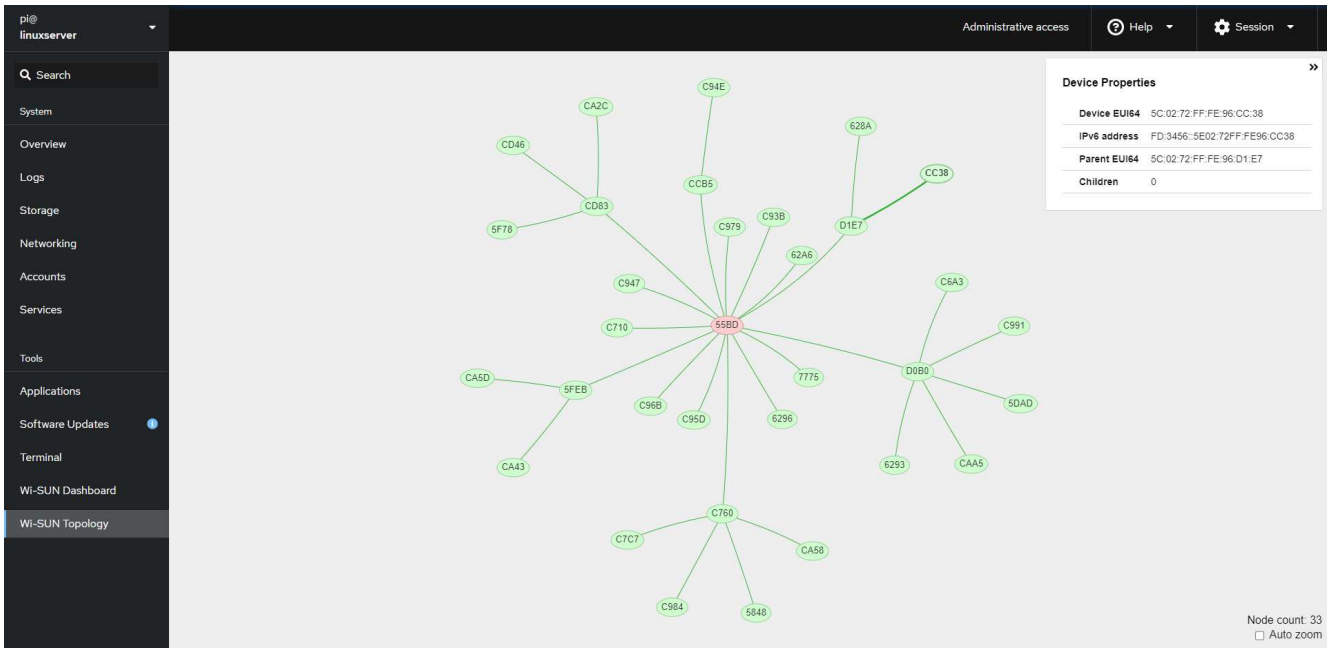
The Wi-SUN border router GUI provides a user interface to configure the border router by modifying the wsbrd.conf file and allows the user to start, restart, and stop the Wi-SUN border router service. In the same interface, the tool shows the running configuration on the border router (Network name, 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN PAN Defect

Wi-SUN PAN Defect

Enhancing Silicon Labs Wi-SUN Networks' Reliability During Power Outages

In the world of wireless communication, ensuring uninterrupted connectivity is paramount. Wi-SUN networks, designed for robust and scalable IoT deployments, face unique challenges during power outages, particularly when the Border Routers lose their main power source. This topic introduces an innovative feature that empowers Silicon labs Wi-SUN Networks to handle Border Router power issues faster, and enables seamless transitions to alternative Personal Area Networks (PANs), ensuring reliability and performance under adverse conditions.

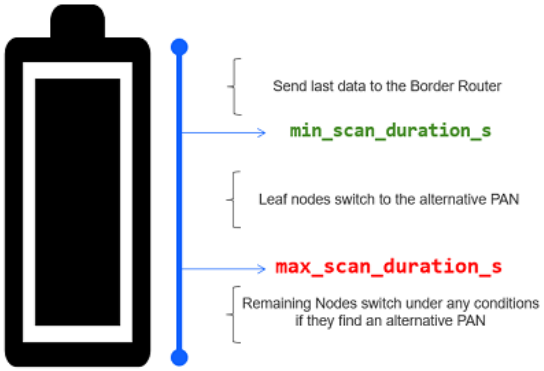
The Role of PAN Defect IE in Power Outage Management

The **PAN Defect Information Element (IE)** is a game-changer for Silicon Labs Wi-SUN networks. It acts as a network-wide alert, signaling routers that the Border Router has lost its primary power source. This feature prompts a proactive response, enabling routers to explore and transition to alternative PANs, thereby maintaining uninterrupted connectivity.

A common scenario involves equipping the border router with a backup battery to handle power outages. The PAN Defect IE plays a crucial role in this setup by providing two essential options that ensure smooth network transitions.

- **min_scan_duration_s**: This option specifies the minimum time the leaf nodes should wait before transitioning to another network.
- **max_scan_duration_s**: Once this duration is reached, all nodes will transition to another PAN if it exists, regardless of conditions.

The following figure illustrates a timeline of the backup battery life, highlighting the two PAN Defect IE options and the potential actions taken at each interval.



Key Considerations

- **Interoperability**: As the PAN Defect IE is a non-standard feature, non-Silicon Labs devices will not recognize it, but will forward it in PCs.
- **Optimized Timing**: Set `max_scan_duration_s` to less than the backup power's expected lifetime to ensure timely transitions without network interruptions.

PAN Defect in Action

In this section, we explore the different phases of nodes transition from the dying PAN to a well functioning alterative PAN. From the power issue detection by the Border Router application to the PAN transition, each step ensures resilience and

Copyright © 2025 Silicon Laboratories. All rights reserved.

continuity of the network.

1. Detection

When the Border Router application detects a power issue, it incorporates the PAN Defect IE into the PAN Configuration packets (PC).

2. Notification

The Border Router disseminates the PAN Defect IE across the network, halting new connections and notifying all routers of the issue.

3. Nodes Response

Nodes receiving the PAN Defect IE take the following steps:

- **Forwarding the Alert**: The IE is shared across the network.
- **Scanning for Alternatives**: Nodes start searching for new parent routers with different PAN IDs.
- **Transitioning**: Nodes start transitioning to the alternative PAN or not, following the roaming procedure. The following section explains this step in detail.

Transitioning Scenarios

To ensure smooth transitioning, the nodes behave differently depending on if they are leaf nodes or if they are routers with children. This transition mechanism differentiation, guarantees an efficient gradual transition ensuring minimal disruption.

Leaf Nodes

When the `min_scan_duration_s` is reached, leaf nodes transition to an alternative PAN immediately upon finding a suitable parent in the new PAN and making sure that a PC has been sent.

Routers With Children

Routers with children keep their current PAN while they continue to scan for an alternative PAN until either:

- `max_scan_duration_s` **is reached**: At this point, they ensure at least one PC has been sent, and transition.
- **No alternative PAN is found**: The scan continues while maintaining the current PAN.

Fallback Mechanism

If no alternative PAN is available, routers will stay connected to the defective Border Router while it is alive and persistently scan for an alternative PAN.

PAN Defect Implementation

Because of its PAN wide nature, the PAN Defect IE configuration and enablement, solely relies on the Border router. To implement the the PAN Defect in your network, follow the steps below.

Monitor the BR Power Source

The Border Router application should monitor the Border Router's power status. When a power issue is detected, it should enable the PAN Defect IE advertisement.

Enable PAN Defect Advertisement

Using Wi-SUN - SoC Border Router

On the **Wi-SUN - SoC Border Router**, use the API `sl_wisun_br_pan_defect_advertise()` to enable or disable the PAN Defect advertisement and configure the `min_scan_duration_s` and the `max_scan_duration_s`:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
sl_wisun_br_pan_defect_advertise(bool defect_advertise, uint32_t min_scan_duration_s, uint32_t max_scan_duration_s);
```

- `defect_advertise` : Enable or disable PAN defect advertisement.
- `min_scan_duration_s` : Minimum time before transitioning if conditions are met.
- `max_scan_duration_s` : Time after which routers must transition to an alternative PAN.

Using the Linux Border Router

On the Linux Border Router, the Pan Defect can be managed either by the `wsbrd_cli` or dBus depending on the user preference.

Via `wsbrd_cli` :

Use the `wsbrd_cli` command to manage the PAN Defect IE:

- To start the IE advertisement and set the `max_scan_duration_s` and `max_scan_duration_s`, use the following command:

```
sudo wsbrd_cli pan-defect start [min_scan_duration_] [max_scan_duration_s]
```

- To stop the IE advertisement, use the following command:

```
sudo wsbrd_cli pan-defect stop
```

Via dBus:

The `wsbrd` dBus APIs allow setting a custom IE in the PAN Configuration packet using `leCustomInsert` .

- Refer to the following command example to **enable** PAN Defect using dBus:

```
sudo gdbus call --system --dest com.silabs.Wisun.BorderRouter --object-path /com/silabs/Wisun/BorderRouter --method
com.silabs.Wisun.BorderRouter.leCustomInsert -- 1 0x49
```

  - **1** to indicate nested short Wi-SUN payload IE (WP-IE).
  - **0x49** is the id of the PAN defect IE.
  - **[0x01,0x3C,0x00,0x00,0x00,0x58,0x02,0x00,0x00]** is the content of the IE described as follow:
    - byte 0: enable1/disable:0 (in the example provided: 0x01)
    - byte 1→ byte 4: min_scan_duration_s (in the example provided: 0x3C,0x00,0x00,0x00 → 60sec)
    - byte 5 → byte 9: max_scan_duration_s (in the example provided: 0x58,0x02,0x00,0x00 → 600sec)
  - **[0x02]** to include the IE in PC Wi-SUN frame types.
- The following command can be used to **stop** including the PAN defect IE in the PC packets:

```
sudo gdbus call --system --dest com.silabs.Wisun.BorderRouter --object-path /com/silabs/Wisun/BorderRouter --method
com.silabs.Wisun.BorderRouter.leCustomInsert -- 1 0x49 [] []
```

Simulation Results

To evaluate the performance and effectiveness of the PAN defect feature, Silicon Labs conducted a series of simulations using various network topologies and sizes. These tests aimed to compare two scenarios:

- With PAN Defect Feature: The border router proactively informs nodes before going offline.
- Without PAN Defect Feature: The border router abruptly goes offline without prior notification.

For each simulation, one key metric was analyzed:

- **Node Downtime**: The time each node remains disconnected during the recovery process.

The results highlight the significant advantages of the PAN defect feature in minimizing disruption.

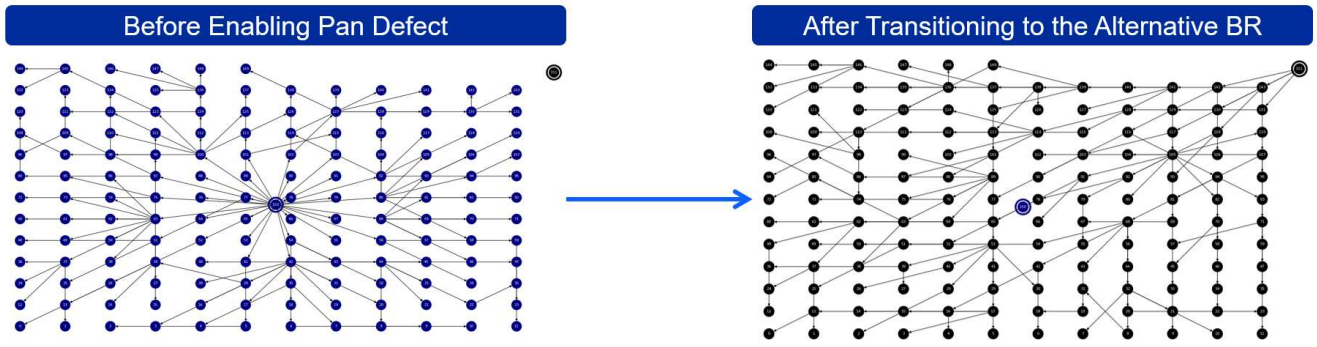
Test Scenario

Copyright © 2025 Silicon Laboratories. All rights reserved.

We will conduct two different tests, one with 150 nodes and the second with 300 nodes.

Topology

We adopted a mesh topology for the evaluation test with a distance of 300 meters between the nodes. The main Border Router is in the middle of the network, and the alternative Border router is placed in the corner in both tests. The following figures are for the 150 nodes test, we used the same topology for 300 nodes test.



Network and PAN defect Configuration

The network configuration is set to a ***medium*** profile. The supposed backup battery life is 30 minutes and the PAN Defect options are set to the following:

- `min_scan_duration_s` : Set to 300 seconds.
- `max_scan_duration_s` : Set to 1200 seconds.

Below is a summary of the simulation results:

Test Results

The table below resumes the test results:

	PAN Defect Enabled	PAN Defect Disabled
Network Size	Average Node Downtime	Average Node Downtime
150 nodes	135 sec	2581 sec
300 nodes	356 sec	3282 sec

Conclusion

By integrating the PAN Defect IE, Wi-SUN networks can maintain high performance and reliability during power outages, ensuring devices and applications remain connected. This proactive approach minimizes downtime, making Wi-SUN networks a trusted choice for resilient IoT deployments.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Using the Wi-SUN Network Measurement Application

Using the Wi-SUN Network Measurement Application

NOTE: This section replaces *AN1364: Using the Wi-SUN Network Measurement Application*. Further updates to this application note will be provided here.

The Wi-SUN Network Measurement Application provides a simple tool for testing Silicon Labs Wi-SUN FAN (Wireless Smart Ubiquitous Network, Field Area Network) stack performance. Using the IPv6 ping capability, the application can trigger round-trip latency tests within a Wi-SUN mesh network. It provides information about the packet error rate, Routing Protocol for Low-Power and Lossy Networks (RPL) routing metrics, average ping latencies with different neighbors, and more. To do so, the application exposes two interfaces: a Command Line Interface (CLI) and the WSTK mainboard LCD screen.

The Wi-SUN FAN stack is supported on the EFR32FG12 and EFR32MG12 devices. The radio boards used must support one of the sub-GHz bands.

Silicon Labs recommends that you review the Quick Start Guide to familiarize yourself with the Silicon Labs Wi-SUN stack and Simplicity Studio 5 before reading this document.

The following features are provided as part of the Wi-SUN Network Measurement application:

- Transmits pings and receives pong replies.
- Computes ping latencies.
- Configures the number of pings to be sent during a test.
- Configures the ping payload size.
- Configures the devices to target with the ping packets.
- Provides the minimum, maximum, and average ping latencies on a given test.
- Provides the ping error rate, the MAC-level packet error rate, and RPL metrics.
- Enables access to the previous features through an LCD screen or a CLI. In addition to these features the Wi-SUN Network Measurement application behaves like a standard Wi-SUN router once connected to a Wi-SUN network. During the connection steps, it automatically selects a suitable parent to connect to the network. When connected, it can be selected by another Wi-SUN device to be its parent.

The Wi-SUN Network Measurement application requires you to have at least two WSTK mainboards and two preferably from the following list of matching radio boards:

- brd4163a
- brd4164a
- brd4170a
- brd4172a
- brd4173a
- brd4253a
- brd4254a

Copyright © 2025 Silicon Laboratories. All rights reserved.

Theory Of Operation

Theory of Operation

This section discusses the operation of the Wi-SUN Network Measurement application independently from the interface. You can execute these either through the CLI or the LCD screen interface. Specific interface information is shared in the following sections.

For the first step you need to start a Wi-SUN network using one of these solutions:

- The Wi-SUN SoC Border Router demo available in the Gecko SDK through Simplicity Studio 5
- The Wi-SUN Linux Border Router solution.

Follow the steps to get your border router up and running and move to the next chapter.

Flash the Application

In the Simplicity Studio 5 Launcher perspective, select the EFR32 evaluation kit you want to use to run the Wi-SUN Network Measurement application. In the **EXAMPLE PROJECTS & DEMOS** panel, find the **Wi-SUN - SoC Network Measurement** project, and click **CREATE** to add it to your workspace.

Finally, build the wisun_soc_network_measurement project and flash it on the selected EFR32. Refer to the Quick Start Guide for the detailed steps.

Connect to a Wi-SUN Network

By default, the Wi-SUN Network Measurement application tries to connect to the **Wi-SUN Network**. If a network with this name and a matching Wi-SUN PHY configuration is available nearby, the **Performance Measurement** node connects to it using the best parent node available (selected by the RPL routing protocol).

There are two solutions to change the default network name:

- Using Simplicity Studio 5 prior to building the project: open the wisun_soc_network_measurement.slcp file and go to the **SOFTWARE COMPONENTS** panel. After you select the **Wi-SUN/Application/Setting** component, you can configure the default connection settings. Under **Wi-SUN network configuration**, modify the **Network Name** parameter to match the name of your Wi-SUN network.
- Use the CLI available in the application. For more information, see Command Line Interface.

Each EFR32 radio board reference number defaults to a specific Wi-SUN PHY configuration matching its radio capabilities. To change the Wi-SUN PHY used by the project, refer to chapter 4 of UG495: Silicon Labs Wi-SUN Developer's Guide.

Start a Ping Test

You can trigger a ping test using either interface of the Wi-SUN Network Measurement application (LCD screen or CLI). The test consists of emitting ping packets on the Wi-SUN network to an IPv6 target (other Wi-SUN nodes or external IPv6 devices). This device replies with pong packets for each ping received. When the pong is received by the **Performance Measurement** node, the Wi-SUN Network Measurement application computes a round-trip latency on the communication link. During the test, the Wi-SUN Network Measurement application monitors several metrics to provide insights into the Wi-SUN stack behavior.

Retrieve Test Results

After a ping test has ended, test results are available. They sum up the following information:

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Ping packet loss percentage
- Minimum ping latency in milliseconds
- Maximum ping latency in milliseconds
- Average ping latency in milliseconds

In addition to the ping test results, the Wi-SUN Network Measurement application also monitors metrics associated with the Wi-SUN stack.

Impact of Wi-SUN signaling on ping test results

Due to the characteristics of a Wi-SUN network, where network management such as PAN advertising can take a certain time (it requires sending a PAN Advertisement on each channel, and there are often as much as 129 channels), using a ping to measure network latency needs to be taken cautiously.

Over several pings there will be mostly short ping durations, and some longer durations when network signaling takes precedence over data transmission. When network signaling occurs, the ping duration will be impacted.

The 'max' and 'average' ping results will be impacted by network management. To get a valid 'latency' value to compare between PHYs configurations, we'll only keep the 'min' ping duration of at least 100 pings in tables and graphs below. The ‘Wi-SUN SoC Network Measurement’ application is designed to allow this test.

> **Note**: The `network_size` parameter will have a significant impact on these 'signaling perturbations'. The smaller network sizes use more frequent network signaling and therefore have the highest impact.

Wi-SUN Metrics

To provide a better understanding on the Wi-SUN stack behavior the stack maintains internal statistics. The Wi-SUN Network Measurement application monitors several metrics during the tests:

- Lifetime
- MAC transmission count
- MAC transmission failed
- RPL rank
- Expected Transmission Count (ETX)
- Received Signal Level OUT (RSL out)
- Received Signal Level IN (RSL in)

The metrics are maintained for each neighbor device in the routed network (that is, the parent and all of the children in the RPL mesh tree). Each metric is described in the following sections.

Lifetime

The lifetime is a counter maintained with each neighbor to evaluate the state of the connection with this device. After each successful unicast communication with the neighbor, the counter is reset to 2200 seconds. When no unicast communication is exchanged with the neighbor, the counter is decreased each second.

MAC Transmission Count

The MAC transmission count (or MAC TX count) represents the number of packets transmitted over the air. A single ping attempt can result in several MAC-level transmission if the first attempts have not been successful. The number also considers packets not related to the ping test itself. Packets routed by the device to its parent and network maintenance related communications also increase this number.

MAC Transmission Failed

The MAC transmit failed (or MAC TX failed) represents the number of unsuccessful transmission attempts. In the Wi-SUN FAN specification, the peer must acknowledge every unicast packet. If a packet is not acknowledged within a given

Copyright © 2025 Silicon Laboratories. All rights reserved.

timeout, the transmission is considered failed. The packet is retransmitted by the Wi-SUN stack MAC layer until it reaches the maximum number of transmission attempts.

RPL Rank

The RPL rank represents the rank or hop distance from the tree root (that is, the Wi-SUN border router). For more information, see section 3.5 of RFC 6550.

Expected Transmission Count

The Expected Transmission Count (ETX) is an Exponentially Weighted Moving Average (EWMA) of the number of expected packet transmissions required for error-free reception at destination. The ETX is calculated as (frame transmission attempts)/(received frame acknowledgements) *128 with a maximum value of 1024, where 0 received frame acknowledgments sets ETX to the maximum value. This metric is maintained as part of the RPL protocol RFC 6550 and participates in the best parent selection.

Received Signal Level OUT

The Received Signal Level out (RSL out) is an EWMA of the received signal level for the node-to-neighbor direction. The RSL is calculated as the received signal level relative to standard thermal noise (290°K) at 1 Hz bandwidth or -174 dBm. This provides a range of -174 (0) to +80 (254) dBm.

Received Signal Level IN

The Received Signal Level in (RSL_in) is an EWMA of the received signal level for the neighbor-to-node direction. The RSL is calculated as the received signal level relative to standard thermal noise (290°K) at 1 Hz bandwidth (= -174 dBm). This provides a range of -174 (0) to +80 (254) dBm in 1 dBm steps. The RSL_in value is communicated through an information element in the Wi-SUN packet exchanges between the device and its neighbor.

Test Configuration

The Wi-SUN Network Measurement application provides several settings to configure the ping test. The capabilities vary depending on the interface used but some of the configurations are shared:

- **Packet count**: Defines the number of ping packets sent per target during the tests (default value 10). If the test targets the border router and the parent, 10 ping attempts would be sent to the border router followed by 10 attempts targeting the parent.
- **Packet length**: Defines the payload size of the ping packet sent to the targets (default value 40). The size corresponds to the ICMPv6 payload including type field (1 byte), code (1 byte), checksum (2 bytes), identifier (2 bytes), sequence number (2 bytes), and finally data field (dependent on the packet length configuration).
- **Target devices**: Defines which Wi-SUN devices are targeted by the ping test. There are three options:
  - Every known device, that is, the border router, the parent, and the device children
  - The border router
  - The parent

> **Note**: If the device running the Wi-SUN Network Measurement application is connected directly to the border router, the parent device and the border router are equivalent.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Lcd Screen Interface

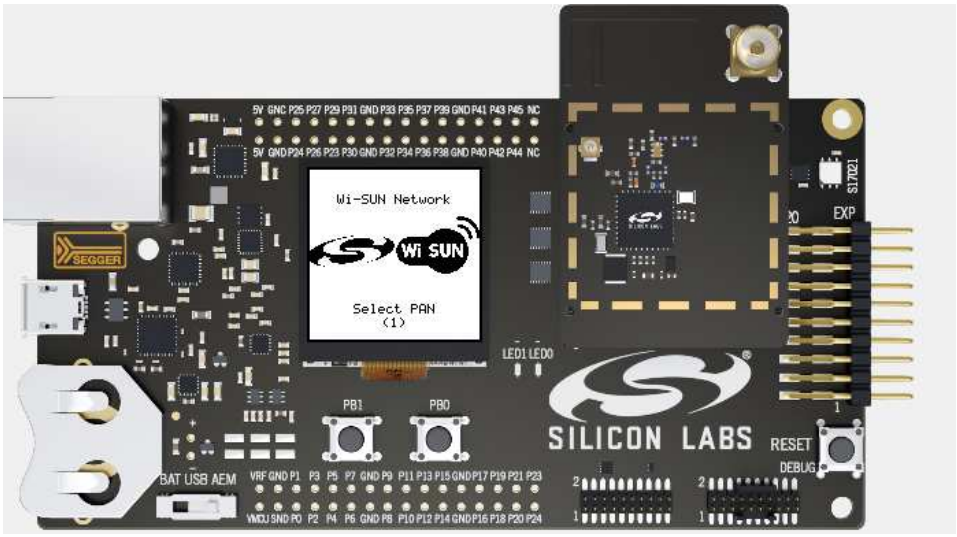
LCD Screen Interface

The first interface that can be used to interact with the Wi-SUN Network Measurement application is the mainboard LCD screen and two push buttons. They provide an easy-to-use interface to configure/start ping tests and consult the test results. The LCD interface can leverage most of the application features with few limitations related to the limited interface it offers (for example, it is not possible to input an IPv6 address to start a test with a “unknown” device).

> **Note**: When using the LCD screen interface, keep track of the actions associated with each push button because they can change depending on the context.

Connection Screen

The Wi-SUN Network Measurement application starts with a connection attempt to the Wi-SUN network configured in the project. The network name is displayed in the upper part of the screen. The connection steps are shown in the lower section (Wi-SUN connection steps 1 to 5 on the LCD screen).



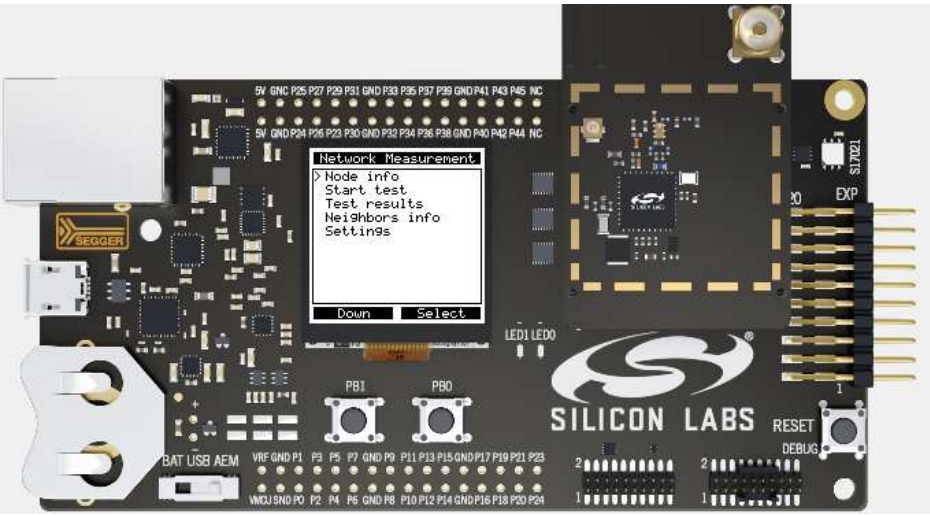
When the node is successfully connected to the Wi-SUN network, the Wi-SUN Network Measurement application switches the screen to display the main menu.

Note: If the Wi-SUN Network Measurement application is stuck in the connection step 1 **Select PAN**, verify the configuration used matches the border router configuration (network name and Wi-SUN PHY used).

Main Menu

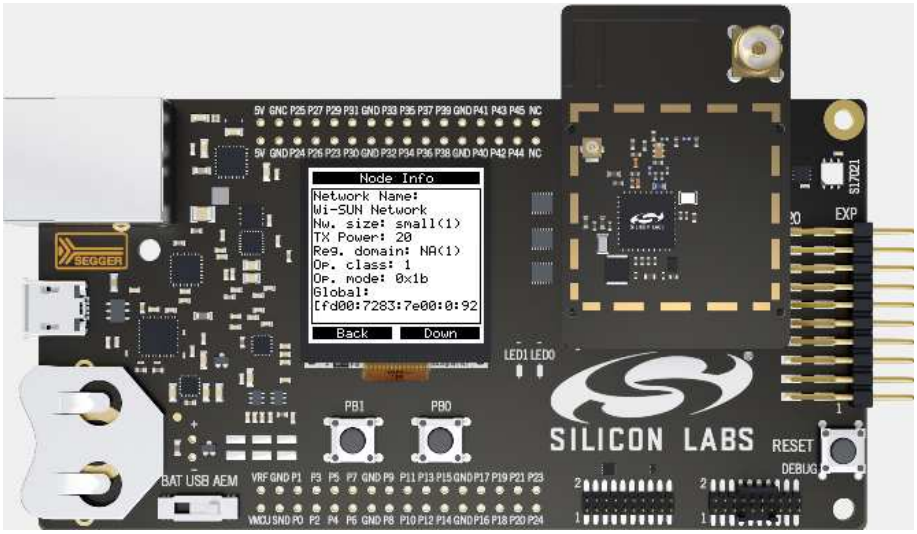
The Network Measurement main menu provides access to the Wi-SUN Network Measurement application main features. Scroll down through the options using push button 1 (**PB1**). Select a sub-menu using **PB0**.

Copyright © 2025 Silicon Laboratories. All rights reserved.



Node Info Screen

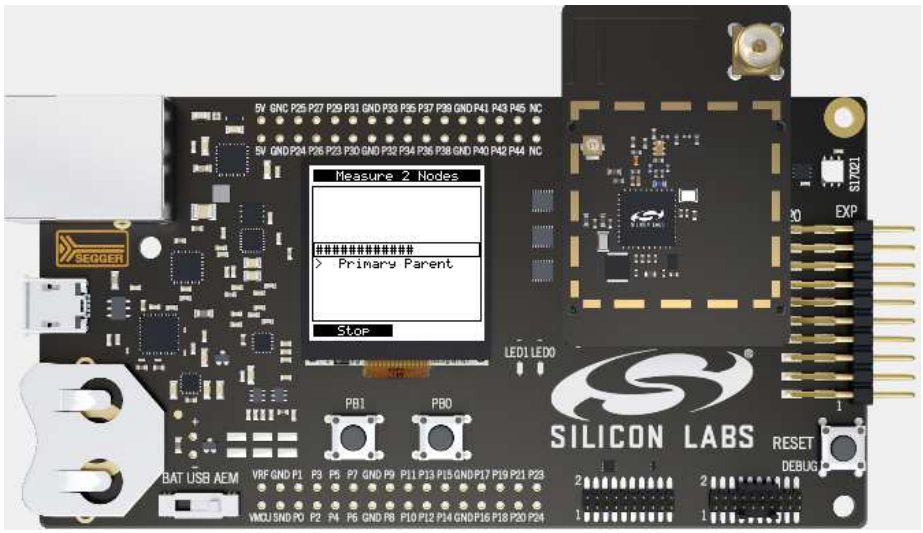
The Node Info screen displays information related to the Wi-SUN device and its configuration (network name, Wi-SUN PHY configuration, TX power, IPV6 addresses, and so on). You can scroll through the information available using **PB0**.



Start Test Option

If you select **Start Test** on the main menu, the Wi-SUN Network Measurement application starts a ping test with the settings previously configured with the other menus. A progress bar indicates the number of ping packets sent over the packet count configured for the test. The test is replicated for each configured target device (border router, parent, and children).

Copyright © 2025 Silicon Laboratories. All rights reserved.



You can stop the test for a given target at any time by pressing **PB1**. When the tests are completed, pressing **PB0** opens the test results display.

Test Results Display

You can access the Test Results display either after a test or from the main menu. The results are split for each available target. When you select a target, you access its test results, including the ping packet loss and associated percentage, the minimum/maximum/average ping latency, the test settings, and the associated RPL metrics.

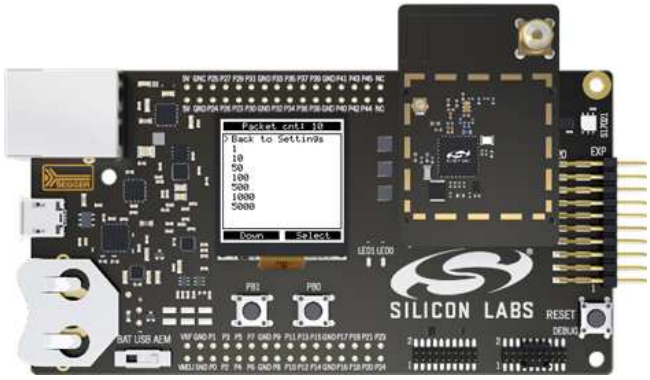
Neighbors Info Screen

The Neighbors Info screen provides access to metrics individualized per neighbor device. This includes the mandatory parent in the Wi-SUN network but also all the potential children using the device to communicate with the rest of the network. The information includes the device IPv6 address and numerous RPL-related metrics.

Settings Menu

The Settings menu enables you to configure the ping test settings like packet count, packet length, and target devices. In each sub-menu, the currently configured value is displayed in the upper black box. You can select a new value using the two push buttons.

In the following figure, the application packet count is currently configured to 10 packets.



Copyright © 2025 Silicon Laboratories. All rights reserved.

Command Line Interface

Command Line Interface

The CLI provides an alternative interface to the LCD screen with additional features and test scripting. In addition to exposing a broader API, the CLI enables you to:

- Configure the network name, network size, and TX output power.
- Control the connection state.
- Connect or disconnect the Wi-SUN node.
- Input IPv6 addresses to perform ping tests with devices other than the border router, parent, and children.
- Configure IPv6 UDP iperf tests

The Wi-SUN Network Measurement application leverages the common Wi-SUN application CLI for standard Wi-SUN actions. It adds a specific “measure” command that can start ping tests equivalent to the one triggered from the LCD screen interface.

Three command domains are available: `about`, `wisun`, and `iperf`. You can list these by executing `help`:

```
> help

about                Get info about the running app
iperf                iPerf2 commands
wisun                Wi-SUN commands
```

The `about` command domain is limited to displaying information about the current application (it is provided by the ‘app info’ component).

```
about
Wi-SUN Network Measurement Application
Versions:
  * Compiler (GCC): 10.3.1
  * Micrium OS kernel: 5.12.1
  * Wi-SUN: 1.1.0
  * mbed TLS: 3.1.0

\>
```

You can access the list of `wisun` commands by executing `wisun help` at the command prompt:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
> wisun help

join_fan10              Connect to a Wi-SUN network using FAN1.0 settings
join_fan11              Connect to a Wi-SUN network using FAN1.1 settings
join_explicit           Connect to a Wi-SUN network using explicit PHY settings
disconnect              Disconnect from the Wi-SUN network
set                   Set a variable
                    [*] empty | help | [string] Key [string] Value
get                   Get a variable
                    [*] empty | help | [string] Key
save                  Save variables to non-volatile storage
reset                 Reset variables to default settings
measure                 Measure a remote host or quick measurement
                    [string] Remote address or 'all' or 'parent' or 'br'
                    [uint16] Count of measurement packets
                    [uint16] Measurement packet length
ping                  Ping a remote host

                    [string] Remote address
```

And the list of `iperf` commands by executing `iperf help` at the command prompt:

```
> iperf help
 get                  Get configuration parameters
                     [*] empty | [string] Key
 set                  Set configuration parameters
                     [*] empty | [string] Key [string] Value
 server               Start preconfigured iPerf Server test
                     [*] help
 client               Start preconfigured iPerf Client test
                     [*] help
```

For `wisun` or `iperf` command domains, `set` and `get` functions are available to check (`<domain\> get`) or change (`<domain\> set`) current settings. To only list existing settings, use `<domain> get <subdomain> help` .

The Wi-SUN Network Measurement application maintains Wi-SUN network settings that you can access by executing `wisun get wisun` . This returns all the settings available in the Wi-SUN CLI component. You can check whether a setting is only readable by executing `wisun get wisun help` or if it can be modified (by executing `wisun set`) as shown in the following examples.

```
> wisun get wisun help

[rw] wisun.network_name
[rw] wisun.network_size
[rw] wisun.tx_power
[rw] wisun.regulatory_domain
[rw] wisun.operating_class
[rw] wisun.operating_mode
[ro] wisun.connection_state
[ro] wisun.ip_address_global
[ro] wisun.ip_address_link_local
[ro] wisun.ip_address_border_router
[rw] wisun.regulation                       Regional regulation [uint8]
[rw] wisun.regulation_warning_threshold             Transmission warning threshold in
                                    percent (-1 to disable) [int8]

[rw] wisun.regulation_alert_threshold               Transmission alert threshold in percent
                                    (-1 to disable) [int8]
```

Similarly, the writable settings are listed when you execute `wisun set wisun help` .

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
> wisun set wisun help

[rw] wisun.network_name
[rw] wisun.network_size
[rw] wisun.tx_power
[rw] wisun.regulatory_domain
[rw] wisun.operating_class
[rw] wisun.operating_mode
[rw] wisun.regulation                          Regional regulation [uint8]
[rw] wisun.regulation_warning_threshold        Transmission warning threshold in
                                          percent (-1 to disable) [int8]
[rw] wisun.regulation_alert_threshold          Transmission alert threshold in percent

                                          (-1 to disable) [int8]
```

This list is the same as the `get` list without the read-only ([ro]) values.

The `iperf` command domain has additional help about the possible parameters, accessible using `iperf get help`.

```
> iperf get help
Help of iperf 'get' and 'set' methods

Available sub-domains :
 options
 results

[csut]: corresponds to client/server/udp/tcp. Indicates in which case the parameter applies

Type 'iperf [get or set] \<sub-domain\> help'

eg. 'iperf get options help' to get the help of all options
   'iperf get options' for all current options values
   'iperf get results' for test results
   'iperf get options.\<option\>' for a specific option value
```

This shows that there are 2 `iperf` sub-domains: `options` and `results`.

Since options may differ depending on the test case (Client or Server, UDP or TCP), an indicator is added on each line:

```
[csut]: corresponds to client/server/udp/tcp. Indicates in which case the parameter applies
```

So, options can be displayed using `iperf get options help`.

```
iperf get options help
[csu ] options.port                Server port to listen on/connect to [uint16] (default 5001)
[c u ] options.remote_addr           IPv6 remote host address  [string]
[c u ] options.bandwidth             Unused if packet_number is set. Otherwise used to compute
                              Packet_number as bandwidth\*duration/buffer_length  [uint32]
                              (default 20000)
[csu ] options.bw_format             Unused if packet_number is set. One of [bits/s, Kbits/s,
                              Mbits/s, Gbits/s, bytes/s, Kbytes/s, Mbytes/s, Gbytes/s]
                              [string] (default bits/s)
[c u ] options.packet_number         Number of packets to send. (Rules out bandwidth)  [uint16]
                              (default 0=unset)
[c u ] options.buffer_length         buffer_length
[c u ] options.duration              Test duration in seconds [uint16] (default 10)
[csu ] options.interval              Seconds between periodic bandwidth reports  [uint16]
                              (default 1)
>
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Above, we see that the iperf server has only a limited number of options: `port`, `bw_format`, and `interval`, while the iperf client has more.

By default, `options.packet_number` is not set, and the actual value used during an iperf client test is computed based on `bandwidth`, `duration`, and `buffer_length`. This is used for most tests.

Similarly, results can be displayed using `iperf get results help`.

```
> iperf get results help
[csu ] results.json             Last test result in json format
[csu ] results.text             Last test result in text format
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

IPv6 UDP iperf Testing

IPv6 UDP iperf Testing

It is possible to test iperf throughput in both direction:

Using **iperf client**, the Wi-SUN node is the client sending Datagrams to an iperf server which needs to be started on another IPv6 address.

Using **iperf server**, the Wi-SUN node is the server and is waiting for datagrams coming from another IPv6 address on the defined options.port (default 5001, the usual iperf port).

In both cases:

- The Border Router must be started
- The Wi-SUN node(s) need to be connected to the Wi-SUN network
- We test using nodes directly connected to the Border Router (single hop). With multiple hops, the throughput is to be divided by the number of hops.
- The iperf server must be started before the client

Add IPv6 address to Border Router tun0

To allow using Linux regular iperf (iperf2) on the Border Router, the `tun0` interface needs to have a known IPv6 address.

We use the `IPv6_prefix` set for `wsbrd` with suffix `::1`. It is visible in the tun0 Iv6 address or in the wsbrd configuration file.

Retrieving the `tun0 IPv6` prefix:

`ip address show tun0 | grep global`

`inet6 **fd00:6172:6d00**:0:4de2:8ac8:d8cb:780f/64 scope global dynamic mngtmpaddr stable-privacy`

Adding a fixed/known IPv6 address to `tun0`:

`sudo ip -6 address add \<IPv6\_prefix[0:14]\>::1 dev tun0`

example:

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

Check:

`ip address show tun0 | grep global | grep ::1`

`inet6 **fd00:6172:6d00::1**/64 scope global`

Using this method, it is possible for any Wi-SUN node to know what this address is, checking its own `global` IPv6 address and replacing the last digits with `::1`.

Wi-SUN Node as client

We test the Wi-SUN node as a client (sending Datagrams to the server) using the Raspberry Pi supporting the Border Router application, wsbrd. The iperf server must be started first, then the client.

On the Linux Border Router:

- (One time) Install iperf2 (`apt-get install iperf`) if not already installed.
- (After each restart of wsbrd) Add a fixed/known IPv6 address to tun0 as described above.

Copyright © 2025 Silicon Laboratories. All rights reserved.

- **Note**: You will need to set this IPv6 address on the client as `options.remote_addr`.
- Start iperf2 as a UDP IPv6 server:
  `iperf --server --udp --ipv6\_domain --interval 1 –-format KBytes`
  - or (using short arguments)
    `iperf -s -u -V -i 1 -f KBytes`
- The iperf server will start waiting for incoming data:

```
------------------------------------------------------------
Server listening on UDP port 5001
UDP buffer size:  176 KByte (default)
------------------------------------------------------------
```

On the Wi-SUN Node:

> **Note**: Apart from the server's IPV6 address, all default values should match a typical iperf test and can keep their default values.

- Check that the Wi-SUN Node is connected to the Wi-SUN network:
  `> wisun get wisun.connection_state`
  `wisun.join_state = Operational (5)`
- Check that the parent is the Border Router (to make sure we're testing on a single hop):
  TBD (not available in current iperf app)
- Set the IPv6 address to match the fixed server's IPV6 address set above:
  `wisun get wisun.ip_address_global`
  `wisun.ip_address_global = **fd00:6172:6d00**:fe6d:5a15`
  (where the IPV6 prefix is the 14 first characters)
  `iperf set options.remote_addr <server's IPV6 address>`
  example:
  `iperf set options.remote_addr* **fd00:6172:6d00::1**`
- (optionally) Set the bandwidth format. This is the unit used by `options.bandwidth`. we recommend setting it to `bits/s`.
  `iperf set options.bw_format bits/s`
- (optionally) Set the number of payload bytes to the maximum before fragmentation occurs using UDP, i.e.1234. This is to maximize the throughput.
  `iperf set options.buffer_length 1234`
- (optionally) Set the test duration in seconds (default = 10)
  `iperf set options.duration 60`
- (optionally) Set the iperf port (default = 5001)
  `iperf set options.port 5001`
- (optionally) Set the iperf bandwidth (default = 20000). The bandwidth should be in a range attainable by the selected Wi-SUN PHY. In our initial tests, we're using Wi-SUN FAN, NA-915MHz, 1b (2FSK 50kbps mi=1.0), so 50 kbits/sec is the maximum we could expect if using 100% of the link as a permanent link (which is not the case, according to Wi-SUN specification). Half this is what we will reasonably expect to achieve.
  `iperf set options.bandwidth 20000`
- (optionally) Set the display interval in seconds (default = 1):
  `iperf set options.interval 1`
- (optionally) Set the number of packets to transmit (default = 0/unset).
  - Set this only if you want to set an exact number of packets instead of `options.bandwidth` for `options.duration` seconds.
  `iperf set options.packet_number 10`
- Check your options settings (only checking relevant options, marked with [c.u.])

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
iperf get options
[csu ] options.port = 5001
[c u ] options.remote_addr = "**fd00:6172:6d00::1**"
[c u ] options.bandwidth = 20000
[csu ] options.bw_format = bits/s
[c u ] options.packet_number = 0
[c u ] options.buffer_length = 1234 bytes
[c u ] options.duration = 60 s
[csu ] options.interval = 1 s
```

- Start the iperf client test:
  `iperf client`

> **Note**: In this setup, the iperf server being run on a Linux machine doesn't need to be restarted upon each test. It can be left running.

Test results are displayed in the Wi-SUN Node's console (Client) and in the Linux (Server) console in a format very similar to regular iperf2.

Wi-Sun Node as Server

Here we test the Wi-SUN node as an iperf server (receiving Datagrams from clients) using the Raspberry Pi supporting the Border Router application, wsbrd as the client. As before, the iperf server must be started first, then the client. The difference with regular iperf is that **iperf server** must be called once per test.

On the Wi-SUN Node:

> **Note**: Apart from the server's IPV6 address, all default values should match a typical iperf test and can keep their default values.

- Check that the Wi-SUN Node is connected to the Wi-SUN network:
  `> wisun get wisun.connection_state`
  `wisun.connection_state = Operational (5)`
- Check that the parent is the Border Router (to make sure we're testing on a single hop):
  TBD (not available in current iperf app)
- (optionally) Set the bandwidth display format.
  `iperf set options.bw_format bits/s`
- (optionally) Set the iperf port (default = 5001)
  `iperf set options.port 5001`
- (optionally) Set the display interval in seconds (default = 1)
  `iperf set options.interval 1`
- Check your options settings (only checking relevant options, marked with [.su.]):

```
iperf get options
[csu ] options.port = 5001
[csu ] options.bw_format = bits/s
[csu ] options.interval = 1 s
```

- Display the global IPv6 address:
  `wisun get wisun.ip_address_global`
  `wisun.ip_address_global = **fd00:6172:6d00:0:5e02:72ff:fe96:c6a3**`
- Start the iperf server test:

Copyright © 2025 Silicon Laboratories. All rights reserved.

`iperf server`

> **Note**: In this setup, the iperf server is running on the Wi-SUN node and only accepts one test. **iperf server** needs to be relaunched for each test.

On the Linux Border Router:

- (One time) Install iperf2 (`apt-get install iperf`) if not already installed.
- Start iperf2 as a UDP IPv6 client using the Wi-SUN node's global IPv6 address.
  `iperf --client* **fd00:6172:6d00:0:5e02:72ff:fe96:c6a3** *--udp --ipv6_domain --interval 1 --time 10 --bandwidth 20 --format k`
  - or (using short arguments)
    `iperf -c **fd00:6172:6d00:0:5e02:72ff:fe96:c6a3** -u -V -i 1 -t 10 -b 20 -f k`
    Test results are displayed in the Wi-SUN Node's console (Client) and in the Linux (Server) console in a format very similar to regular iperf2.

Wi-SUN Throughput Measurement

Maximizing Throughput

To maximize the iperf throughput, we set on the Wi-SUN node `options.buffer_length` to the maximum size before fragmentation occurs of 1234 bytes. In this case, the Wi-SUN overhead (on top of UDP) is minimal, and user throughput is maximum.

Again, to maximize the iperf throughput, the iperf implementation in the Wi-SUN Network Measurement application uses the UDP protocol (TCP would require acknowledges which would negatively impact the throughput).

Finding the Throughput Limit

Due to the characteristics of a Wi-SUN network, where PAN advertising can take a certain time (it requires sending a PAN Advertisement on each channel, and there are often as much as 129 channels), using an iperf test to measure network throughput needs to be taken cautiously.

During the iperf test there will be periods of time when iperf data flows without interruption, leading to a momentary 'best case' throughput.

There will also be periods when network signaling takes precedence over data transmission. When network signaling occurs, the iperf throughput will be negatively impacted.

To get a valid throughput value to compare between PHY configurations, we look for the **max sustainable throughput without packet loss** on a long duration iperf test of **at least 60 seconds**. The idea is that a 60 second test with a given bandwidth will fill up the buffers up to overflowing if the stack is unable to cope with the requested throughput.

The effect of such overflowing is regularly lost packets during the duration of the test, reflecting in the final server report as a non-zero lost packets count.

The test consists in finding the maximum `options.bandwidth` which allows 0 lost packet over 60 sec. We approach this limit through several consecutive iperf client tests. The figure we look for is the number of lost packets on the server side (since we're using UDP, the client doesn't know if the data passes through).

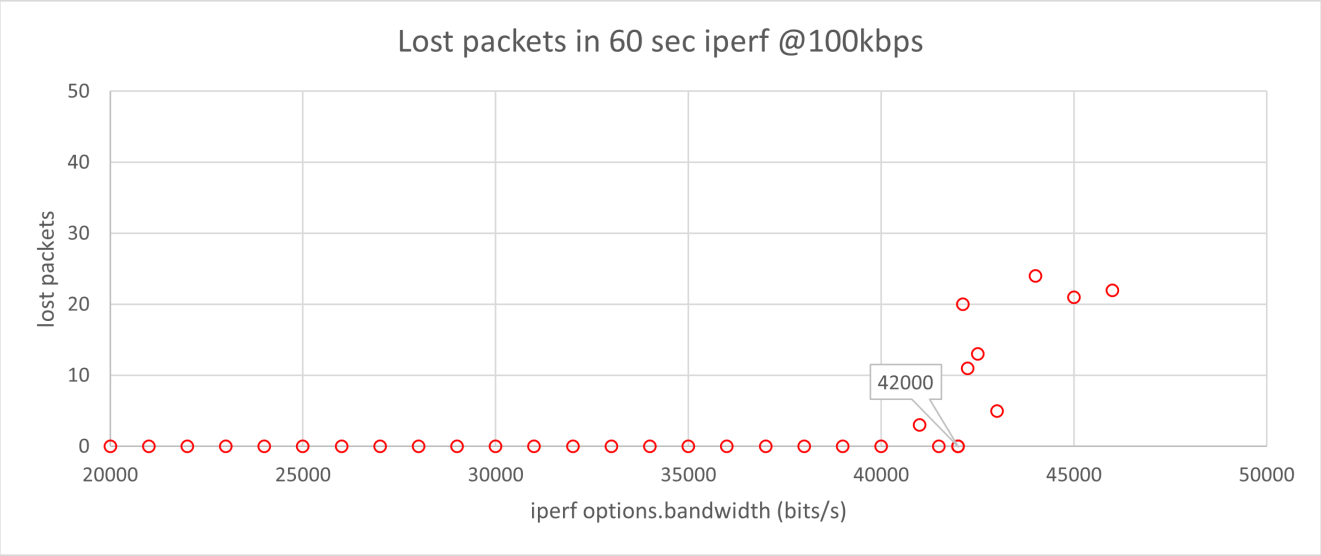
- If the number of lost packets is 0, the Wi-SUN PHY can withstand such a throughput. Increase the `options.bandwidth` and retest.
- If the number of lost packets is NOT 0, the Wi-SUN PHY is unable to withstand such a throughput. Decrease the `options.bandwidth` and retest.

Do this in decreasing steps until you have a small enough `options.bandwidth` step (we used a 1000 bit step in our tests).

The value we keep is the maximum `options.bandwidth` without packet loss.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Below is an illustration of such a limit search, where the limit is 42000 bit/s:



Copyright © 2025 Silicon Laboratories. All rights reserved.

CoAP Communication Between The Linux Border Router And The Network Performance Application

CoAP Communication Between the Linux Border Router and the Network Performance Application

The Network Measurement Application supports a CoAP server to allow the Wi-SUN Border Router to get data from the node using the linux libcoap client. Follow the steps below to get CoAP network performance data on your Border Router, without the need to directly access the CLI on the Wi-SUN node.

Install the libcoap library on the Border Router

```
$ sudo apt-get install libcoap-1-0
$ sudo apt-get install libcoap-1-0-bin
```

Get your node's global address

From the node cli, make sure that you are connected to the network.

```
> wisun get wisun.connection_state
wisun.connection_state = Operational (5)
```

Then, get your global IPv6 address:

```
> wisun get wisun.ip_address_global
wisun.ip_address_global = fd12:3456::20d:6fff:fe20:be0a
```

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 Network Performance application, the available resources can be retrieved using the libcoap GET method with the standard discovery entry-point.

```
$ coap-client -m get -N coap://[fd12:3456::20d:6fff:fe20:be0a]:5683/.well-known/core
v:1 t:NON c:GET i:32bf {} [ ]
</cli>;ct=40,</cli/nbrinfo>;rt="nbrinfo";if="cli",</cli/iperf>;rt="iperf";if="cli",</cli/ping>;rt="ping";if="cli"
```

The above shows that we have access to the following commands:

```
</cli>;ct=40,

</cli/nbrinfo>;rt="nbrinfo";if="cli",

</cli/iperf>;rt="iperf";if="cli",

</cli/ping>;rt="ping";if="cli"
```

Check the node and Border Router info using:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
**$ coap-client -m get -N  coap://[fd00:6172:6d00:0:b6e3:f9ff:fea6:3aa]:5683/cli/nbrinfo**

v:1 t:NON c:GET i:ff62 {} [ ]
{
"Border Router":
{"addr":"fd00:6172:6d00:0:92fd:9fff:fe6d:5a15","lt":0,"txc":0,"txf":0,"txmsc":0,"txmsf":0,"rplr":0,"etx":0,"rslo":0,"rsli":0},
"Primary Parent":
{"addr":"fd00:6172:6d00:0:92fd:9fff:fe6d:5a15","lt":2037,"txc":2096,"txf":9,"txmsc":0,"txmsf":0,"rplr":128,"etx":150,"rslo":142,"rsli":138},
"Child0":
{"addr":"fd00:6172:6d00:0:5e02:72ff:fe96:c97f","lt":2159,"txc":162,"txf":0,"txmsc":0,"txmsf":0,"rplr":65535,"etx":276,"rslo":152,"rsli":151}
}
```

Calling iperf using CoAP

It is possible to get the available iperf options using the `-e` text payload coap-client parameter:

$ coap-client -m get -N -e "iperf help" coap://[fd00:6172:6d00:0:b6e3:f9ff:fea6:3aa]:5683/cli/iperf

```
v:1 t:NON c:GET i:4807 {} [ ]

iPerf Remote Control CLI Help.

 Writable options:
   -options.remote_addr
   -options.bandwidth
   -options.duration
   -options.interval
 Example: iperf set options.bandwidth 10000

 Start iPerf command: iperf <client|server>
```

Set the iperf options using:

$ coap-client -m get -N -e "iperf set options.bandwidth 10000"
coap://[fd00:6172:6d00:0:b6e3:f9ff:fea6:3aa]:5683/cli/iperf

```
v:1 t:NON c:GET i:d665 {} [ ]
10000
```

$ coap-client -m get -N -e "iperf set options.remote_addr fd00:6172:6d00:0:92fd:9fff:fe6d:5a15"
coap://[fd00:6172:6d00:0:b6e3:f9ff:fea6:3aa]:5683/cli/iperf

```
v:1 t:NON c:GET i:4c8a {} [ ]
fd00:6172:6d00:0:92fd:9fff:fe6d:5a15
```

Launch the iperf test using:

$ coap-client -m get -N -e "iperf client" coap://[fd00:6172:6d00:0:b6e3:f9ff:fea6:3aa]:5683/cli/iperf

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
v:1 t:NON c:GET i:10d8 {} [ ]
{
  "Test-3": {
    "id":     3,
    "status": "DONE",
    "error":  "NONE",
    "options": {
      "mode":        "CLIENT",
      "protocol":    "IPV6_UDP",
      "port":        5001,
      "remote_addr": "fd00:6172:6d00::192fd:9fff:fe6d:5a15",
      "bandwidth":   10000,
      "packet_nbr":  10,
      "buf_len":     1234,
      "duration_ms": 10000,
      "win_size":    0,
      "persistent":  false,
      "interval_ms": 1000,
      "bw_format": "bits/s"
    },
    "statistic": {
      "nbr_calls":            10,
      "bytes":                12340,
      "tot_packets":          10,
      "nbr_rcv_snt_packets":  10,
      "errs":                 0,
      "transitory_error_cnts": 0,
      "last_recv_pkt_cnt":    0,
      "ts_curr_recv_ms":      0,
      "ts_prev_recv_ms":      0,
      "ts_curr_sent_ms":      19794004,
      "ts_prev_sent_ms":      0,
      "udp_jitter":           0,
      "udp_rx_last_pkt":      0,
      "udp_lost_pkt":         0,
      "udp_out_of_order":     0,
      "udp_dup_pkt":          0,
      "udp_async_error":      false,
      "end_err":              false,
      "ts_start_ms":          19783880,
      "ts_end_ms":            19794012,
      "bandwidth":            9743,
      "finack_tot_len":       0,
      "finack_duration_ms":   0,
      "finack_pkt":           0
    }
  }
}
```


Copyright © 2025 Silicon Laboratories. All rights reserved.

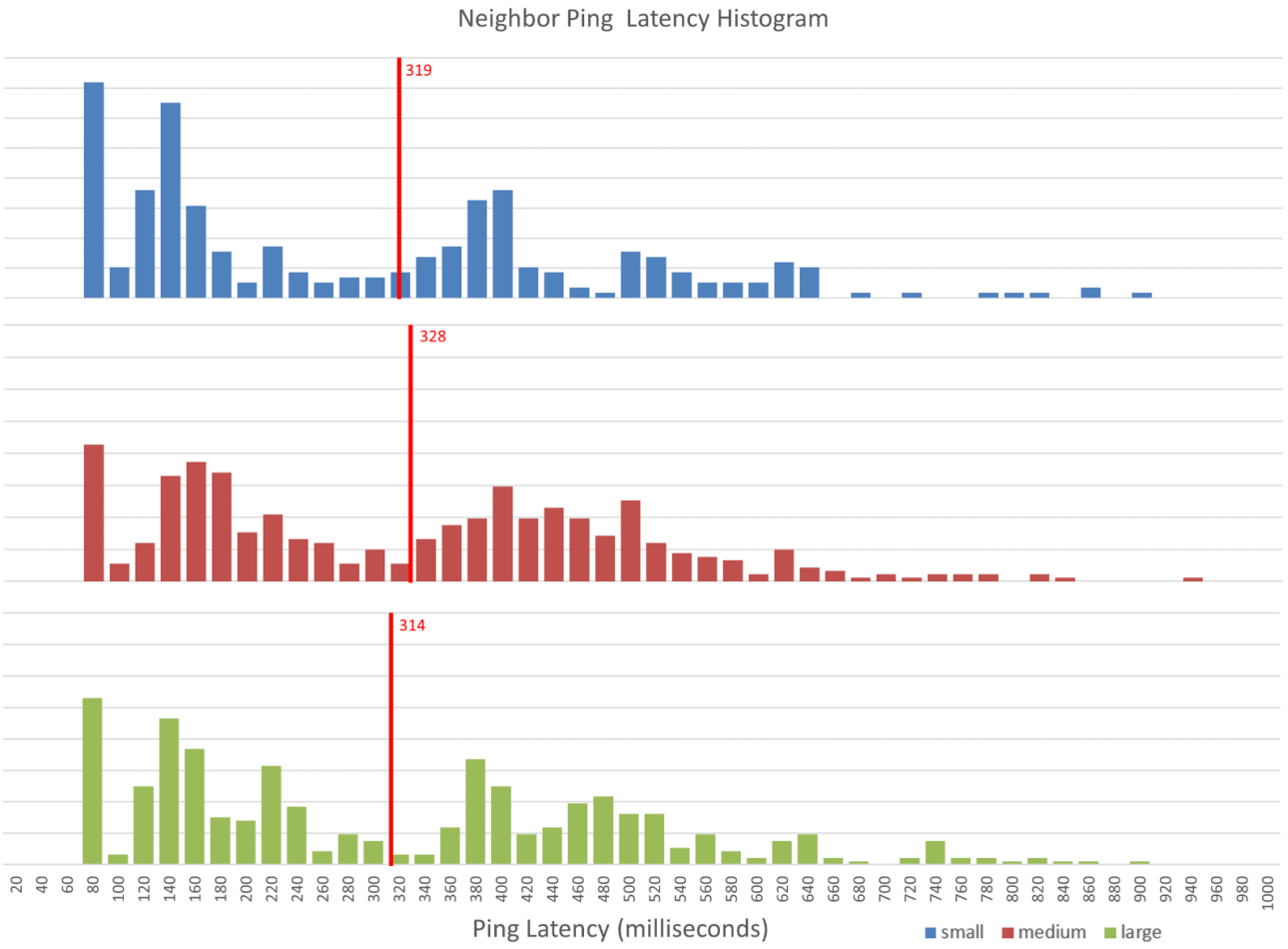
Ping Latency Improvement

Ping Latency Improvement

Using the Wi-SUN Network Measurement application in its default configuration (Lowest speed Wi-SUN PHY, Network size set to *small,* and default broadcast dwell interval), the ping latency performance is not optimal. This chapter describes ways to optimize the solution ping latency and explains some of the trade-offs associated with the modifications.

AN1330 Application Note

AN1330: Silicon Labs Wi-SUN Mesh Network Performance reports the ping latency internal tests performed by Silicon Labs in real-world deployments. The tests focus on characterizing the expected Wi-SUN stack latency in point-to-point communications but also in round trip messages traveling over several hops. The tests are performed with the Wi-SUN stack default (non-optimized) configuration using the 50 Kbps North America Wi-SUN PHY. The results provide a good reference point to compare your own test results. Keep in mind that using a different region PHY impacts the results in the following figure.

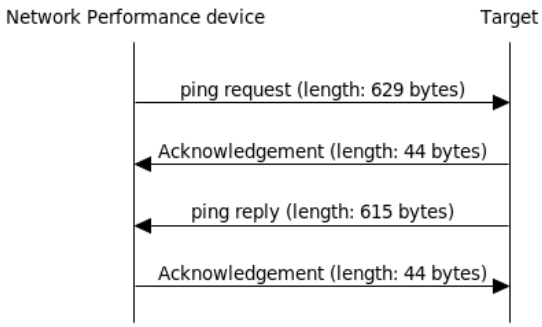


Wi-SUN PHY Impact

The first critical configuration impacting the ping latency is the Wi-SUN PHY which is used by every device in a Wi-SUN network. A point-to-point ping is composed of four messages over the air. A ping with a payload of 512 bytes triggers 1332

Copyright © 2025 Silicon Laboratories. All rights reserved.

bytes exchanged on the air. It represents approximately 30 milliseconds spent sending messages on the radio medium (see the following figure).



The Wi-SUN PHY also impacts the transmissions triggered to manage the Wi-SUN network. Both the speed and number of channels available in the PHY factor into the penalty that is applied to a ping latency sent close to a network management event. The two main time-consuming network management mechanisms are the PAN advertisement and PAN configuration events. For example, a PAN advertisement sent using the 50 kbps North America PHY using 128 channels holds the radio for approximately two seconds (about 15 milliseconds spent on each channel). The PAN advertisement and configuration events explain the periodical delays that can be noticed during a long ping test. Using a Wi-SUN PHY with a higher modulation speed and fewer channels highly reduces the impact of those network management events on the ping latency (for example, 300 kbps North America PHY with 41 channels). The drawback or trade-offs are the effective distance for a good communication between nodes in the network is reduced. Finally, you need to consider how changing the network size setting can affect the network management event intervals. See *Network Configuration and Size Setting* below.

In addition to the PHY speed itself, the frequency hopping mechanism and the associated channel switches can delay transmissions of packets to prevent sending a message on a channel change. The default unicast dwell interval is 255 milliseconds.

Network Configuration and Size Setting

The network configuration plays an integral role in the way packets flow through the Wi-SUN network. These network configurations impact the ping latency:

- The border router broadcast interval defines at which interval a broadcast dwell interval happens. By default, the border router broadcast interval is set to 1 seconds and 20 milliseconds. You can modify it on the border router side. The configuration of the border router broadcast interval propagates throughout the Wi-SUN network and every device part of the network uses the same interval. If an application focuses on unicast messages, Silicon Labs recommends that you increase the border route broadcast interval.
- The broadcast dwell interval defines the time of a broadcast frequency hopping slot (by default, 255 milliseconds equal to the unicast default dwell interval). The longer the broadcast dwell interval, the less time remains for unicast communications. Silicon Labs recommends that you reduce the broadcast dwell interval for a unicast-focused application. You change this configuration on the border router side.
- The network size setting dictates the intervals between network management events among other things. To optimize the ping latency, Silicon Labs recommends that you use the *large* or *medium* configuration depending on your network deployment size to increase the network management exchanges interval. Be aware of the potential trade-off with the network connection speed and network configuration update speed.

Considering all the options available for a Wi-SUN solution developer, you can greatly improve the ping latency from the initial results retrieved from the default solution configuration. Keep in mind that every setting change is a trade-off with another Wi-SUN network performance component.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Direct Connect

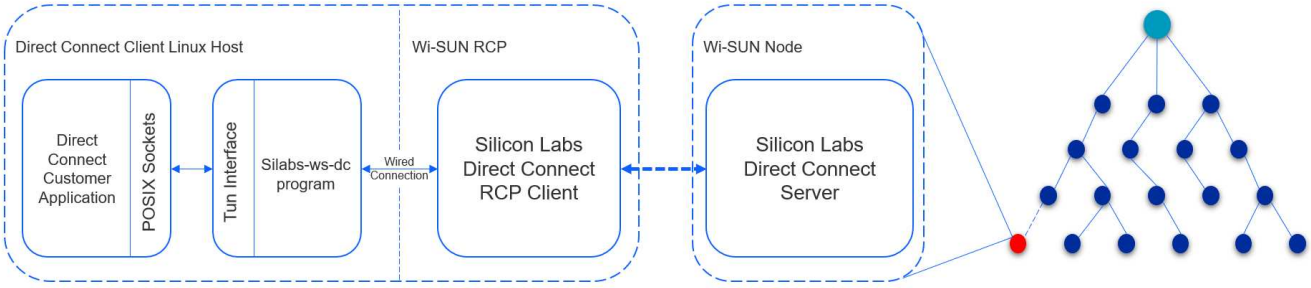
SiLicon Labs Direct Connect Tool

In the fast-paced world of industrial IoT and on-site debugging, maintaining reliable and direct communication with devices is critical. Yet, traditional methods often depend on the device's integration within its network, adding layers of complexity when immediate access is needed. Direct Connect is a groundbreaking interface designed to bridge that gap through a seamless wireless link that supports IPv6 traffic.

What sets Direct Connect apart is its independence from the Wi-SUN network. Whether the device is fully joined, in the process of joining, or completely disconnected from the network, Direct Connect ensures uninterrupted access. This capability makes it an invaluable tool for industrial implementations, where devices often operate in dynamic environments with evolving network conditions.

With Direct Connect, technicians and engineers can directly interface with Wi-SUN devices to perform diagnostics, deploy updates, or test configurations without the need to rely on the device's network status. This Silicon Labs solution not only simplifies on-site debugging but also enhances operational efficiency by eliminating the downtime associated with network dependencies.

> **Limitations**Direct Connect Tool is only supported on FFNs, LFNs do not support this feature.



Prerequisites

Direct connect well functioning relies on two main entities, the Direct Connect client that runs on a Linux host plus the RCP, and the Direct Connect Server that runs on the targeted Wi-SUN node. For this user guide we will need:

- 2x Wi-SUN Compatible Radio Boards
- A Linux host that can be a Raspberry Pi

And on the software side, we will use:

- Simplicity Studio v5 with SiSDK v2024.12.0 or later
- wisun-br-linux v2.2.0 or later

Setup Preparation

Linux Host

The Direct Connect tool is delivered with the wisun-br-linux tools package and called `silabs-ws-dc`.

Copyright © 2025 Silicon Laboratories. All rights reserved.

To install `silabs-ws-dc` on the Linux host, start by cloning the wisun-br-linux repository.

```
git clone https://github.com/SiliconLabs/wisun-br-linux.git
```

After cloning the repository, refer to Compiling section in `wsbrd-br-linux` to install the necessary dependencies and mbedTLS. Then, run the following `cmake` command instead of the one in the readme to install `silabs-ws-dc`.

```
mkdir build && cd build && cmake -G Ninja -DCOMPILE_DEVTOOLS=true ..
ninja
sudo ninja install
```

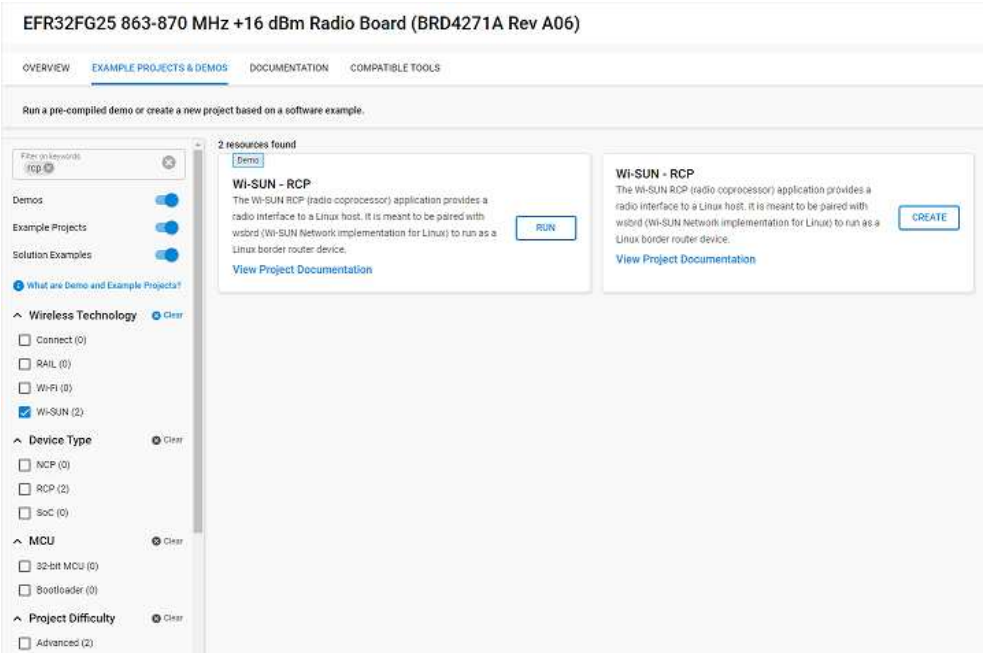
Wi-SUN RCP

The `silabs-ws-dc` tool uses the same RCP that works with wsbrd that is delivered in SiSDK 2024.12.0. To get the RCP project, install Simplicity Studio v5 and follow this guide to install the SiSDK Package.

After installing Studio and the required SDK follow those steps to flash the Wi-SUN RCP to the board.

Flashing the Wi-SUN RCP

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.



Wi-SUN Node

The *Wi-SUN - CLI example* provide a set of command lines to evaluate the Direct Connect tool. The following steps explains how to flash the *Wi-SUN - CLI example* to the board.

Flashing the *Wi-SUN - CLI example*

1. In the **Debug Adapters** view, select the device to be used as the Wi-SUN CLI.
2. Navigate to the **EXAMPLE PROJECTS & DEMOS** tab and turn off the **Example Projects** filter.
3. Click **RUN** next to the *Wi-SUN - CLI example* project.

Direct Connect Configuration

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configuring the Direct Connect tool involves setting up two key components: the **Direct Connect server** and the **Direct Connect client**. For a successful and secure connection, it's critical that both the server and client configurations are synchronized, particularly regarding the Pairwise Master Key (PMK) and the Wi-SUN node MAC address.

The PMK is the encryption key that ensures secure communication between the server and the client. Both sides must use the same PMK to establish a trusted and encrypted link. A mismatch in the PMK will result in connection failure, as the Direct Connect tool relies on this shared key for authentication and data protection.

Having the same PMK on both sides allows to establish a direct connection without performing the Wi-SUN authentication flow (EAP-TLS), removing the need of an authentication server and additional certificates.

Direct Connect Client

The Direct Connect client's configuration is managed through the `silabs-ws-dc` configuration file, allowing precise customization to establish a secure link with a Wi-SUN device. This connection is built around two critical parameters: the MAC address (EUI-64) of the target device and the Pairwise Master Key (PMK). These elements ensure that communication is uniquely tied to the intended device, providing robust security and preventing unauthorized access.

An example configuration file is provided in the `wisun-br-linux/examples` directory. You can use this example as a starting point by either modifying it directly or creating a new configuration file based on your specific setup requirements.

Below is an example configuration that highlights key settings:

```
# Path of the serial port connected to the RCP.
uart_device=/dev/ttyACM0

# Serial baudrate in bps
uart_baudrate=115200

# PHY Configuration
domain=EU
chan_plan_id=32
phy_mode_id=1

# Target MAC address
target_eui64=b4:35:22:ff:fe:98:29:ff

# Pairwise Master Key
target_pmk=34:ba:32:26:a0:b2:ad:66:7c:9f:66:02:e5:db:75:77:dd:bd:5d:2b:34:3a:93:06:2b:90:c0:7b:e2:8e:4e:54
```

Direct Connect Server

The server-side configuration must mirror the client in terms of the PMK value. While the MAC address identifies the target device on the client side, the server only needs the shared PMK and the same PHY configuration of `silabs-ws-dc` to establish a secure session.

To configure the Wi-SUN Node, connect to the console in Simplicity Studio. In the **Debug Adapters** view, right click on the Wi-SUN node board and select **launch Console**. To get a prompt on the Console, go to the *Serial 1* tab and press Enter.

Setting the Pairwise Master Key

Run the following command in the Wi-SUN node console to set the PMK:

```
wisun set wisun.direct_connect_pmk 34:ba:32:26:a0:b2:ad:66:7c:9f:66:02:e5:db:75:77:dd:bd:5d:2b:34:3a:93:06:2b:90:c0:7b:e2:8e:4e:54
```

Enable Direct Connect

Once the PMK is set, you need to explicitly enable the Direct Connect feature on the server. This step activates the feature, allowing the Direct Connect server to connect with the Direct Connect client.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Run the following command to enable Direct Connect:

```
wisun set_direct_connect_state 1 (1 to enable, 0 to disable)
```

Direct Connect in Action

Establishing the Direct Connect Link

To establish a Direct Connect link, the process begins with the Direct Connect client initiating communication with the target device.

Run `silabs-ws-dc` to Initiate the Connection

On the Linux host execute the following command:

```
sudo silabs-ws-dc -F ./silabs-ws-dc.conf
```

This command starts the Direct Connect client using the configuration file *silabs-ws-dc.conf* and sends a Direct Connect Solicit frame to the target device. This frame serves as a request for the device to establish a connection with the client.

Receiving the Direct Connect Request

On the targeted Wi-SUN device, the *Wi-SUN - CLI example* will indicate receipt of the request with a message like:

```
[Direct Connection request from fe80::b635:22ff:fe98:29b2]
```

> The `fe80::b635:22ff:fe98:29b2` is the link-local IPv6 address used by `silabs-ws-dc` . This address will vary depending on your specific setup.

Accepting the Direct Connect Link

To accept the incoming connection request, run the following command on the *Wi-SUN - CLI example*:

```
wisun accept_direct_connect_link fe80::b635:22ff:fe98:29b2
```

If successful, the *Wi-SUN - CLI example* will display:

```
[Accepted connection request]
```

Confirmation of a Successful Connection

Once the connection is fully established, both the *Wi-SUN - CLI example* and the `silabs-ws-dc` tool will provide confirmation messages:

- On the Wi-SUN CLI example:

  ```
  [Direct Connect Link fe80::b635:22ff:fe98:29b2: connected]
  ```

- On the `silabs-ws-dc` logs:

  ```
  Direct Connection established with b4:35:22:ff:fe:98:29:ff
  b4:35:22:ff:fe:98:29:ff reachable at fe80::b635:22ff:fe98:29ff
  ```

Using and Managing the Direct Connect Link

Copyright © 2025 Silicon Laboratories. All rights reserved.

Once the Direct Connect link is established, it provides a secure, point-to-point communication channel that supports UDP, TCP, and ICMP traffic. This enables a wide range of networking operations such as testing, debugging, and data exchange between the Direct Connect Client and the Direct Connect Server.

Using the Link

- **Versatility:** The link operates independently of the Wi-SUN network status, meaning it functions whether the device is connected to a Wi-SUN network or not.
- **Direct Communication:** Communication is exclusively between the Direct Connect Client and Server. Routing to other devices within the Wi-SUN network is not supported. This ensures that the connection remains private and direct, focusing solely on the configured devices.

This direct access is particularly useful for performing device-specific diagnostics, deploying configurations, or running network tests without relying on the broader Wi-SUN infrastructure.

Closing the Link

To terminate the Direct Connect link, simply stop the `silabs-ws-dc` process on the client using `Ctrl+C` or the following command if the client is running in the background:

```
sudo pkill silabs-ws-dc
```

Once the client stops, the link will disconnect automatically. After 60 seconds, the *Wi-SUN - CLI example* will display the following message, confirming the disconnection:

```
[Direct Connect Link fe80::b635:22ff:fe98:29b2: disconnected]
```

Connection Troubleshooting

Connection loss

If the Direct Connect client loses connection with the server, it will automatically try to reconnect by sending Direct Connect Solicit frames again and indicate the following:

```
Direct Connection with b4:35:22:ff:fe:98:29:ff lost, attempting to reconnect...
```

Connection failure

If the `silabs-ws-dc` is unable to establish a Direct Connection with the server, it will indicate the following:

```
b4:35:22:ff:fe:98:29:ff is unreachable, please check your configuration
```

In that case, make sure that the used parameters are set accordingly on both sides, refer to Direct Connect Configuration.

The Wi-SUN CLI may also indicate the following after accepting the direct connection:

```
[Direct Connect Link fe80::b635:22ff:fe98:29b2: error]
```

In that case, make sure the PMK used is the same on both sides and restart `silabs-ws-dc`.

Final Thoughts

The Direct Connect link offers a powerful and secure way to interact with Wi-SUN devices, making it an essential tool for industrial IoT applications. Its independence from the Wi-SUN network ensures flexibility, while its direct communication

Copyright © 2025 Silicon Laboratories. All rights reserved.

channel guarantees simplicity and reliability. Managing the link, from initiation to closure, is straightforward, making it a valuable asset for engineers and developers.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Dynamic Device Provisioning

Wi-SUN Dynamic Data Provisioning

Dynamic Data Provisioning (DDP) is a powerful tool designed to streamline the generation and injection of Wi-SUN certificates and device keys. This innovative solution, comprising a set of Python scripts available in the wisun application repo repository, interacts seamlessly with Wi-SUN devices. The core of this tool is the **Wi-SUN - DDP** application that can run in RAM, to provide a robust and flexible approach to device provisioning. In the following sections, we explore the tool by providing a step by step guide to generate the Wi-SUN certificates and provision the Wi-SUN devices with the generated keys.

wisun application repo

Prerequisites

The following requirements are key to start using the Wi-SUN Dynamic Data Provisioning tool:

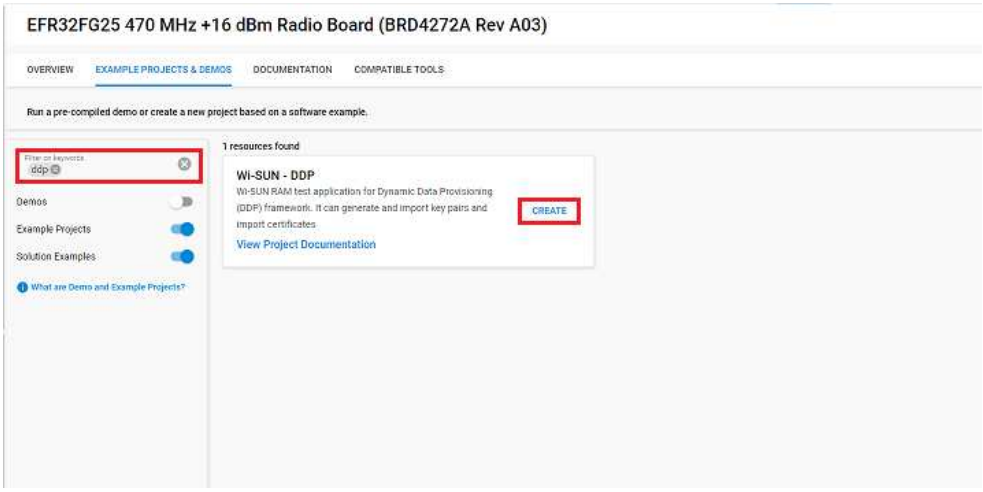
- The DDP tool scripts hosted in wisun_applications.
- The **Wi-SUN - DDP** application delivered with Simplicity Studio v5 SiSDK 2024.12.0 or later.
- One Wi-SUN Compatible Radio Board.

Getting Started

Wi-SUN - DDP project

To start using the DDP tool, first install Simplicity Studio v5 and follow this guide to install the SiSDK Package. Once installed, create and build the Wi-SUN DDP application following the next steps:

1. Open Simplicity Studio v5 and plug the Wi-SUN device to your laptop.
2. In the **Debug Adapters** view, select the device that will be provisioned.
3. Navigate to the **EXAMPLE PROJECTS & DEMOS** tab and write **DDP** in the ***Filter on Keywords*** box.
4. Click **Create** next to the ***Wi-SUN - DDP*** project.



5. After creating the project, follow this guide to build your project.

Copyright © 2025 Silicon Laboratories. All rights reserved.

> **Note**: Don't flash the project to the Board. The built binary will be used by the the DDP scripts.

DDP Tool Scripts

The DDP Tool scripts works in combination with the **Wi-SUN - DDP** project. Use the following command to clone the [wisun_applications][wisun application repo] repository:

```
git clone https://github.com/SiliconLabs/wisun_applications.git
```

DDP in Action

Example Scripts: SigningServer.py and provision.py

The repository includes two example scripts: SigningServer.py and provision.py. These scripts serve as practical demonstrations of how to use the Dynamic Data Provisioning (DDP) tool.

- `SigningServer.py` : This script uses OpenSSL to generate a certificate chain, including an intermediate certificate. It showcases the process of creating a secure certificate hierarchy.
- `provision.py` : This script generates a private and public key pair, exports the public key, signs it with the previously generated certificate chain, and then loads the entire chain along with the new certificate. It illustrates the steps involved in key generation and certificate signing.

> Note that these scripts are provided as examples to help you understand the usage of the DDP tool.

Start using the DDP Tool

Now that all elements are prepared, start using the DDP tool to provision the device by following the next steps:

- Use the following commands to install the DDP tool requirements:

  ```
  cd wisun_applications/ddp
  python3 -m pip install -r requirements.txt
  ```

- Next, use the `SigningServer.py` script to generate the certificates chain. Following is an example using Silicon labs OID. Use the command with the parameters suiting your use case:

  ```
  python3 SigningServer.py SetupCerts --co "Silicon Labs" --cu FR --oid 1.3.6.1.4.1.39873.1
  ```

  - **--co:** The name of the OEM, Used for certificate content.
  - **--cu:** The country of the OEM, Used for certificate content.
  - **--oid:** The OID of the product, used for certificate content.
- After generating the certificates, provision the device. In this step, use `provisioning.py` , which takes as arguments the device serial number and the DDP application binary that was built previously:

  ```
  python3 provision.py --soc xg25 --jlink_ser <JLink serial number> --prov_img [path to DDP application binary]
  ```

Conclusion

In conclusion, Dynamic Data Provisioning (DDP) is a valuable tool for the Wi-SUN Networks development, providing a straightforward and secure method for generating and injecting Wi-SUN certificates and device keys. Using the ddp python scripts and the **Wi-SUN - DDP** application, this tool ensures efficient and reliable device provisioning.

Copyright © 2025 Silicon Laboratories. All rights reserved.

API Reference

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.
- Application Settings provides an interface for Application Setting
- 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.
- Trivial File Transfer Protocol (TFTP) Client (Alpha) Trivial File Transfer Protocol (TFTP) Client API
- Application Memory Statistic (Alpha) provides solution for gathering application memory statistic
- Application Status over CoAP (Alpha) provides solution to send application status notifications over CoAP
- Silicon Labs socket API (deprecated) provides a compatibility with Silicon Labs former socket API.
- Keychain provides simplified certificate/private key handling.

Versioning

Copyright © 2025 Silicon Laboratories. All rights reserved.

Silicon Labs Wi-SUN solution follows the Semantic Versioning guidelines for release cycle transparency and to maintain backward compatibility.

Copyright © 2025 Silicon Laboratories. All rights reserved.

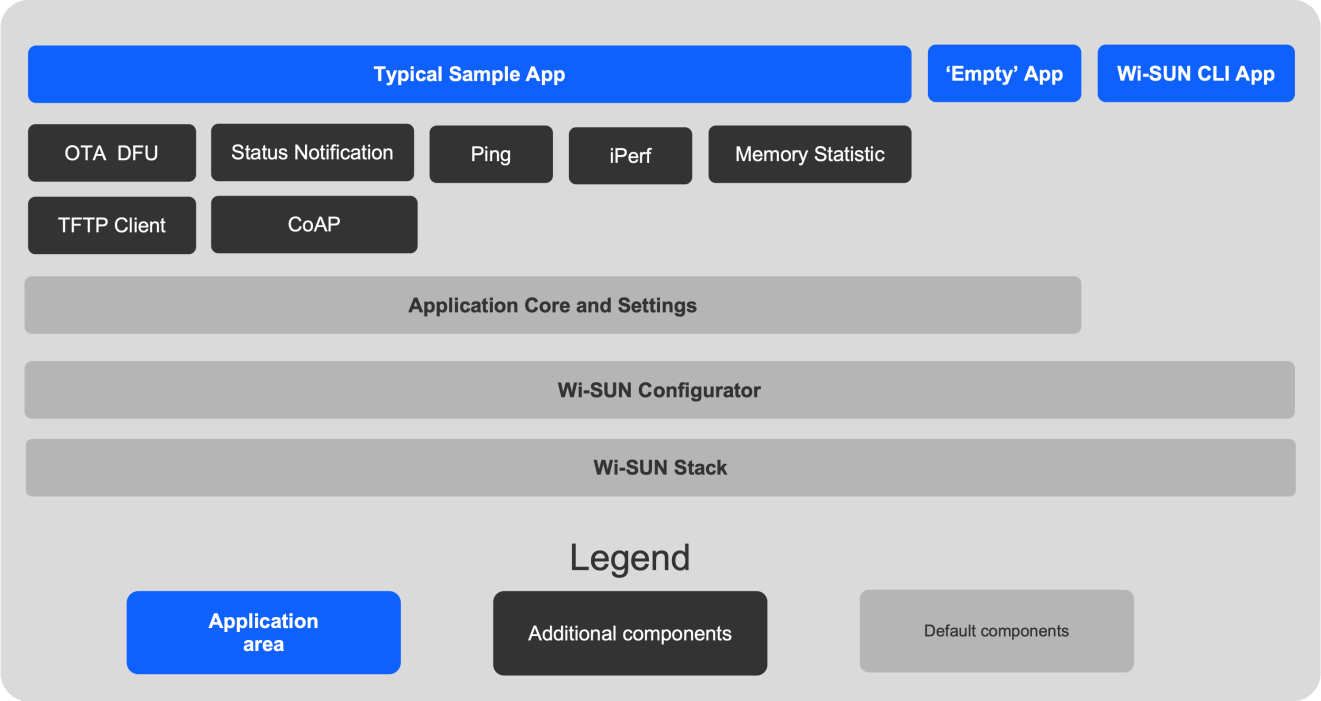
Wi-SUN Services

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
- Application Core
- Application Settings
- CoAP
- Ping
- iPerf
- Over-The-Air Device Firmware Upgrade (Alpha)
- Trivial File Transfer Protocol (TFTP) Client (Alpha)
- Application Memory Statistic (Alpha)

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Application Status over CoAP (Alpha)
- Silicon Labs socket API (deprecated)
- Keychain

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_connect(const uint8_t *network_name) SL_DEPRECATED_API_SDK_2024_6
Connect to a Wi-SUN network.

sl_status_t sl_wisun_util_get_phy_config(sl_wisun_phy_config_t *phy_config) SL_DEPRECATED_API_SDK_2024_6
Get PHY settings of the first RAIL configuration listed in RAIL's channelConfigs array.

Function Documentation

sl_wisun_util_connect

```
sl_status_t sl_wisun_util_connect (const uint8_t * network_name)
```

Connect to a Wi-SUN network.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[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.

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack.

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

Type	Direction	Argument Name	Description
sl_wisun_phy_config_t *	[out]	phy_config	Pointer to PHY configuration

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Application Core

Application Core

The Application Core component is designed to offer utilities commonly used in Wi-SUN applications with the following features:

- **The sl_wisun_app_core_util_connect_and_wait() function** configures the network size setting, the TX output power, the certificates and keys. The function is waiting for the connected state. Its final action is to initiate the Wi-SUN connection process. The sl_wisun_app_core_network_connect() and sl_wisun_app_core_util_network_is_connected() functions also provide an easy way to establish a connection and wait for it to be ready.
- **Event handlers for basic Wi-SUN network events** include the basic event handler implementations.
- **Available addresses**. The sl_wisun_app_core_get_current_addresses() function retrieves known IPv6 addresses.
- **Status Getter**. Use sl_wisun_app_core_get_state() function to get the status of the Stack API calls. The sl_wisun_app_core_get_time_stat() function retrives the connection time, disconnection time stamps and total elapsed time in those states.
- **Limited Function Node (LFN) device support**. Use sl_wisun_app_core_get_device_type() to get the selected device type for the project. The device type can be changed in the Wi-SUN configurator interface. sl_wisun_app_core_get_lfn_profile() and sl_wisun_app_core_get_lfn_params() provide an interface to access the selected LFN profile and its parameters.

The following code snippet shows how the connection can be performed in an application thread:

```
#include "sl_wisun_app_core.h"
#include "sl_wisun_app_core_util.h"

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

  // Connect to the wisun network.
  sl_wisun_app_core_util_connect_and_wait();

  while (1) {

    // Put your application code here.
    osDelay(1);
  }
}
```

Modules

Configurations

Type definitions

void	sl_wisun_app_core_util_project_info_init(const char *app_name) Initialize Wi-SUN project information.
void	sl_wisun_app_core_util_project_info_print(const bool json_format) Print Wi-SUN project information.
const app_project_info_t *	sl_wisun_app_core_util_project_info_get(void) Get Wi-SUN Project info.

Copyright © 2025 Silicon Laboratories. All rights reserved.

void	sl_wisun_app_core_util_wait_for_connection(void) Wait for the connection.
void	sl_wisun_app_core_util_connect_and_wait(void) Connect and wait for connection.
bool	sl_wisun_app_core_util_network_is_connected(void) The network is connected.
void	sl_wisun_app_core_util_dispatch_thread(void) Thread dispatch function.

Functions

void	sl_wisun_app_core_get_current_addresses(sl_wisun_app_core_current_addr_t *const dest_addresses) Get the current addresses.
sl_wisun_device_type_t	sl_wisun_app_core_get_device_type(void) Get Wi-SUN device type.
sl_wisun_join_state_t	sl_wisun_app_core_get_join_state(void) Get Wi-SUN join state.
const sl_wisun_lfn_params_t *	sl_wisun_app_core_get_lfn_params(void) Get LFN parameters.
sl_wisun_lfn_profile_t	sl_wisun_app_core_get_lfn_profile(void) Get Wi-SUN LFN profile.
bool	sl_wisun_app_core_get_regulation_active(void) Return the state of the regional regulation.
sl_status_t	sl_wisun_app_core_get_regulation_thresholds(sl_wisun_app_core_reg_thresholds_t *thresholds_out) Get the warning and alert levels for approaching/exceeded the TX budget.
sl_status_t	sl_wisun_app_core_get_remaining_tx_budget(uint32_t *const budget_out) Get the remaining budget from the transmission quota.
sl_status_t	sl_wisun_app_core_get_state(uint32_t *const state) Get application core state.
void	sl_wisun_app_core_get_time_stat(sl_wisun_app_core_time_stat_t *const tstat) Get time statistic.
void	sl_wisun_app_core_init(void) Initialize Wi-SUN application core.
void	sl_wisun_app_core_network_connect(void) Connect to the Wi-SUN network.
void	sl_wisun_app_core_set_regulation_active(bool enabled) Set the regional regulation to active or passive.
void	sl_wisun_app_core_set_regulation_thresholds(const int8_t warning_level, const int8_t alert_level) Set up warning and alert thresholds for the regional regulation.
sl_status_t	sl_wisun_app_core_wait_state(const uint32_t state, const uint32_t timeout) Waiting for application core state flags.

sl_wisun_app_core_util_project_info_init

Copyright © 2025 Silicon Laboratories. All rights reserved.

> void sl_wisun_app_core_util_project_info_init (const char * app_name)

Initialize Wi-SUN project information.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	app_name	Application name

Init internal instance

sl_wisun_app_core_util_project_info_print

> void sl_wisun_app_core_util_project_info_print (const bool json_format)

Print Wi-SUN project information.

Parameters

Type	Direction	Argument Name	Description
const bool	[in]	json_format	Json format required indicator

Print project info in pretty or json format.

sl_wisun_app_core_util_project_info_get

> const app_project_info_t * sl_wisun_app_core_util_project_info_get (void)

Get Wi-SUN Project info.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Get a constant instance of internal Wi-SUN project info **Returns**

- app_project_info_t * Project info

sl_wisun_app_core_util_wait_for_connection

> void sl_wisun_app_core_util_wait_for_connection (void)

Wait for the connection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function doesn't call the sl_wisun_app_core_network_connect() function. The function provides a delay loop with optional heart beat printing till the connection state has not been changed.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_app_core_util_connect_and_wait

```
void sl_wisun_app_core_util_connect_and_wait (void )
```

Connect and wait for connection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The function calls sl_wisun_app_core_network_connect() function and sl_wisun_app_core_util_wait_for_connection() function. It can be useful at the beginning of application thread.

sl_wisun_app_core_util_network_is_connected

```
bool sl_wisun_app_core_util_network_is_connected (void )
```

The network is connected.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Wrapper function of join state getter **Returns**

- true Connected
- false Not connected

sl_wisun_app_core_util_dispatch_thread

```
void sl_wisun_app_core_util_dispatch_thread (void )
```

Thread dispatch function.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

For low power LFN mode, the delay value is 'SL_WISUN_APP_CORE_THREAD_LP_DISPATCH_MS', for FFN mode, the delay is 1ms

Function Documentation

sl_wisun_app_core_get_current_addresses

```
void sl_wisun_app_core_get_current_addresses (sl_wisun_app_core_current_addr_t *const dest_addresses)
```

Get the current addresses.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_app_core_current_addr_t *const	[out]	dest_addresses	Destination

Copy cached addresses into destination.

sl_wisun_app_core_get_device_type

```
sl_wisun_device_type_t sl_wisun_app_core_get_device_type (void )
```

Get Wi-SUN device type.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Getter to get device type. Device type can be SL_WISUN_LFN or SL_WISUN_ROUTER (FFN) **Returns**

- l_wisun_device_type_t Device type.

sl_wisun_app_core_get_join_state

```
sl_wisun_join_state_t sl_wisun_app_core_get_join_state (void )
```

Get Wi-SUN join state.

Parameters

Type	Direction	Argument Name	Description
void	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.

sl_wisun_app_core_get_lfn_params

```
const sl_wisun_lfn_params_t * sl_wisun_app_core_get_lfn_params (void )
```

Get LFN parameters.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- sl_wisun_lfn_params_t pointer to the set profile parameters of LFN.

sl_wisun_app_core_get_lfn_profile

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_lfn_profile_t sl_wisun_app_core_get_lfn_profile (void)

Get Wi-SUN LFN profile.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Getter to get LFN profile. Device type can be SL_WISUN_LFN_PROFILE_TEST, SL_WISUN_LFN_PROFILE_BALANCED or SL_WISUN_LFN_PROFILE_ECO **Returns**

- sl_wisun_lfn_profile_t LFN profile.

sl_wisun_app_core_get_regulation_active

bool sl_wisun_app_core_get_regulation_active (void)

Return the state of the regional regulation.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function tells the caller if a regulation is currently active. **Returns**

- Boolean indicating if a regional regulation is currently active.

sl_wisun_app_core_get_regulation_thresholds

sl_status_t sl_wisun_app_core_get_regulation_thresholds (sl_wisun_app_core_reg_thresholds_t * thresholds_out)

Get the warning and alert levels for approaching/exceeded the TX budget.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_app_core_reg_thresholds_t *	[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**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_app_core_get_remaining_tx_budget

sl_status_t sl_wisun_app_core_get_remaining_tx_budget (uint32_t *const budget_out)

Get the remaining budget from the transmission quota.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint32_t *const	[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**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_app_core_get_state

```
sl_status_t sl_wisun_app_core_get_state (uint32_t *const state)
```

Get application core state.

Parameters

Type	Direction	Argument Name	Description
uint32_t *const	[out]	state	is the indicator of the error

The function retrieves the application core state value (32 bits) **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_app_core_get_time_stat

```
void sl_wisun_app_core_get_time_stat (sl_wisun_app_core_time_stat_t *const tstat)
```

Get time statistic.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_app_core_time_stat_t *const	[out]	tstat	Time statistic structure

Create a copy of time statistic storage with up-to-date values

sl_wisun_app_core_init

```
void sl_wisun_app_core_init (void )
```

Initialize Wi-SUN application core.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Initializing mutex, socket handler and set Wi-SUN settings.

sl_wisun_app_core_network_connect

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
void sl_wisun_app_core_network_connect (void )
```

Connect to the Wi-SUN network.

Parameters

Type	Direction	Argument Name	Description
void	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.

sl_wisun_app_core_set_regulation_active

```
void sl_wisun_app_core_set_regulation_active (bool enabled)
```

Set the regional regulation to active or passive.

Parameters

Type	Direction	Argument Name	Description
bool	[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).

sl_wisun_app_core_set_regulation_thresholds

```
void sl_wisun_app_core_set_regulation_thresholds (const int8_t warning_level, const int8_t alert_level)
```

Set up warning and alert thresholds for the regional regulation.

Parameters

Type	Direction	Argument Name	Description
const int8_t	[in]	warning_level	new percentage for the warning threshold
const int8_t	[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.

sl_wisun_app_core_wait_state

```
sl_status_t sl_wisun_app_core_wait_state (const uint32_t state, const uint32_t timeout)
```

Waiting for application core state flags.

Parameters

Type	Direction	Argument Name	Description
const uint32_t	[in]	state	state variable which includes the expected flags

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
const uint32_t	[in]	timeout	timeout for waiting

The function waits for application core states described in the state variable. Multiple state flags can be set. The function waits for all flags. Flags are not cleared in this function. **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define SL_WISUN_APP_CORE_DEFAULT_REGULATION_ALERT_THRESHOLD 95U
Transmission duration level alert threshold.

#define SL_WISUN_APP_CORE_DEFAULT_REGULATION_WARNING_THRESHOLD 85U
Transmission duration level warning threshold.

#define SL_WISUN_APP_CORE_MAIN_STACK_SIZE_WORD 256UL
Recommended Application thread stack size.

#define SL_WISUN_APP_CORE_REGULATION SL_WISUN_APP_CORE_REGULATION_NONE
Regulation.

#define SL_WISUN_APP_CORE_REGULATION_ARIB 1U
Regulation ARIB.

#define SL_WISUN_APP_CORE_REGULATION_NONE 0U
Regulation None.

#define SL_WISUN_APP_CORE_REGULATION_WPC 2U
Regulation WPC.

#define SL_WISUN_APP_CORE_THREAD_LP_DISPATCH_MS 1000UL
Thread dispatch delay for low power mode.

#define SL_WISUN_APP_CORE_TX_BUDGET 720000UL
Wi-SUN regulation transmission budget.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Type definitions

Modules

sl_wisun_app_core_current_addr

sl_wisun_app_core_reg_thresholds

sl_wisun_app_core_time_stat

Enumerations

enum sl_wisun_app_core_state {

SL_WISUN_APP_CORE_STATE_NETWORK_CONNECTED = 0
SL_WISUN_APP_CORE_STATE_NETWORK_DISCONNECTED
SL_WISUN_APP_CORE_STATE_NETWORK_CONNECTION_LOST
SL_WISUN_APP_CORE_STATE_SETTING_ERROR
SL_WISUN_APP_CORE_STATE_CONNECTION_ERROR
SL_WISUN_APP_CORE_STATE_SET_NETWORK_SIZE_ERROR
SL_WISUN_APP_CORE_STATE_SET_TX_POWER_ERROR
SL_WISUN_APP_CORE_STATE_SET_DWELL_INTERVAL_ERROR
SL_WISUN_APP_CORE_STATE_SET_MAC_ADDR_ERROR
SL_WISUN_APP_CORE_STATE_SET_ALLOW_MAC_ADDR_ERROR
SL_WISUN_APP_CORE_STATE_SET_DENY_MAC_ADDR_ERROR
SL_WISUN_APP_CORE_STATE_SET_TRUSTED_CERTIFICATE_ERROR
SL_WISUN_APP_CORE_STATE_SET_DEVICE_CERTIFICATE_ERROR
SL_WISUN_APP_CORE_STATE_SET_DEVICE_PRIVATE_KEY_ERROR
SL_WISUN_APP_CORE_STATE_GET_RF_SETTINGS_ERROR

}

Error flag bits enum type definition.

Typedefs

typedef struct sl_wisun_app_core_current_addr — sl_wisun_app_core_current_addr_t
Current address storage structure definition.

typedef struct sl_wisun_app_core_reg_thresholds — sl_wisun_app_core_reg_thresholds_t
Regulation thresholds.

typedef enum sl_wisun_app_core_state — sl_wisun_app_core_state_t
Error flag bits enum type definition.

typedef struct sl_wisun_app_core_time_stat — sl_wisun_app_core_time_stat_t
Application time statistic.

Enumeration Documentation

sl_wisun_app_core_state

> sl_wisun_app_core_state

Copyright © 2025 Silicon Laboratories. All rights reserved.

Error flag bits enum type definition.

Enumerator	
SL_WISUN_APP_CORE_STATE_NETWORK_CONNECTED	Network connected.
SL_WISUN_APP_CORE_STATE_NETWORK_DISCONNECTED	Network disconnected.
SL_WISUN_APP_CORE_STATE_NETWORK_CONNECTION_LOST	Network connection lost.
SL_WISUN_APP_CORE_STATE_SETTING_ERROR	Setting Error Flag bit.
SL_WISUN_APP_CORE_STATE_CONNECTION_ERROR	Connection Failed Error Flag bit.
SL_WISUN_APP_CORE_STATE_SET_NETWORK_SIZE_ERROR	Network Size Error Flag bit.
SL_WISUN_APP_CORE_STATE_SET_TX_POWER_ERROR	TX Power Error Flag bit.
SL_WISUN_APP_CORE_STATE_SET_DWELL_INTERVAL_ERROR	Dwel interval Error flag bit.
SL_WISUN_APP_CORE_STATE_SET_MAC_ADDR_ERROR	Setting MAC address Error Flag bit.
SL_WISUN_APP_CORE_STATE_SET_ALLOW_MAC_ADDR_ERROR	setting Allow mac address Error Flag bit
SL_WISUN_APP_CORE_STATE_SET_DENY_MAC_ADDR_ERROR	setting Deny mac address Error Flag bit
SL_WISUN_APP_CORE_STATE_SET_TRUSTED_CERTIFICATE_ERROR	Trusted Certificate Error Flag bit.
SL_WISUN_APP_CORE_STATE_SET_DEVICE_CERTIFICATE_ERROR	Device Certificate Error Flag bit.
SL_WISUN_APP_CORE_STATE_SET_DEVICE_PRIVATE_KEY_ERROR	Device Private Key Error Flag bit.
SL_WISUN_APP_CORE_STATE_GET_RF_SETTINGS_ERROR	Get RF Setting Error Flag bit.

Typedef Documentation

sl_wisun_app_core_current_addr_t

```
typedef struct sl_wisun_app_core_current_addr sl_wisun_app_core_current_addr_t
```

Current address storage structure definition.

sl_wisun_app_core_reg_thresholds_t

```
typedef struct sl_wisun_app_core_reg_thresholds sl_wisun_app_core_reg_thresholds_t
```

Regulation thresholds.

sl_wisun_app_core_state_t

```
typedef enum sl_wisun_app_core_state sl_wisun_app_core_state_t
```

Error flag bits enum type definition.

sl_wisun_app_core_time_stat_t

```
typedef struct sl_wisun_app_core_time_stat sl_wisun_app_core_time_stat_t
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Application time statistic.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_app_core_current_addr

Current address storage structure definition.

Public Attributes

in6_addr_t	border_router Border Router address.
in6_addr_t	global Global address.
in6_addr_t	link_local Link local address.
in6_addr_t	primary_parent Primary Parent address.
in6_addr_t	secondary_parent Secondary Parent address.

Public Attribute Documentation

border_router

```
in6_addr_t sl_wisun_app_core_current_addr::border_router
```

Border Router address.

global

```
in6_addr_t sl_wisun_app_core_current_addr::global
```

Global address.

link_local

```
in6_addr_t sl_wisun_app_core_current_addr::link_local
```

Link local address.

primary_parent

```
in6_addr_t sl_wisun_app_core_current_addr::primary_parent
```

Primary Parent address.

Copyright © 2025 Silicon Laboratories. All rights reserved.

secondary_parent

```
in6_addr_t sl_wisun_app_core_current_addr::secondary_parent
```

Secondary Parent address.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_app_core_reg_thresholds

Regulation thresholds.

Public Attributes

int8_t	alert_threshold Alert thresholds.
int8_t	warning_threshold Warning thresholds.

Public Attribute Documentation

alert_threshold

```
int8_t sl_wisun_app_core_reg_thresholds::alert_threshold
```

Alert thresholds.

warning_threshold

```
int8_t sl_wisun_app_core_reg_thresholds::warning_threshold
```

Warning thresholds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_app_core_time_stat

Application time statistic.

Public Attributes

uint32_t	conn_cnt Connection counter.
uint64_t	connected_ms Last connected ms.
uint64_t	curr_ms Current ms.
uint64_t	disconnected_ms Last disconnected ms.
uint64_t	tot_connected_ms Total connected ms.
uint64_t	tot_disconnected_ms Total disconnected ms.

Public Attribute Documentation

conn_cnt

```
uint32_t sl_wisun_app_core_time_stat::conn_cnt
```

Connection counter.

connected_ms

```
uint64_t sl_wisun_app_core_time_stat::connected_ms
```

Last connected ms.

curr_ms

```
uint64_t sl_wisun_app_core_time_stat::curr_ms
```

Current ms.

disconnected_ms

```
uint64_t sl_wisun_app_core_time_stat::disconnected_ms
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Last disconnected ms.

tot_connected_ms

```
uint64_t sl_wisun_app_core_time_stat::tot_connected_ms
```

Total connected ms.

tot_disconnected_ms

```
uint64_t sl_wisun_app_core_time_stat::tot_disconnected_ms
```

Total disconnected ms.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Application Settings

Application Settings

The Application Settings component provides the fundamental API for getting and setting Wi-SUN related settings within the application code. After initializing with the app_wisun_setting_init() function, the setters and getters can be used in the application to interface with runtime settings such as network name, network size, TX power, and PHY configuration.

A notification mechanism is implemented and is triggered by the appropriate settings changes.

The following code snippet demonstrates the API usage:

```
#include "sl_wisun_api.h"
#include "sl_wisun_app_setting.h"
#include "sl_wisun_app_core_util.h"

static void change_settings(void) {
  app_setting_wisun_t setting = { 0 };
  sl_status_t status = SL_STATUS_FAIL;

  // Disconnect before setting new values
  sl_wisun_disconnect();

  // Get all the settings
  status = app_wisun_setting_get(&setting);
  if (status != SL_STATUS_OK) {
    // Error handling
    return;
  }

  // Increasing TX power
  setting.tx_power += 10;

  status = app_wisun_setting_set_tx_power(&setting.tx_power);
  if (status != SL_STATUS_OK) {
    // Error handling
    return;
  }

  // Connect with the new settings
  sl_wisun_app_core_util_connect_and_wait();
}
```

Modules

Type definitions

Functions

void app_wisun_setting_ack_notification(const app_setting_notification_t notif, const uint8_t channel)
Acknowledge notification.

sl_status_t app_wisun_setting_get(app_setting_wisun_t *const wisun_setting)
Get the Wi-SUN settings.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_status_t	app_wisun_setting_get_network_name(char *const name, uint8_t size) Get the Wi-SUN network name.
sl_status_t	app_wisun_setting_get_network_size(uint8_t *const size) Get the Wi-SUN network size.
sl_status_t	app_wisun_setting_get_phy(sl_wisun_phy_config_t *const phy) Get the Wi-SUN PHY.
sl_status_t	app_wisun_setting_get_tx_power(int16_t *const tx_power) Get the Wi-SUN TX power.
void	app_wisun_setting_init(void) Initialize Wi-SUN setting.
sl_status_t	app_wisun_setting_init_phy_cfg(void) Init internal PHY settings.
bool	app_wisun_setting_is_notified(const app_setting_notification_t notif, const uint8_t channel) Is setting notified getter.
sl_status_t	app_wisun_setting_set_network_name(const char *const name) Set the Wi-SUN network name.
sl_status_t	app_wisun_setting_set_network_size(const uint8_t *const size) Set the Wi-SUN network size.
sl_status_t	app_wisun_setting_set_phy(const sl_wisun_phy_config_t *const phy) Set the Wi-SUN PHY.
sl_status_t	app_wisun_setting_set_tx_power(const int16_t *const tx_power) Set the Wi-SUN TX power.
sl_status_t	app_wisun_setting_subscribe_notification(const app_setting_notification_t notif, uint8_t *const channel) Subscribe to setting notification channel.
void	app_wisun_setting_unsubscribe(const app_setting_notification_t notif, const uint8_t channel) Unsubscribe from setting notification.

Macros

#define	APP_SETTING_DEFAULT_SUBSCRIPT_CH 0U Wi-SUN settings default subscription channel.
#define	APP_SETTING_NETWORK_NAME_MAX_SIZE (SL_WISUN_NETWORK_NAME_SIZE + 1) Wi-SUN default network name size.

Function Documentation

app_wisun_setting_ack_notification

> void app_wisun_setting_ack_notification (const app_setting_notification_t notif, const uint8_t channel)

Acknowledge notification.

Parameters

Type	Direction	Argument Name	Description
const app_setting_notification_t	[in]	notif	ype of the notification 'app_setting_notification_t'
const uint8_t	[in]	channel	channel number, this is the output of the subscription

Copyright © 2025 Silicon Laboratories. All rights reserved.

Delete notification channel flag

app_wisun_setting_get

```
sl_status_t app_wisun_setting_get (app_setting_wisun_t *const wisun_setting)
```

Get the Wi-SUN settings.

Parameters

Type	Direction	Argument Name	Description
app_setting_wisun_t *const	[out]	wisun_setting	is the obtained Wi-SUN setting

Returns

- sl_status_t if the getting is successful it returns SL_STATUS_OK, otherwise error code.

app_wisun_setting_get_network_name

```
sl_status_t app_wisun_setting_get_network_name (char *const name, uint8_t size)
```

Get the Wi-SUN network name.

Parameters

Type	Direction	Argument Name	Description
char *const	[out]	name	pointer where the name is copied onto
uint8_t	[in]	size	is the size of the name buffer

Returns

- sl_status_t it is successful if it returns SL_STATUS_OK otherwise it is not.

app_wisun_setting_get_network_size

```
sl_status_t app_wisun_setting_get_network_size (uint8_t *const size)
```

Get the Wi-SUN network size.

Parameters

Type	Direction	Argument Name	Description
uint8_t *const	[out]	size	pointer where the network size is copied onto

Returns

- sl_status_t it is successful if it returns SL_STATUS_OK otherwise it is not.

app_wisun_setting_get_phy

```
sl_status_t app_wisun_setting_get_phy (sl_wisun_phy_config_t *const phy)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Get the Wi-SUN PHY.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_phy_config_t *const	[out]	phy	pointer where the PHY is copied onto

Returns

- sl_status_t it is successful if it returns SL_STATUS_OK otherwise it is not.

app_wisun_setting_get_tx_power

```
sl_status_t app_wisun_setting_get_tx_power (int16_t *const tx_power)
```

Get the Wi-SUN TX power.

Parameters

Type	Direction	Argument Name	Description
int16_t *const	[out]	tx_power	pointer where the TX power is copied onto

Returns

- sl_status_t it is successful if it returns SL_STATUS_OK otherwise it is not.

app_wisun_setting_init

```
void app_wisun_setting_init (void )
```

Initialize Wi-SUN setting.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

app_wisun_setting_init_phy_cfg

```
sl_status_t app_wisun_setting_init_phy_cfg (void )
```

Init internal PHY settings.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Full radio config: data initialized with default HPY for appropriate board Simple PHY: get first element of PHY list by 'sl_wisun_util_get_rf_settings' **Returns**

- sl_status_t it is successful if it returns SL_STATUS_OK otherwise it is not.

Copyright © 2025 Silicon Laboratories. All rights reserved.

app_wisun_setting_is_notified

bool app_wisun_setting_is_notified (const app_setting_notification_t notif, const uint8_t channel)

Is setting notified getter.

Parameters

Type	Direction	Argument Name	Description
const app_setting_notification_t	[in]	notif	Type of the notification 'app_setting_notification_t'
const uint8_t	[in]	channel	channel number, this is the output of the subscription

Polling notification flag **Returns**

- bool true if channel is notified by setting, otherwise false

app_wisun_setting_set_network_name

sl_status_t app_wisun_setting_set_network_name (const char *const name)

Set the Wi-SUN network name.

Parameters

Type	Direction	Argument Name	Description
const char *const	[out]	name	is the network name that will be set.

Returns

- sl_status_t if the set is successful it returns SL_STATUS_OK, otherwise error code.

app_wisun_setting_set_network_size

sl_status_t app_wisun_setting_set_network_size (const uint8_t *const size)

Set the Wi-SUN network size.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *const	[in]	size	is the pointer to network size that will be set.

Returns

- sl_status_t if the set is successful it returns SL_STATUS_OK, otherwise error code.

app_wisun_setting_set_phy

sl_status_t app_wisun_setting_set_phy (const sl_wisun_phy_config_t *const phy)

Set the Wi-SUN PHY.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_phy_config_t *const	[out]	phy	is the PHY that will be set.

Returns

- sl_status_t if the set is successful it returns SL_STATUS_OK, otherwise error code.

app_wisun_setting_set_tx_power

> sl_status_t app_wisun_setting_set_tx_power (const int16_t *const tx_power)

Set the Wi-SUN TX power.

Parameters

Type	Direction	Argument Name	Description
const int16_t *const	[out]	tx_power	is the pointer to TX power that will be set.

Returns

- sl_status_t if the set is successful it returns SL_STATUS_OK, otherwise error code.

app_wisun_setting_subscribe_notification

> sl_status_t app_wisun_setting_subscribe_notification (const app_setting_notification_t notif, uint8_t *const channel)

Subscribe to setting notification channel.

Parameters

Type	Direction	Argument Name	Description
const app_setting_notification_t	[in]	notif	Type of the notification 'app_setting_notification_t'
uint8_t *const	[out]	channel	Provided channel number, this is the output of the subscription

Notifications are received on the channel (output) about notification type **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

app_wisun_setting_unsubscribe

> void app_wisun_setting_unsubscribe (const app_setting_notification_t notif, const uint8_t channel)

Unsubscribe from setting notification.

Parameters

Type	Direction	Argument Name	Description
const app_setting_notification_t	[in]	notif	Type of the notification 'app_setting_notification_t'

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
const uint8_t	[in]	channel	channel number, this is the output of the subscription

Delete notification channel flag from subscripted channel flags

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Type definitions

Modules

app_setting_wisun

Enumerations

enum app_setting_notification {

APP_SETTING_NOTIFICATION_SET_NETWORK_NAME = 0UL
APP_SETTING_NOTIFICATION_SET_NETWORK_SIZE
APP_SETTING_NOTIFICATION_SET_TX_POWER
APP_SETTING_NOTIFICATION_SET_PHY_CFG

}

App settings notification channels.

Typedefs

typedef enum app_setting_notification — app_setting_notification_t
App settings notification channels.

typedef struct app_setting_wisun — app_setting_wisun_t
Wisun setting structure.

Enumeration Documentation

app_setting_notification

```
app_setting_notification
```

App settings notification channels.

Enumerator	
APP_SETTING_NOTIFICATION_SET_NETWORK_NAME	Set Network Name notification.
APP_SETTING_NOTIFICATION_SET_NETWORK_SIZE	Set Network Size notification.
APP_SETTING_NOTIFICATION_SET_TX_POWER	Set TX Power notification.
APP_SETTING_NOTIFICATION_SET_PHY_CFG	Set TX Power notification.

Typedef Documentation

app_setting_notification_t

```
typedef enum app_setting_notification app_setting_notification_t
```

App settings notification channels.

Copyright © 2025 Silicon Laboratories. All rights reserved.

app_setting_wisun_t

```
typedef struct app_setting_wisun app_setting_wisun_t
```

Wisun setting structure.

Copyright © 2025 Silicon Laboratories. All rights reserved.

app_setting_wisun

Wisun setting structure.

Public Attributes

uint8_t	device_type Device type.
bool	is_default_phy default flag
uint8_t	keychain Key chain.
uint8_t	keychain_index Key chain index.
uint8_t	lfn_profile LFN profile.
char	network_name Network Name.
uint8_t	network_size Network size.
sl_wisun_phy_config_t	phy PHY settings.
int16_t	tx_power_ddbm TX Power.

Public Attribute Documentation

device_type

```
uint8_t app_setting_wisun::device_type
```

Device type.

is_default_phy

```
bool app_setting_wisun::is_default_phy
```

default flag

keychain

```
uint8_t app_setting_wisun::keychain
```

Key chain.

Copyright © 2025 Silicon Laboratories. All rights reserved.

keychain_index

```
uint8_t app_setting_wisun::keychain_index
```

Key chain index.

lfn_profile

```
uint8_t app_setting_wisun::lfn_profile
```

LFN profile.

network_name

```
char app_setting_wisun::network_name[APP_SETTING_NETWORK_NAME_MAX_SIZE]
```

Network Name.

network_size

```
uint8_t app_setting_wisun::network_size
```

Network size.

phy

```
sl_wisun_phy_config_t app_setting_wisun::phy
```

PHY settings.

tx_power_ddbm

```
int16_t app_setting_wisun::tx_power_ddbm
```

TX Power.

Copyright © 2025 Silicon Laboratories. All rights reserved.

CoAP

CoAP

The Wi-SUN CoAP component offers an application layer implementation built on top of the Wi-SUN stack and its socket API. The CoAP protocol is specifically designed as an Internet Application Protocol for constrained devices operating in lossy networks (as defined in RFC 7252). Wi-SUN FAN networks precisely fit this definition.

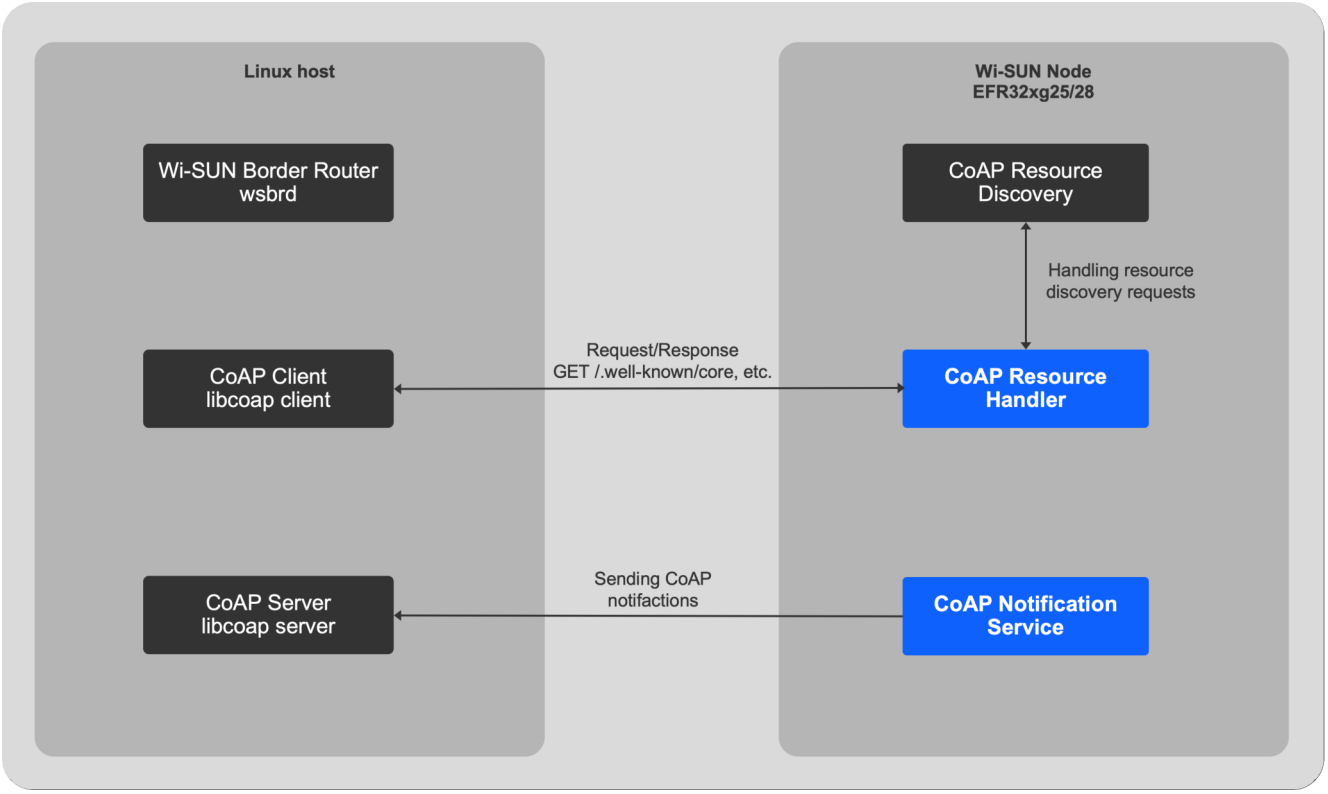
The CoAP implementation relies on the UDP transport layer for sending and receiving packets. Within the component, you'll find a set of helper functions that assist in building CoAP payloads, parsing CoAP packets, and freeing CoAP packets. Additionally, the CoAP Resource Handler service manages registered resources.

For resource discovery, the Resource Discovery request provides an interface to retrieve information about available resources. These resources have additional attributes beyond the URI (Uniform Resource Identifier) path, which can be selectively filtered using specific Resource Discovery requests.

Furthermore, the CoAP Notification service allows the creation and schedule of notifications for the remote host. The schedule time and send conditions can be customized based on the application requirements. To incorporate the CoAP component into the application, simply add it to the project and initialize it using sl_wisun_coap_init().

The CoAP component is compatible with third-party implementations, such as the libcoap server and client binaries.

The following block diagram shows the architecture of CoAP component:



The following example shows the CoAP resource and notification initialization:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```c
#include <string.h>
#include "sl_wisun_coap.h"
#include "sl_wisun_coap_rhnd.h"
#include "sl_wisun_coap_notify.h"

#define NOTIFY_URI_PATH "notify/temperature"

// CoAP Resource Handler callback
static sl_wisun_coap_packet_t *
callback_to_handle_requests(const sl_wisun_coap_packet_t * const req_packet) {
  sl_wisun_coap_packet_t *resp_packet = NULL;

  // Build pre-initialized CoAP packet with content response message code
  resp_packet = sl_wisun_coap_build_response(req_packet, COAP_MSG_CODE_RESPONSE_CONTENT);
  if (resp_packet == NULL) {
    return NULL;
  }

  // Build your CoAP payload (resp_packet->payload_ptr)
  // and set the payload length (resp_packet->payload_len)

  return resp_packet;
}

static bool condition_callback(const sl_wisun_coap_notify_t * notify) {
  // Build your notification condition and return the check result
  (void) notify;
  return true;
}

static sl_wisun_coap_packet_t *
callback_to_build_packet(const sl_wisun_coap_notify_t *notify) {
  // Initialize a CoAP Packet
  static sl_wisun_coap_packet_t pkt = {
    .msg_code = COAP_MSG_CODE_REQUEST_PUT, // Message Code: PUT request
    .msg_id = 1234, // Message ID
    .msg_type = COAP_MSG_TYPE_NON_CONFIRMABLE, // Message type
    .content_format = COAP_CT_TEXT_PLAIN, // Content format
    .uri_path_ptr = NULL, // URI Path pointer
    .uri_path_len = 0, // URI Path length
    .payload_ptr = NULL, // Payload pointer
    .payload_len = 0, // Payload length
    .token_ptr = NULL, // Token pointer
    .token_len = 0, // Token length
    .options_list_ptr = NULL // Option list pointer
  };

  (void) notify;

  // Set the URI path
  pkt.uri_path_ptr = (uint8_t *)NOTIFY_URI_PATH;
  pkt.uri_path_len = strlen(NOTIFY_URI_PATH);

  // Set your notification payload content
  // pkt.payload_ptr = your_prebuilt_buf;
  // pkt.payload_len = your_payload_length;

  // Return with the notification packet
  return &pkt;
}


static void init_resource_and_notification(void) {
  // Return value
  int32_t ret = 0;

  // Resource descriptor allocation
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
 sl_wisun_coap_rhnd_resource_t coap_resource ={0};// Notification handler allocation
 sl_wisun_coap_notify_t notify ={0};// Prepare resource with default contentsl_wisun_coap_rhnd_resource_init(&coap_resource);// Resource URI path
 coap_resource.data.uri_path ="/sensor/temperature";// Resource type
 coap_resource.data.resource_type ="temp";// Interface
 coap_resource.data.interface ="i2c-sensor";// Function pointer to handle requests, like temperature measurement and response packet build.// Pointer type is sl_wisun_coap_packet_t * (*)(const sl_wisun_coap_packet_t * const req_packet)
 coap_resource.auto_response = callback_to_handle_requests;// You can initialize an extended callback instead of the standard callback type.// The extended callback receives the source address of the requests.// The function pointer type is// sl_wisun_coap_packet_t * (*)(const sockaddr_in6_t * const src_addr, const sl_wisun_coap_packet_t * const req_packet)// coap_resource.auto_response_ext = extended_callback_to_handle_requests;// Redirect response// The callback performs a response redirection. You can build your new destination address// and retrieve the new_addr argument// Pointer type is// void (*)(sockaddr_in6_t * const new_addr,const sl_wisun_coap_packet_t * const req_packet)// coap_resource.redirect_response = NULL;// Resource is discoverable. You can get information with resource discovery request// GET /.well-known/core
 coap_resource.discoverable =true;// Add resource to the resource listassert(sl_wisun_coap_rhnd_resource_add(&coap_resource)==SL_STATUS_OK);// Set the notification ID
 notify.id ="my-notification";// Set the destination address of notification
 ret =inet_pton(AF_INET6,"2001:db8::1",&notify.remote_addr.sin6_addr);assert(ret ==1);
 notify.remote_addr.sin6_port =htons(5683);// Set the schedule time of the notification, 1s is in the example
 notify.schedule_time_ms =1000;// Initialize the condition callback.// The callback checks the condition whether the created CoAP packet should be sent// The function pointer type is bool (*)(const sl_wisun_coap_notify_t * notify)
 notify.condition_cb = condition_callback;// Initialize the notification handler that builds the CoAP packet// The function pointer type is// sl_wisun_coap_packet_t * (*)(const sl_wisun_coap_notify_t * notify)
 notify.hnd_cb = callback_to_build_packet;// Add the notification to the notification listassert(sl_wisun_coap_notify_add(&notify)==SL_STATUS_OK);}
```

Modules

- Configurations
- Type definitions
- CoAP Resource Handler API
- CoAP Notification API

Functions

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.
int16_t	sl_wisun_coap_builder(uint8_t *dest_buff, const sl_wisun_coap_packet_t *message) CoAP message builder Wi-SUN.
uint16_t	sl_wisun_coap_builder_calc_size(const sl_wisun_coap_packet_t *message) CoAP packet calc size wrapper.
void	sl_wisun_coap_destroy_packet(sl_wisun_coap_packet_t *packet) Destroy packet.
__STATIC_INLINE void	sl_wisun_coap_destroy_payload_str(char *str) Destroy payload string.
__STATIC_INLINE void	sl_wisun_coap_destroy_uri_path_str(const char *uri_str) Destroy URI path string.
void	sl_wisun_coap_free(void *addr) Free Wi-SUN CoAP.

Copyright © 2025 Silicon Laboratories. All rights reserved.

const sl_wisun_coap_handle_t *	sl_wisun_coap_get_lib_handler(void) Get the library handler pointer from the internal handler structure.
char *	sl_wisun_coap_get_payload_str(const sl_wisun_coap_packet_t *const packet) Prepare payload string.
char *	sl_wisun_coap_get_uri_path_str(const sl_wisun_coap_packet_t *const packet) Prepare URI path string.
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.
sl_wisun_coap_packet_t *	sl_wisun_coap_parser(uint16_t packet_data_len, uint8_t *packet_data) CoAP parser 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.

Macros

#define	SL_COAP_SERVICE_LOOP () CoAP Service loop.

Function Documentation

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

Type	Direction	Argument Name	Description
const sl_wisun_coap_packet_t *	[in]	req	request
sl_wisun_coap_message_code_t	[in]	msg_code	message code to build

Used sn_coap_build_response. **Returns**

- sl_wisun_coap_header_t* built packet ptr on the heap

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint8_t *	[out]	dest_buff	destination buffer for raw message
const sl_wisun_coap_packet_t *	[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

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

Type	Direction	Argument Name	Description
const sl_wisun_coap_packet_t *	[in]	message	message ptr

Used sn_coap_builder_calc_needed_packet_data_size. **Returns**

- uint16_t size

sl_wisun_coap_destroy_packet

> void sl_wisun_coap_destroy_packet (sl_wisun_coap_packet_t * packet)

Destroy packet.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_coap_packet_t *	N/A	packet	packet

It must be used to avoid memory leaks! Free the all of allocated buffer for packet

sl_wisun_coap_destroy_payload_str

> __STATIC_INLINE void sl_wisun_coap_destroy_payload_str (char * str)

Destroy payload string.

Parameters

Type	Direction	Argument Name	Description
char *	[in]	str	String

'\0' terminated string in the heap, it must be freed

sl_wisun_coap_destroy_uri_path_str

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
__STATIC_INLINE void sl_wisun_coap_destroy_uri_path_str (const char * uri_str)
```

Destroy URI path string.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	uri_str	URI string ptr

Call free on allocated pointer

sl_wisun_coap_free

```
void sl_wisun_coap_free (void * addr)
```

Free Wi-SUN CoAP.

Parameters

Type	Direction	Argument Name	Description
void *	N/A	addr	address ptr

OS-dependent thread-safe implementation.

Referenced by sl_wisun_coap_destroy_payload_str , and sl_wisun_coap_destroy_uri_path_str

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

Type	Direction	Argument Name	Description
void	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

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

Type	Direction	Argument Name	Description
const sl_wisun_coap_packet_t *const	[in]	packet	Packet

'\0' terminated string in the heap, it must be freed **Returns**

Copyright © 2025 Silicon Laboratories. All rights reserved.

- char* payload string, NULL on error

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

Type	Direction	Argument Name	Description
const sl_wisun_coap_packet_t *const	[in]	packet	Packet

'\0' terminated string in the heap, it must be freed **Returns**

- char* URI path string, NULL on error

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

Type	Direction	Argument Name	Description
const sl_wisun_coap_tx_callback	[in]	tx_callback	TX callback, if it's NULL, the default callback is applied
const sl_wisun_coap_rx_callback	[in]	rx_callback	RX callback, if it's NULL, the default callback is applied
const sl_wisun_coap_version_t	[in]	version	CoAP version

Set the Wi-SUN CoAP internal descriptor.

sl_wisun_coap_init_default

__STATIC_INLINE void sl_wisun_coap_init_default (void)

Initialize Wi-SUN CoAP default.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Initializes the internal descriptor with default values.

sl_wisun_coap_malloc

void * sl_wisun_coap_malloc (uint16_t size)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Implement malloc.

Parameters

Type	Direction	Argument Name	Description
uint16_t	N/A	size	size for malloc

OS-dependent thread-safe implementation. **Returns**

- void* the memory pointer

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

Type	Direction	Argument Name	Description
uint16_t	[in]	packet_data_len	packet data size
uint8_t *	[in]	packet_data	packet data ptr

Used sn_coap_parser **Returns**

- sl_wisun_coap_packet_t* Parsed packet pointer

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

Type	Direction	Argument Name	Description
const sl_wisun_coap_packet_t *	[in]	packet	packet to print
const bool	[in]	hex_format	hex format bool to decide buffer print format

Pretty printer function, with hex format option for buffers

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define SL_WISUN_COAP_EASY_CLNT_SRV_INSTANCE_ENABLE 0U
Enable easy CoAP client and CoAP server instances.

#define SL_WISUN_COAP_MEM_USE_STATIC_MEMORY_POOL 0U
Enable Static Memory Pool usage.

#define SL_WISUN_COAP_NOTIFY_DEFAULT_REMOTE_ADDR "2001:db8::1"
Default remote address to notify.

#define SL_WISUN_COAP_NOTIFY_DEFAULT_REMOTE_PORT 5683U
Default port of the remote host to notify.

#define SL_WISUN_COAP_NOTIFY_MAX_NOTIFICATION 10U
Maximum capacity of the CoAP Notification Table.

#define SL_WISUN_COAP_NOTIFY_SERVICE_ENABLE 1U
Enable Notification service.

#define SL_WISUN_COAP_NOTIFY_SOCK_BUFF_SIZE 1024UL
Notification service socket communication buffer size.

#define SL_WISUN_COAP_NOTIFY_STACK_SIZE_WORD 256UL
Notification service stack size in CPU word.

#define SL_WISUN_COAP_RESOURCE_HND_MAX_RESOURCES 10U
Maximum capacity of the CoAP Resource Table.

#define SL_WISUN_COAP_RESOURCE_HND_SERVICE_ENABLE 1U
Enable Resource Handler service.

#define SL_WISUN_COAP_RESOURCE_HND_SERVICE_PORT 5683U
Resource handler service listener port.

#define SL_WISUN_COAP_RESOURCE_HND_SOCK_BUFF_SIZE 1024UL
Resource handler service socket communication buffer size.

#define SL_WISUN_COAP_RESOURCE_HND_STACK_SIZE_WORD 256UL
Resource handler service stack size in CPU word.

#define SL_WISUN_COAP_RESOURCE_HND_VERBOSE_MODE_ENABLE 0U
Enable Resource Handler verbose mode.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Type definitions

Modules

sl_wisun_coap

Typedefs

typedef void(*	sl_wisun_coap_free_t)(void *mem) Wi-SUN CoAP free function pointer typedef.
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 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_options_list_s	sl_wisun_coap_option_list_t Wi-SUN CoAP option list typedef.
typedef sn_coap_option_numbers_e	sl_wisun_coap_option_num_t Wi-SUN CoAP option number typedef.
typedef sn_coap_hdr_s	sl_wisun_coap_packet_t Wi-SUN CoAP message 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 struct sl_wisun_coap	sl_wisun_coap_t Wi-SUN CoAP descriptor structure.
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 coap_version_e	sl_wisun_coap_version_t Wi-SUN CoAP version typedef.

Macros

#define	SL_WISUN_COAP_URI_PATH_MAX_SIZE (128U) Maximum size of the URI path string.

Typedef Documentation

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_coap_free_t

```
typedef void(* sl_wisun_coap_free_t) (void *mem) )(void *mem)
```

Wi-SUN CoAP free function pointer typedef.

sl_wisun_coap_handle_t

```
typedef struct coap_s sl_wisun_coap_handle_t
```

Wi-SUN CoAP handler for Wi-SUN component.

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.

sl_wisun_coap_message_code_t

```
typedef sn_coap_msg_code_e sl_wisun_coap_message_code_t
```

Wi-SUN CoAP message code typedef.

sl_wisun_coap_message_type_t

```
typedef sn_coap_msg_type_e sl_wisun_coap_message_type_t
```

Wi-SUN CoAP message type typedef.

sl_wisun_coap_option_list_t

```
typedef sn_coap_options_list_s sl_wisun_coap_option_list_t
```

Wi-SUN CoAP option list typedef.

sl_wisun_coap_option_num_t

```
typedef sn_coap_option_numbers_e sl_wisun_coap_option_num_t
```

Wi-SUN CoAP option number typedef.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_coap_packet_t

```
typedef sn_coap_hdr_s sl_wisun_coap_packet_t
```

Wi-SUN CoAP message typedef.

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.

sl_wisun_coap_t

```
typedef struct sl_wisun_coap sl_wisun_coap_t
```

Wi-SUN CoAP descriptor structure.

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.

sl_wisun_coap_version_t

```
typedef coap_version_e sl_wisun_coap_version_t
```

Wi-SUN CoAP version typedef.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_coap

Wi-SUN CoAP descriptor structure.

Public Attributes

sl_wisun_coap_free_t	free free function
sl_wisun_coap_handle_t *	handler lib handler
sl_wisun_coap_malloc_t	malloc malloc function
sl_wisun_coap_rx_callback	rx_callback RX callback.
sl_wisun_coap_tx_callback	tx_callback TX callback.
sl_wisun_coap_version_t	version CoAP version.

Public Attribute Documentation

free

```
sl_wisun_coap_free_t sl_wisun_coap::free
```

free function

handler

```
sl_wisun_coap_handle_t* sl_wisun_coap::handler
```

lib handler

malloc

```
sl_wisun_coap_malloc_t sl_wisun_coap::malloc
```

malloc function

rx_callback

```
sl_wisun_coap_rx_callback sl_wisun_coap::rx_callback
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

RX callback.

tx_callback

```
sl_wisun_coap_tx_callback sl_wisun_coap::tx_callback
```

TX callback.

version

```
sl_wisun_coap_version_t sl_wisun_coap::version
```

CoAP version.

Copyright © 2025 Silicon Laboratories. All rights reserved.

CoAP Resource Handler API

CoAP Resource Handler API

Modules

Type definitions

Functions

const sl_wisun_coap_rhnd_resource_t *	sl_wisun_coap_rhnd_get_resources(void) Get resources.
void	sl_wisun_coap_rhnd_init(void) Initialization of the CoAP Resource Discovery internals.
bool	sl_wisun_coap_rhnd_is_request_packet(const sl_wisun_coap_packet_t *const packet) Is request packet?
void	sl_wisun_coap_rhnd_print_resources(void) Print resources.
sl_status_t	sl_wisun_coap_rhnd_reset_auto_response(const char *uri_path) Reset auto-response callback.
sl_status_t	sl_wisun_coap_rhnd_resource_add(const sl_wisun_coap_rhnd_resource_t *const src_resource) Register a new resource into the CoAP Resource table.
__STATIC_INLINE void	sl_wisun_coap_rhnd_resource_init(sl_wisun_coap_rhnd_resource_t *const src_resource) Initialise empty resource.
sl_status_t	sl_wisun_coap_rhnd_resource_remove_by_uri(const char *uri_path) Remove a resource from the CoAP Resource table referenced by name.
sl_status_t	sl_wisun_coap_rhnd_set_auto_response(const char *uri_path, sl_wisun_coap_rhnd_auto_resp_t response) Set auto-response callback.

Function Documentation

sl_wisun_coap_rhnd_get_resources

```
const sl_wisun_coap_rhnd_resource_t * sl_wisun_coap_rhnd_get_resources (void )
```

Get resources.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Getting linked list of resources **Returns**

- const sl_wisun_coap_rhnd_resource_t* Resource list, on error is NULL ptr

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_coap_rhnd_init

```
void sl_wisun_coap_rhnd_init (void )
```

Initialization of the CoAP Resource Discovery internals.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_wisun_coap_rhnd_is_request_packet

```
bool sl_wisun_coap_rhnd_is_request_packet (const sl_wisun_coap_packet_t *const packet)
```

Is request packet?

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_coap_packet_t *const	[in]	packet	Packet

helper function **Returns**

- true Request packet
- false Response or Empty packet

sl_wisun_coap_rhnd_print_resources

```
void sl_wisun_coap_rhnd_print_resources (void )
```

Print resources.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Print resource URI paths

sl_wisun_coap_rhnd_reset_auto_response

```
sl_status_t sl_wisun_coap_rhnd_reset_auto_response (const char * uri_path)
```

Reset auto-response callback.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	uri_path	URI path

Copyright © 2025 Silicon Laboratories. All rights reserved.

Set callback ptr to NULL in resource descriptor **Returns**

- sl_status_t SL_STATUS_OK on success, SL_STATUS_FAIL on error

sl_wisun_coap_rhnd_resource_add

```
sl_status_t sl_wisun_coap_rhnd_resource_add (const sl_wisun_coap_rhnd_resource_t *const src_resource)
```

Register a new resource into the CoAP Resource table.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_coap_rhnd_resource_t *const	[in]	src_resource	the new resource's descriptor

Returns

- True if the registration is successful

sl_wisun_coap_rhnd_resource_init

```
__STATIC_INLINE void sl_wisun_coap_rhnd_resource_init (sl_wisun_coap_rhnd_resource_t *const src_resource)
```

Initialise empty resource.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_coap_rhnd_resource_t *const	[in]	src_resource	Resource descriptor

Set pointers to NULL and flags to false

References sl_wisun_coap_rhnd_resource::auto_response , sl_wisun_coap_rhnd_resource::auto_response_ext , sl_wisun_coap_rhnd_resource::data , sl_wisun_coap_rhnd_resource::discoverable , sl_wisun_coap_rhnd_resource_data::interface , sl_wisun_coap_rhnd_resource::next , sl_wisun_coap_rhnd_resource::redirect_response , sl_wisun_coap_rhnd_resource_data::resource_type , and sl_wisun_coap_rhnd_resource_data::uri_path

sl_wisun_coap_rhnd_resource_remove_by_uri

```
sl_status_t sl_wisun_coap_rhnd_resource_remove_by_uri (const char * uri_path)
```

Remove a resource from the CoAP Resource table referenced by name.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	uri_path	URI path of the resource to be removed

Returns

- True if removal is successful

sl_wisun_coap_rhnd_set_auto_response

Copyright © 2025 Silicon Laboratories. All rights reserved.

> sl_status_t sl_wisun_coap_rhnd_set_auto_response (const char * uri_path, sl_wisun_coap_rhnd_auto_resp_t response)

Set auto-response callback.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	uri_path	URI path
sl_wisun_coap_rhnd_auto_resp_t	[in]	response	Response callback

Set callback for resource **Returns**

- sl_status_t SL_STATUS_OK on success, SL_STATUS_FAIL on error

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Type definitions

Modules

sl_wisun_coap_rhnd_resource

sl_wisun_coap_rhnd_resource_data

Typedefs

typedef sl_wisun_coap_packet_t *(* — sl_wisun_coap_rhnd_auto_resp_ext_t)(const sockaddr_in6_t *const src_addr, const sl_wisun_coap_packet_t *const req_packet)
Extended auto response callback for resources.

typedef sl_wisun_coap_packet_t *(* — sl_wisun_coap_rhnd_auto_resp_t)(const sl_wisun_coap_packet_t *const req_packet)
Auto-Response callback for resources.

typedef void(* — sl_wisun_coap_rhnd_redirect_resp_t)(sockaddr_in6_t *const new_addr, const sl_wisun_coap_packet_t *const req_packet)
Redirect response callback type to create new destination socket address.

typedef struct sl_wisun_coap_rhnd_resource_data — sl_wisun_coap_rhnd_resource_data_t
Resource table data.

typedef struct sl_wisun_coap_rhnd_resource — sl_wisun_coap_rhnd_resource_t
Resource table entry.

Typedef Documentation

sl_wisun_coap_rhnd_auto_resp_ext_t

```
typedef sl_wisun_coap_packet_t *(* sl_wisun_coap_rhnd_auto_resp_ext_t) (const sockaddr_in6_t *const src_addr, const sl_wisun_coap_packet_t *const req_packet) )(const sockaddr_in6_t *const src_addr, const sl_wisun_coap_packet_t *const req_packet)
```

Extended auto response callback for resources.

sl_wisun_coap_rhnd_auto_resp_t

```
typedef sl_wisun_coap_packet_t *(* sl_wisun_coap_rhnd_auto_resp_t) (const sl_wisun_coap_packet_t *const req_packet) ) (const sl_wisun_coap_packet_t *const req_packet)
```

Auto-Response callback for resources.

sl_wisun_coap_rhnd_redirect_resp_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef void(* sl_wisun_coap_rhnd_redirect_resp_t) (sockaddr_in6_t *const new_addr, const sl_wisun_coap_packet_t *const req_packet) )(sockaddr_in6_t *const new_addr, const sl_wisun_coap_packet_t *const req_packet)
```

Redirect response callback type to create new destination socket address.

sl_wisun_coap_rhnd_resource_data_t

```
typedef struct sl_wisun_coap_rhnd_resource_data sl_wisun_coap_rhnd_resource_data_t
```

Resource table data.

sl_wisun_coap_rhnd_resource_t

```
typedef struct sl_wisun_coap_rhnd_resource sl_wisun_coap_rhnd_resource_t
```

Resource table entry.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_coap_rhnd_resource

Resource table entry.

Public Attributes

sl_wisun_coap_rhnd_auto_resp_t	auto_response Auto response.
sl_wisun_coap_rhnd_auto_resp_ext_t	auto_response_ext Auto response extended If it's registered in the resource descriptor, it will be called instead of auto_response.
sl_wisun_coap_rhnd_resource_data_t	data Resource data.
bool	discoverable Resource discoverable.
struct sl_wisun_coap_rhnd_resource *	next Next pointer.
sl_wisun_coap_rhnd_redirect_resp_t	redirect_response Redirect response callback.

Public Attribute Documentation

auto_response

```
sl_wisun_coap_rhnd_auto_resp_t sl_wisun_coap_rhnd_resource::auto_response
```

Auto response.

auto_response_ext

```
sl_wisun_coap_rhnd_auto_resp_ext_t sl_wisun_coap_rhnd_resource::auto_response_ext
```

Auto response extended If it's registered in the resource descriptor, it will be called instead of auto_response.

data

```
sl_wisun_coap_rhnd_resource_data_t sl_wisun_coap_rhnd_resource::data
```

Resource data.

discoverable

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
bool sl_wisun_coap_rhnd_resource::discoverable
```

Resource discoverable.

next

```
struct sl_wisun_coap_rhnd_resource* sl_wisun_coap_rhnd_resource::next
```

Next pointer.

redirect_response

```
sl_wisun_coap_rhnd_redirect_resp_t sl_wisun_coap_rhnd_resource::redirect_response
```

Redirect response callback.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_coap_rhnd_resource_data

Resource table data.

Public Attributes

const char *	interface inteface type string
const char *	resource_type resource type string
const char *	uri_path resource URI path string

Public Attribute Documentation

interface

```
const char* sl_wisun_coap_rhnd_resource_data::interface
```

inteface type string

resource_type

```
const char* sl_wisun_coap_rhnd_resource_data::resource_type
```

resource type string

uri_path

```
const char* sl_wisun_coap_rhnd_resource_data::uri_path
```

resource URI path string

Copyright © 2025 Silicon Laboratories. All rights reserved.

CoAP Notification API

CoAP Notification API

Modules

Type definitions

Functions

sl_status_t	sl_wisun_coap_notify_add(const sl_wisun_coap_notify_t *const src_notify) Register a new notification.
const sl_wisun_coap_notify_t *	sl_wisun_coap_notify_get_by_id(const char *id) Get notification by ID.
void	sl_wisun_coap_notify_init(void) Initialization of the CoAP notification service.
sl_status_t	sl_wisun_coap_notify_init_default_inst(sl_wisun_coap_notify_t *const notify, const char *id) Init default notification instance.
void	sl_wisun_coap_notify_print_notifications(void) Print notifications.
sl_status_t	sl_wisun_coap_notify_remove_by_id(const char *id) Remove notification from the CoAP notification table referenced by name.
sl_status_t	sl_wisun_coap_notify_send_notification(const sl_wisun_coap_notify_t *const notify) Send notification.

Function Documentation

sl_wisun_coap_notify_add

> sl_status_t sl_wisun_coap_notify_add (const sl_wisun_coap_notify_t *const src_notify)

Register a new notification.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_coap_notify_t *const	[in]	src_notify	the new notification's descriptor to add

Notification is added to the internal storage. **Returns**

- sl_status_t SL_STATUS_OK if the notification is added successfully, otherwise SL_STATUS_FAIL

sl_wisun_coap_notify_get_by_id

> const sl_wisun_coap_notify_t * sl_wisun_coap_notify_get_by_id (const char * id)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Get notification by ID.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	id	unique ID of the notification (custom string)

Get pointer to the notification instance in the storage. **Returns**

- sl_wisun_coap_notify_t * pointer to notify instance if succeeded, otherwise NULL ptr

sl_wisun_coap_notify_init

> void sl_wisun_coap_notify_init (void)

Initialization of the CoAP notification service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

If the service is enabled, the notification thread is initialized also.

sl_wisun_coap_notify_init_default_inst

> sl_status_t sl_wisun_coap_notify_init_default_inst (sl_wisun_coap_notify_t *const notify, const char * id)

Init default notification instance.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_coap_notify_t *const	[in]	notify	notify descriptor for init
const char *	[in]	id	unique ID of the notification (custom string)

Default remote address and port are defined in the config file. **Returns**

- sl_status_t SL_STATUS_OK if the initialization is succeeded, otherwise SL_STATUS_FAIL

sl_wisun_coap_notify_print_notifications

> void sl_wisun_coap_notify_print_notifications (void)

Print notifications.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Print content of the internal storage.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Print resource URI paths

sl_wisun_coap_notify_remove_by_id

```
sl_status_t sl_wisun_coap_notify_remove_by_id (const char * id)
```

Remove notification from the CoAP notification table referenced by name.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	id	Notification ID

Remove notification from the internal storage. **Returns**

- sl_status_t SL_STATUS_OK if the notification is removed successfully, otherwise SL_STATUS_FAIL

sl_wisun_coap_notify_send_notification

```
sl_status_t sl_wisun_coap_notify_send_notification (const sl_wisun_coap_notify_t *const notify)
```

Send notification.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_coap_notify_t *const	[in]	notify	Notification instance. It includes all the necessary information about destination.

Notification should be initialised properly before sending. **Returns**

- sl_status_t SL_STATUS_OK if the notification is sent successfully, otherwise SL_STATUS_FAIL

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Type definitions

Modules

sl_wisun_coap_notify

Typedefs

typedef struct sl_wisun_coap_notify	sl_wisun_coap_notify_t CoAP Notify type definition.

Typedef Documentation

sl_wisun_coap_notify_t

```
typedef struct sl_wisun_coap_notify sl_wisun_coap_notify_t
```

CoAP Notify type definition.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_coap_notify

CoAP Notify type definition.

Public Attributes

bool(*	condition_cb Condition check callback, it returns with the result of send condition (true: send, false: not send)
sl_wisun_coap_packet_t *(*	hnd_cb Packet handler callback, it returns with the built CoAP packet.
const char *	id ID string.
sockaddr_in6_t	remote_addr Remote address.
uint32_t	schedule_time_ms Schedule time (ms)
uint32_t	tick_ms Tick counter (ms)

Public Attribute Documentation

condition_cb

```
bool(* sl_wisun_coap_notify::condition_cb) (const struct sl_wisun_coap_notify *notify)
```

Condition check callback, it returns with the result of send condition (true: send, false: not send)

hnd_cb

```
sl_wisun_coap_packet_t *(* sl_wisun_coap_notify::hnd_cb) (const struct sl_wisun_coap_notify *notify)
```

Packet handler callback, it returns with the built CoAP packet.

id

```
const char* sl_wisun_coap_notify::id
```

ID string.

remote_addr

```
sockaddr_in6_t sl_wisun_coap_notify::remote_addr
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Remote address.

schedule_time_ms

```
uint32_t sl_wisun_coap_notify::schedule_time_ms
```

Schedule time (ms)

tick_ms

```
uint32_t sl_wisun_coap_notify::tick_ms
```

Tick counter (ms)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Ping

Ping

The Ping component implements a ping service using the ICMPv6 protocol. It can send ICMPv6 Echo Request, receive corresponding responses, and calculate round-trip latency.

Applications can emit a single ping packet using the sl_wisun_ping_request() API. To receive the pong response, the sl_wisun_ping_response() API is called. The sl_wisun_ping() function offers a straightforward solution for periodic ping packet transmission and reception. To halt the process, use the sl_wisun_ping_stop() function.

Ping test parameters such as size, number of pings sent, and timeout for response absence are configurable. Initialization of the component is done by calling the sl_wisun_ping_init() function.

The following example demonstrates how the ping request can be called:

```
#include "sl_wisun_ping.h"

static void ping_remote(void) {
  int32_t ret = 0;
  sockaddr_in6_t remote_addr = { 0 };
  sl_status_t status = SL_STATUS_FAIL;

  // Create address structure from ping
  ret = inet_pton(AF_INET6,
                  "2001:db8::1",
                  &remote_addr.sin6_addr);

  if (ret != 1) {
    // Error handling
    return;
  }

  // Send and receive ping requests and responses
  status = sl_wisun_ping(&remote_addr, // Ping remote_addr IPv6 address
                         10, // with 10 packets
                         512, // packet length is 512 bytes
                         NULL, // Custom statitistic handler is not required
                         NULL); // Request/Response handler is not required
  if (status != SL_STATUS_OK) {
    // Error handling
    return;
  }
}
```

Modules

Configurations

Type definitions

Functions

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

void sl_wisun_ping_stop(void)
Stop the current ping process.

Macros

#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.

#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_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_TYPE_ECHO_REQUEST (128U)
Ping echo request type field value.

#define SL_WISUN_PING_TYPE_ECHO_RESPONSE (129U)
Ping echo response type field value.

Function Documentation

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

Type	Direction	Argument Name	Description
const sockaddr_in6_t *const	[in]	remote_addr	Remote destination address
const uint16_t	[in]	packet_count	Count of packets
const uint16_t	[in]	packet_length	ICMPv6 packet length including header
sl_wisun_ping_stat_hnd_t	[in]	stat_hnd	Custom statistic handler function
sl_wisun_ping_req_resp_done_hnd_t	[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

Copyright © 2025 Silicon Laboratories. All rights reserved.

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL or SL_STATUS_ABORT

sl_wisun_ping_init

```
void sl_wisun_ping_init (void )
```

Initialize the ping service module.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function initializes the service thread, mutex, and message queues.

sl_wisun_ping_request

```
void sl_wisun_ping_request (const sl_wisun_ping_info_t *const ping_request)
```

Send a ping request.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_ping_info_t *const	[in]	ping_request	Ping Request Information

The function sends a single ICMPv6 request.

sl_wisun_ping_response

```
void sl_wisun_ping_response (sl_wisun_ping_info_t *const ping_response)
```

Retrieve a ping response.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_ping_info_t *const	[out]	ping_response	Ping Response Information

The function retrieves a ping response information.

sl_wisun_ping_stop

```
void sl_wisun_ping_stop (void )
```

Stop the current ping process.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
void	N/A		

Reset request and response queues and send a special ping request with interrupt ping status.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define SL_WISUN_PING_MAX_PAYLOAD_LENGTH 144U
Maximum length of payload.

#define SL_WISUN_PING_PACKET_COUNT 4U
Default packet count for ping.

#define SL_WISUN_PING_PACKET_INTERVAL 1000U
Packet Interval.

#define SL_WISUN_PING_PACKET_SIZE 40U
Default packet size for ping.

#define SL_WISUN_PING_STACK_SIZE_WORD 256U
Ping thread stack size in word.

#define SL_WISUN_PING_TIMEOUT_MS 10000U
Ping timeout.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Modules

sl_wisun_ping_echo_request

sl_wisun_ping_info

sl_wisun_ping_stat

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 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.
typedef void(*	sl_wisun_ping_stat_hnd_t)(sl_wisun_ping_stat_t *stat) Ping statistic typedef.
typedef struct sl_wisun_ping_stat	sl_wisun_ping_stat_t Statistic ping type definition.

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.

sl_wisun_ping_echo_response_t

```
typedef sl_wisun_ping_echo_request_t sl_wisun_ping_echo_response_t
```

Ping response type definition.

sl_wisun_ping_info_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_wisun_ping_info sl_wisun_ping_info_t
```

Ping info structure type definition.

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.

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.

sl_wisun_ping_stat_t

```
typedef struct sl_wisun_ping_stat sl_wisun_ping_stat_t
```

Statistic ping type definition.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_ping_echo_request

Ping echo request packed structure type definitions.

Public Attributes

uint16_t	checksum Checksum.
uint8_t	code Code.
uint16_t	identifier Identifier.
uint8_t	payload Payload array.
uint16_t	sequence_number Sequence number.
uint8_t	type type

Public Attribute Documentation

checksum

```
uint16_t sl_wisun_ping_echo_request::checksum
```

Checksum.

code

```
uint8_t sl_wisun_ping_echo_request::code
```

Code.

identifier

```
uint16_t sl_wisun_ping_echo_request::identifier
```

Identifier.

payload

```
uint8_t sl_wisun_ping_echo_request::payload[SL_WISUN_PING_MAX_PAYLOAD_LENGTH]
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Payload array.

sequence_number

```
uint16_t sl_wisun_ping_echo_request::sequence_number
```

Sequence number.

type

```
uint8_t sl_wisun_ping_echo_request::type
```

type

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_ping_info

Ping info structure type definition.

Public Attributes

uint16_t	identifier Identifier.
bool	lost Lost packet flag.
uint16_t	packet_length ICMPv6 packet length including header.
sockaddr_in6_t	remote_addr Wi-SUN remote address.
uint32_t	response_time_ms Response time millisecond.
uint16_t	sequence_number Sequence number.
uint32_t	start_time_stamp Start time stamp.
uint32_t	stop_time_stamp Stop time stamp.

Public Attribute Documentation

identifier

```
uint16_t sl_wisun_ping_info::identifier
```

Identifier.

lost

```
bool sl_wisun_ping_info::lost
```

Lost packet flag.

packet_length

```
uint16_t sl_wisun_ping_info::packet_length
```

ICMPv6 packet length including header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
sockaddr_in6_t sl_wisun_ping_info::remote_addr
```

Wi-SUN remote address.

response_time_ms

```
uint32_t sl_wisun_ping_info::response_time_ms
```

Response time millisecond.

sequence_number

```
uint16_t sl_wisun_ping_info::sequence_number
```

Sequence number.

start_time_stamp

```
uint32_t sl_wisun_ping_info::start_time_stamp
```

Start time stamp.

stop_time_stamp

```
uint32_t sl_wisun_ping_info::stop_time_stamp
```

Stop time stamp.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_ping_stat

Statistic ping type definition.

Public Attributes

uint32_t	avg_time_ms Average echo time millisecond.
uint16_t	lost Lost packet count.
uint32_t	max_time_ms Maximum echo time millisecond.
uint32_t	min_time_ms Minimum echo time millisecond.
uint16_t	packet_count Packet count.
uint16_t	packet_length Packet length.
sockaddr_in6_t	remote_addr Wi-SUN remote address.

Public Attribute Documentation

avg_time_ms

```
uint32_t sl_wisun_ping_stat::avg_time_ms
```

Average echo time millisecond.

lost

```
uint16_t sl_wisun_ping_stat::lost
```

Lost packet count.

max_time_ms

```
uint32_t sl_wisun_ping_stat::max_time_ms
```

Maximum echo time millisecond.

min_time_ms

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint32_t sl_wisun_ping_stat::min_time_ms

Minimum echo time millisecond.

packet_count

uint16_t sl_wisun_ping_stat::packet_count

Packet count.

packet_length

uint16_t sl_wisun_ping_stat::packet_length

Packet length.

remote_addr

sockaddr_in6_t sl_wisun_ping_stat::remote_addr

Wi-SUN remote address.

Copyright © 2025 Silicon Laboratories. All rights reserved.

iPerf

iPerf

The iPerf component offers a solution compatible with iPerf2 for measuring UDP throughput. It provides comprehensive UDP support and implements both server and client modes, enabling the exchange of packets to assess bandwidth performance, inter-arrival jitter, and packet loss. The component relies on various functions, to configure and execute your iPerf test. Parameters such as the server and client ports, preferred bandwidth, packet count, and remote peer address can be customized. Upon completion of each iPerf test, an iPerf report is generated.

To integrate the iPerf component into the application, simply add it to the project and initialize it using sl_iperf_service_init().

The following examples show how you can initialize a iPerf server and then an iPerf client

```
#include <string.h>
#include "cmsis_os2.h"
#include "sl_iperf.h"
#include "sl_iperf_util.h"

static void exec_iperf_server(void) {
  sl_iperf_test_t srv_test = { 0 };

  // Initialize Server test
  sl_iperf_test_init(&srv_test, SL_IPERF_MODE_SERVER, SL_IPERF_IPROTOV6_UDP);

  // Set minimum time duration for the test
  srv_test.opt.duration_ms = 10000;

  // Add the test to the execution queue
  if (sl_iperf_test_add(&srv_test) == false) {
    // Error handling
    return;
  }

  // Waiting for the test result
  if (sl_iperf_test_get(&srv_test, osWaitForever) == false) {
    // Error handling
    return;
  }

  // Print the result in JSON format
  sl_iperf_print_test_log_json(&srv_test);
}
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
static void exec_iperf_client(void)
{
  sl_iperf_test_t clnt_test = { 0 };

  // Initialize Client test
  sl_iperf_test_init(&clnt_test, SL_IPERF_MODE_CLIENT, SL_IPERF_IPROTOV6_UDP);

  // Configure test parameters
  // Set the remote server address
  strncpy(clnt_test.opt.remote_addr, "2001:db8::1", SL_IPERF_IP_STR_BUFF_LEN);

  // Set the bandwidth to 10 kBit/s
  clnt_test.opt.bandwidth = 10000;

  // Add the test to the execution queue
  if (sl_iperf_test_add(&clnt_test) == false) {
    // Error handling
    return;
  }

  // Waiting for the test result
  if (sl_iperf_test_get(&clnt_test, osWaitForever) == false) {
    // Error handling
    return;
  }

  // Print the result in JSON format
  sl_iperf_print_test_log_json(&clnt_test);
}
```

Modules

- Configurations
- Type definitions
- Portable interface

Functions

void sl_iperf_service_init(void)
Initialize the iPerf service.

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_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_buff(sl_iperf_test_t *const test)
Set the default internal test buffer.

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_udp_client(sl_iperf_test_t *test)
Execute the iPerf UDP client test.

Copyright © 2025 Silicon Laboratories. All rights reserved.

void sl_iperf_test_udp_server(sl_iperf_test_t *test)
Execute the iPerf UDP server test.

Macros

#define SL_IPERF_HEADER_EPOCH_START (0x00001000UL)
iPerf header epoch start 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_TIME_MODE (0x80000000UL)
iPerf header time mode mask

#define SL_IPERF_HEADER_TRIPTIME (0x00000010UL)
iPerf header triptime mask

#define SL_IPERF_HEADER_UDPTEST (0x20000000UL)
iPerf header UDP test mask

#define SL_IPERF_HEADER_VERSION1 (0x80000000UL)
iPerf header version1 mask

#define SL_IPERF_HEADER_VERSION2 (0x04000000UL)
iPerf header version2 mask

#define SL_IPERF_SERVER_UDP_TX_FINACK_COUNT (10U)
iPerf Server TX FINACK max count to retry

#define SL_IPERF_UDP_SERVER_FIN_ACK_SIZE (128U)
iPerf UDP server final ack size

Function Documentation

sl_iperf_service_init

```
void sl_iperf_service_init (void )
```

Initialize the iPerf service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Init OS objects and default contents

sl_iperf_test_add

```
bool sl_iperf_test_add (sl_iperf_test_t *const test)
```

Add the iPerf test to the execution queue.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
sl_iperf_test_t *const	[in]	test	Test descriptor

Add test to the input messagequeue. **Returns**

- true On Success
- false On Failure

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

Type	Direction	Argument Name	Description
sl_iperf_test_t *const	[out]	test	Destination test descriptor
const uint32_t	[in]	timeout_ms	Timeout for getting test from messagequeue

Get the test from the output messagequeue. **Returns**

- true On Success
- false On Failure

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

Type	Direction	Argument Name	Description
sl_iperf_test_t *const	[inout]	test	Test descriptor
sl_iperf_mode_t	[in]	mode	Mode
sl_iperf_protocol_t	[in]	protocol	Protocol

Initialize a test descriptor with default content

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

Type	Direction	Argument Name	Description
sl_iperf_test_t *const	[out]	test	Test descriptor

Helper function

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Type	Direction	Argument Name	Description
sl_iperf_test_t *const	[out]	test	Test descriptor

Helper function

sl_iperf_test_udp_client

> void sl_iperf_test_udp_client (sl_iperf_test_t * test)

Execute the iPerf UDP client test.

Parameters

Type	Direction	Argument Name	Description
sl_iperf_test_t *	[inout]	test	Test descriptor

iPerf UDP client test.

sl_iperf_test_udp_server

> void sl_iperf_test_udp_server (sl_iperf_test_t * test)

Execute the iPerf UDP server test.

Parameters

Type	Direction	Argument Name	Description
sl_iperf_test_t *	[inout]	test	Test descriptor

iPerf UDP server test.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define SL_IPERF_BUFFER_SIZE 1232U
Server receiver buffer size.

#define SL_IPERF_CLIENT_DEFAULT_BANDWIDTH_BPS 20000U
Default bandwidth in bits/sec.

#define SL_IPERF_CLIENT_DEFAULT_SEND_TIME_MS 10000U
Default sending time in millisec.

#define SL_IPERF_DEFAULT_INTERVAL_MS 1000U
Default statistic update interval.

#define SL_IPERF_LOG_JSON_INDENT 2U
Set json formatted log indent in space count.

#define SL_IPERF_MAX_TEST_TIMEOUT_MS 2000U
Maximum possible additional test timeout after duration time.

#define SL_IPERF_MESSAGE_QUEUE_SIZE 16U
Set test message queue size.

#define SL_IPERF_SERVER_DEFAULT_PORT 5001U
Default Server port.

#define SL_IPERF_SOCKET_ADDR_LEN 32U
Set the address length buffer size.

#define SL_IPERF_STACK_SIZE_WORD 256U
Set test thread stack size in word depth.

#define SL_IPERF_VERBOSE_MODE 0U
Enable verbose prints.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Type definitions

Modules

- sl_iperf_clnt_hdr_ext
- sl_iperf_clnt_hdr_ext_isoch_settings
- sl_iperf_clnt_hdr_ext_starttime_fq
- sl_iperf_clnt_hdr_isoch_payload
- sl_iperf_conn
- sl_iperf_log
- sl_iperf_log_str_buff
- sl_iperf_opt
- sl_iperf_stats
- sl_iperf_test
- sl_iperf_udp_clnt_hdr
- sl_iperf_udp_clnt_hdr_v1
- sl_iperf_udp_datagram
- sl_iperf_udp_srv_hdr

Enumerations

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

enum sl_iperf_mode {

SL_IPERF_MODE_SERVER
SL_IPERF_MODE_CLIENT

}

iPerf mode

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_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.

Typedefs

typedef struct sl_iperf_clnt_hdr_ext_isoch_settings — sl_iperf_clnt_hdr_ext_isoch_settings_t
iPerf Client Isochronous settings

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 — 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 Isochronous payload

typedef struct sl_iperf_udp_clnt_hdr_v1 — sl_iperf_clnt_hdr_v1_t
iPerf CLient Header v1

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 int32_t(* sl_iperf_log_print_t)(sl_iperf_log_t *const log, const char *format,...)
Printer function 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 enum sl_iperf_mode — sl_iperf_mode_t
iPerf mode

Copyright © 2025 Silicon Laboratories. All rights reserved.

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 enum sl_iperf_status	sl_iperf_status_t Test status type definition.
typedef void(*	sl_iperf_test_callback_t)(sl_iperf_test_t *) iPerf Test callback type definition
typedef uint16_t	sl_iperf_test_id_t Test ID type definition.
typedef struct sl_iperf_test	sl_iperf_test_t iPerf test descriptor
typedef struct sl_iperf_udp_clnt_hdr	sl_iperf_udp_clnt_hdr_t iPerf Client UDP header
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

Macros

#define	SL_IPERF_IP_STR_BUFF_LEN (40U) iPerf ip address string max length definition

Enumeration Documentation

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

sl_iperf_mode

> sl_iperf_mode

iPerf mode

Enumerator

SL_IPERF_MODE_SERVER	Server mode.
SL_IPERF_MODE_CLIENT	Client mode.

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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_IPERF_TEST_STATUS_DONE	Test done with no error.
SL_IPERF_TEST_STATUS_ERR	Test done with error.

Typedef Documentation

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 Isochronous settings

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

sl_iperf_clnt_hdr_ext_t

```
typedef struct sl_iperf_clnt_hdr_ext sl_iperf_clnt_hdr_ext_t
```

iPerf Client Header extended

sl_iperf_clnt_hdr_isoch_payload_t

```
typedef struct sl_iperf_clnt_hdr_isoch_payload sl_iperf_clnt_hdr_isoch_payload_t
```

iPerf Client Isochronous payload

sl_iperf_clnt_hdr_v1_t

```
typedef struct sl_iperf_udp_clnt_hdr_v1 sl_iperf_clnt_hdr_v1_t
```

iPerf CLient Header v1

sl_iperf_conn_t

```
typedef struct sl_iperf_conn sl_iperf_conn_t
```

iPerf connection descriptor type definition

sl_iperf_error_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef enum sl_iperf_err sl_iperf_error_t
```

iPerf error enumeration type definition

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.

sl_iperf_log_str_buff_t

```
typedef struct sl_iperf_log_str_buff sl_iperf_log_str_buff_t
```

Log string buffer type definition.

sl_iperf_log_t

```
typedef struct sl_iperf_log sl_iperf_log_t
```

iPerf log type definition

sl_iperf_mode_t

```
typedef enum sl_iperf_mode sl_iperf_mode_t
```

iPerf mode

sl_iperf_opt_bw_format

```
typedef enum sl_iperf_opt_bw_format sl_iperf_opt_bw_format
```

Bandwidth format enumeration type definition.

sl_iperf_opt_t

```
typedef struct sl_iperf_opt sl_iperf_opt_t
```

iPerf test option type definition

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_iperf_stats sl_iperf_stats_t
```

iPerf statistic data type definition

sl_iperf_status_t

```
typedef enum sl_iperf_status sl_iperf_status_t
```

Test status type definition.

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

sl_iperf_test_id_t

```
typedef uint16_t sl_iperf_test_id_t
```

Test ID type definition.

sl_iperf_test_t

```
typedef struct sl_iperf_test sl_iperf_test_t
```

iPerf test descriptor

sl_iperf_udp_clnt_hdr_t

```
typedef struct sl_iperf_udp_clnt_hdr sl_iperf_udp_clnt_hdr_t
```

iPerf Client UDP header

sl_iperf_udp_datagram_t

```
typedef struct sl_iperf_udp_datagram sl_iperf_udp_datagram_t
```

iPerf UDP datagram structure type definition

sl_iperf_udp_srv_hdr_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_iperf_udp_srv_hdr sl_iperf_udp_srv_hdr_t
```

iPerf server header

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_clnt_hdr_ext

iPerf Client Header extended

Public Attributes

int16_t	l_flags Lower flags.
uint32_t	l_rate Lower rate.
uint32_t	l_version Lower version.
int32_t	length Length.
uint16_t	reserved Reserved.
uint32_t	tcp_write_prefetch TCP write prefetch.
uint16_t	tos Tos.
int32_t	type Type.
int16_t	u_flags Upper flags.
uint32_t	u_rate Upper rate.
uint32_t	u_version Upper version.

Public Attribute Documentation

l_flags

```
int16_t sl_iperf_clnt_hdr_ext::l_flags
```

Lower flags.

l_rate

```
uint32_t sl_iperf_clnt_hdr_ext::l_rate
```

Lower rate.

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint32_t sl_iperf_clnt_hdr_ext::l_version

Lower version.

length

int32_t sl_iperf_clnt_hdr_ext::length

Length.

reserved

uint16_t sl_iperf_clnt_hdr_ext::reserved

Reserved.

tcp_write_prefetch

uint32_t sl_iperf_clnt_hdr_ext::tcp_write_prefetch

TCP write prefetch.

tos

uint16_t sl_iperf_clnt_hdr_ext::tos

Tos.

type

int32_t sl_iperf_clnt_hdr_ext::type

Type.

u_flags

int16_t sl_iperf_clnt_hdr_ext::u_flags

Upper flags.

u_rate

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
uint32_t sl_iperf_clnt_hdr_ext::u_rate
```

Upper rate.

u_version

```
uint32_t sl_iperf_clnt_hdr_ext::u_version
```

Upper version.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_clnt_hdr_ext_isoch_settings

iPerf Client Isochronous settings

Public Attributes

int32_t	l_burst_ipg Lower Burst IPG.
int32_t	l_fps Lower FPS.
int32_t	l_mean Lower Mean.
int32_t	l_variance Lower Variance.
int32_t	u_burst_ipg Upper Burst IPG.
int32_t	u_fps Upper FPS.
int32_t	u_mean Upper Mean.
int32_t	u_variance Upper Variance.

Public Attribute Documentation

l_burst_ipg

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_burst_ipg
```

Lower Burst IPG.

l_fps

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_fps
```

Lower FPS.

l_mean

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_mean
```

Lower Mean.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::l_variance
```

Lower Variance.

u_burst_ipg

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_burst_ipg
```

Upper Burst IPG.

u_fps

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_fps
```

Upper FPS.

u_mean

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_mean
```

Upper Mean.

u_variance

```
int32_t sl_iperf_clnt_hdr_ext_isoch_settings::u_variance
```

Upper Variance.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_clnt_hdr_ext_starttime_fq

iPerf Client header extended FQ start time

Public Attributes

uint32_t	l_fq_rate Lower FQ rate.
uint32_t	reserved Reserved.
uint32_t	start_tv_sec Start sec.
uint32_t	start_tv_usec Start usec.
uint32_t	u_fq_rate Upper FQ rate.

Public Attribute Documentation

l_fq_rate

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::l_fq_rate
```

Lower FQ rate.

reserved

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::reserved
```

Reserved.

start_tv_sec

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::start_tv_sec
```

Start sec.

start_tv_usec

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::start_tv_usec
```

Start usec.

Copyright © 2025 Silicon Laboratories. All rights reserved.

u_fq_rate

```
uint32_t sl_iperf_clnt_hdr_ext_starttime_fq::u_fq_rate
```

Upper FQ rate.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_clnt_hdr_isoch_payload

iPerf Client Isochronous payload

Public Attributes

uint32_t	burst_period period units microseconds
uint32_t	burst_size Burst size.
uint32_t	frame_id Frame ID.
uint32_t	prev_frameid Previous frame ID.
uint32_t	remaining Remaining.
uint32_t	reserved Reserved.
uint32_t	start_tv_sec Start sec.
uint32_t	start_tv_usec Start usec.

Public Attribute Documentation

burst_period

```
uint32_t sl_iperf_clnt_hdr_isoch_payload::burst_period
```

period units microseconds

burst_size

```
uint32_t sl_iperf_clnt_hdr_isoch_payload::burst_size
```

Burst size.

frame_id

```
uint32_t sl_iperf_clnt_hdr_isoch_payload::frame_id
```

Frame ID.

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint32_t sl_iperf_clnt_hdr_isoch_payload::prev_frameid

Previous frame ID.

remaining

uint32_t sl_iperf_clnt_hdr_isoch_payload::remaining

Remaining.

reserved

uint32_t sl_iperf_clnt_hdr_isoch_payload::reserved

Reserved.

start_tv_sec

uint32_t sl_iperf_clnt_hdr_isoch_payload::start_tv_sec

Start sec.

start_tv_usec

uint32_t sl_iperf_clnt_hdr_isoch_payload::start_tv_usec

Start usec.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_conn

iPerf connection descriptor type definition

Public Attributes

uint8_t *	buff Buffer ptr to receive/transmit.
size_t	buff_size RX/TX buffer transmit.
sl_iperf_socket_addr_t	clnt_addr Client sock addr IP.
bool	run Server (rx'd) or client (tx'd) started.
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.

Public Attribute Documentation

buff

```
uint8_t* sl_iperf_conn::buff
```

Buffer ptr to receive/transmit.

buff_size

```
size_t sl_iperf_conn::buff_size
```

RX/TX buffer transmit.

clnt_addr

```
sl_iperf_socket_addr_t sl_iperf_conn::clnt_addr
```

Client sock addr IP.

run

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
bool sl_iperf_conn::run
```

Server (rx'd) or client (tx'd) started.

socket_id

```
int32_t sl_iperf_conn::socket_id
```

local socket id

socket_id_clnt

```
int32_t sl_iperf_conn::socket_id_clnt
```

Accepted sock used by TCP server to rx.

srv_addr

```
sl_iperf_socket_addr_t sl_iperf_conn::srv_addr
```

Server sock addr IP.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_log

iPerf log type definition

Public Attributes

sl_iperf_log_str_buff_t	buff Buff string instance.
bool	buffered Buffered.
bool	colored Colored.
int32_t	last_res Last result.
int32_t(*	print Stdout and buffer printer.

Public Attribute Documentation

buff

```
sl_iperf_log_str_buff_t sl_iperf_log::buff
```

Buff string instance.

buffered

```
bool sl_iperf_log::buffered
```

Buffered.

colored

```
bool sl_iperf_log::colored
```

Colored.

last_res

```
int32_t sl_iperf_log::last_res
```

Last result.

Copyright © 2025 Silicon Laboratories. All rights reserved.

print

```
int32_t(* sl_iperf_log::print) (struct sl_iperf_log *const log, const char *format,...)
```

Stdout and buffer printer.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_log_str_buff

Log string buffer type definition.

Public Attributes

char *	buff	Buff ptr.
char *	pos	Position ptr.
size_t	size	Size.

Public Attribute Documentation

buff

```
char* sl_iperf_log_str_buff::buff
```

Buff ptr.

pos

```
char* sl_iperf_log_str_buff::pos
```

Position ptr.

size

```
size_t sl_iperf_log_str_buff::size
```

Size.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_opt

iPerf test option type definition

Public Attributes

uint32_t	bandwidth expected bandwidth in bits/sec
uint16_t	buf_len Buf len to tx or rx.
sl_iperf_opt_bw_format	bw_format Bandwidth format.
uint16_t	duration_ms Time in sec to tx.
uint16_t	interval_ms Interval (ms) between bandwidth update.
sl_iperf_mode_t	mode Server or client mode.
bool	multicast Join multicast group (address is stored in 'remote_addr')
uint16_t	packet_nbr Nbr of packets to tx.
bool	persistent Server in persistent mode.
uint16_t	port Server or client port.
sl_iperf_protocol_t	protocol UDP or TCP protocol.
char	remote_addr Remote address for TX.
uint16_t	win_size Win size to tx or rx.

Public Attribute Documentation

bandwidth

```
uint32_t sl_iperf_opt::bandwidth
```

expected bandwidth in bits/sec

buf_len

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint16_t sl_iperf_opt::buf_len

Buf len to tx or rx.

bw_format

sl_iperf_opt_bw_format sl_iperf_opt::bw_format

Bandwidth format.

duration_ms

uint16_t sl_iperf_opt::duration_ms

Time in sec to tx.

interval_ms

uint16_t sl_iperf_opt::interval_ms

Interval (ms) between bandwidth update.

mode

sl_iperf_mode_t sl_iperf_opt::mode

Server or client mode.

multicast

bool sl_iperf_opt::multicast

Join multicast group (address is stored in 'remote_addr')

packet_nbr

uint16_t sl_iperf_opt::packet_nbr

Nbr of packets to tx.

persistent

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
bool sl_iperf_opt::persistent
```

Server in persistent mode.

port

```
uint16_t sl_iperf_opt::port
```

Server or client port.

protocol

```
sl_iperf_protocol_t sl_iperf_opt::protocol
```

UDP or TCP protocol.

remote_addr

```
char sl_iperf_opt::remote_addr[SL_IPERF_IP_STR_BUFF_LEN]
```

Remote address for TX.

win_size

```
uint16_t sl_iperf_opt::win_size
```

Win size to tx or rx.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_stats

iPerf statistic data type definition

Public Attributes

uint32_t	bandwidth Rx or Tx cur bandwidth in bits/s.
uint32_t	bytes Nbr of bytes rx'd or tx'd on net.
bool	end_err Err with UDP FIN or FINACK.
uint32_t	errs Nbr of rx or tx errs.
uint32_t	finack_bw Bandwidth in Final ACK in bits/s (calculated from FinACK tot length and FinACK duration ms)
sl_iperf_ts_ms_t	finack_duration_ms Time duration in Final ACK.
uint32_t	finack_pkt Packet count in Final ACK.
uint32_t	finack_tot_len Total length of received bytes in Final ACK.
uint32_t	last_recv_pkt_cnt Last received packets (for update)
uint32_t	nbr_calls Nbr of I/O sys calls.
uint32_t	nbr_rcv_snt_packets Nbr of rx'd or tx'd packets.
uint32_t	tot_packets Nbr of rx'd or tx'd and lost packets.
uint32_t	transitory_error_cnts Nbr of transitory err.
sl_iperf_ts_ms_t	ts_curr_recv_ms Current received packet timestamp.
sl_iperf_ts_ms_t	ts_curr_sent_ms Current sent packet timestamp.
sl_iperf_ts_ms_t	ts_end_ms End timestamp (ms).
sl_iperf_ts_ms_t	ts_prev_recv_ms Previous received packet timestamp.
sl_iperf_ts_ms_t	ts_prev_sent_ms Previous sent packet timestamp.
sl_iperf_ts_ms_t	ts_start_ms

Copyright © 2025 Silicon Laboratories. All rights reserved.

Start timestamp (ms).

bool	udp_async_error First UDP pkt rx'd.
uint32_t	udp_dup_pkt Nbr of pkt ID rx'd more than once.
int64_t	udp_jitter UDP jitter.
uint32_t	udp_lost_pkt Nbr of UDP pkt lost.
uint32_t	udp_out_of_order Nbr of pkt rx'd out of order.
int32_t	udp_rx_last_pkt Prev pkt ID rx'd.

Public Attribute Documentation

bandwidth

```
uint32_t sl_iperf_stats::bandwidth
```

Rx or Tx cur bandwidth in bits/s.

bytes

```
uint32_t sl_iperf_stats::bytes
```

Nbr of bytes rx'd or tx'd on net.

end_err

```
bool sl_iperf_stats::end_err
```

Err with UDP FIN or FINACK.

errs

```
uint32_t sl_iperf_stats::errs
```

Nbr of rx or tx errs.

finack_bw

```
uint32_t sl_iperf_stats::finack_bw
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Bandwidth in Final ACK in bits/s (calculated from FinACK tot length and FinACK duration ms)

finack_duration_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::finack_duration_ms
```

Time duration in Final ACK.

finack_pkt

```
uint32_t sl_iperf_stats::finack_pkt
```

Packet count in Final ACK.

finack_tot_len

```
uint32_t sl_iperf_stats::finack_tot_len
```

Total length of received bytes in Final ACK.

last_recv_pkt_cnt

```
uint32_t sl_iperf_stats::last_recv_pkt_cnt
```

Last received packets (for update)

nbr_calls

```
uint32_t sl_iperf_stats::nbr_calls
```

Nbr of I/O sys calls.

nbr_rcv_snt_packets

```
uint32_t sl_iperf_stats::nbr_rcv_snt_packets
```

Nbr of rx'd or tx'd packets.

tot_packets

```
uint32_t sl_iperf_stats::tot_packets
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Nbr of rx'd or tx'd and lost packets.

transitory_error_cnts

```
uint32_t sl_iperf_stats::transitory_error_cnts
```

Nbr of transitory err.

ts_curr_recv_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_curr_recv_ms
```

Current received packet timestamp.

ts_curr_sent_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_curr_sent_ms
```

Current sent packet timestamp.

ts_end_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_end_ms
```

End timestamp (ms).

ts_prev_recv_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_prev_recv_ms
```

Previous received packet timestamp.

ts_prev_sent_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_prev_sent_ms
```

Previous sent packet timestamp.

ts_start_ms

```
sl_iperf_ts_ms_t sl_iperf_stats::ts_start_ms
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Start timestamp (ms).

udp_async_error

```
bool sl_iperf_stats::udp_async_error
```

First UDP pkt rx'd.

udp_dup_pkt

```
uint32_t sl_iperf_stats::udp_dup_pkt
```

Nbr of pkt ID rx'd more than once.

udp_jitter

```
int64_t sl_iperf_stats::udp_jitter
```

UDP jitter.

udp_lost_pkt

```
uint32_t sl_iperf_stats::udp_lost_pkt
```

Nbr of UDP pkt lost.

udp_out_of_order

```
uint32_t sl_iperf_stats::udp_out_of_order
```

Nbr of pkt rx'd out of order.

udp_rx_last_pkt

```
int32_t sl_iperf_stats::udp_rx_last_pkt
```

Prev pkt ID rx'd.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_test

iPerf test descriptor

Public Attributes

void(*	cb Callback.
sl_iperf_conn_t	conn Connection.
sl_iperf_error_t	err Error.
sl_iperf_test_id_t	id Test ID.
sl_iperf_log_t *	log Log object ptr.
sl_iperf_opt_t	opt Options.
sl_iperf_stats_t	statistic Statistics.
sl_iperf_status_t	status Status.

Public Attribute Documentation

cb

```
void(* sl_iperf_test::cb) (struct sl_iperf_test *)
```

Callback.

conn

```
sl_iperf_conn_t sl_iperf_test::conn
```

Connection.

err

```
sl_iperf_error_t sl_iperf_test::err
```

Error.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
sl_iperf_test_id_t sl_iperf_test::id
```

Test ID.

log

```
sl_iperf_log_t* sl_iperf_test::log
```

Log object ptr.

opt

```
sl_iperf_opt_t sl_iperf_test::opt
```

Options.

statistic

```
sl_iperf_stats_t sl_iperf_test::statistic
```

Statistics.

status

```
sl_iperf_status_t sl_iperf_test::status
```

Status.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_udp_clnt_hdr

iPerf Client UDP header

Public Attributes

sl_iperf_clnt_hdr_v1_t	base Base v1 header.
sl_iperf_udp_datagram_t	dtg Datagram.
sl_iperf_clnt_hdr_ext_t	extend Extended header.
sl_iperf_clnt_hdr_isoch_payload_t	isoch Isochronous payload.
sl_iperf_clnt_hdr_ext_isoch_settings_t	isoch_settings Isochronous settings.
sl_iperf_clnt_hdr_ext_starttime_fq_t	start_fq Extended start FQ.

Public Attribute Documentation

base

```
sl_iperf_clnt_hdr_v1_t sl_iperf_udp_clnt_hdr::base
```

Base v1 header.

dtg

```
sl_iperf_udp_datagram_t sl_iperf_udp_clnt_hdr::dtg
```

Datagram.

extend

```
sl_iperf_clnt_hdr_ext_t sl_iperf_udp_clnt_hdr::extend
```

Extended header.

isoch

```
sl_iperf_clnt_hdr_isoch_payload_t sl_iperf_udp_clnt_hdr::isoch
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Isochronous payload.

isoch_settings

```
sl_iperf_clnt_hdr_ext_isoch_settings_t sl_iperf_udp_clnt_hdr::isoch_settings
```

Isochronous settings.

start_fq

```
sl_iperf_clnt_hdr_ext_starttime_fq_t sl_iperf_udp_clnt_hdr::start_fq
```

Extended start FQ.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_udp_clnt_hdr_v1

iPerf CLient Header v1

Public Attributes

int32_t	amount Amount.
int32_t	buf_len Buffer Length.
int32_t	flags Flags.
int32_t	num_threads Number of threads.
int32_t	port Port.
int32_t	win_band Win band.

Public Attribute Documentation

amount

```
int32_t sl_iperf_udp_clnt_hdr_v1::amount
```

Amount.

buf_len

```
int32_t sl_iperf_udp_clnt_hdr_v1::buf_len
```

Buffer Length.

flags

```
int32_t sl_iperf_udp_clnt_hdr_v1::flags
```

Flags.

num_threads

```
int32_t sl_iperf_udp_clnt_hdr_v1::num_threads
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Number of threads.

port

```
int32_t sl_iperf_udp_clnt_hdr_v1::port
```

Port.

win_band

```
int32_t sl_iperf_udp_clnt_hdr_v1::win_band
```

Win band.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_udp_datagram

iPerf UDP datagram structure type definition

Public Attributes

int32_t	id Packet id.
int32_t	id2 Packet id.
uint32_t	time_var_sec Time variable for sec.
uint32_t	time_var_usec Time variable for usec.

Public Attribute Documentation

id

```
int32_t sl_iperf_udp_datagram::id
```

Packet id.

id2

```
int32_t sl_iperf_udp_datagram::id2
```

Packet id.

time_var_sec

```
uint32_t sl_iperf_udp_datagram::time_var_sec
```

Time variable for sec.

time_var_usec

```
uint32_t sl_iperf_udp_datagram::time_var_usec
```

Time variable for usec.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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	jitter_sec Jitter hi.
uint32_t	jitter_usec Jitter low.
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	stop_sec Stop time in sec.
uint32_t	stop_usec Stop time in usec.
uint32_t	tot_len_l Tot bytes rx'd low part.
uint32_t	tot_len_u Tot bytes rx'd hi part.

Public Attribute Documentation

dtg

```
sl_iperf_udp_datagram_t sl_iperf_udp_srv_hdr::dtg
```

Udp datagram.

flags

```
int32_t sl_iperf_udp_srv_hdr::flags
```

Server flag.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
uint32_t sl_iperf_udp_srv_hdr::jitter_sec
```

Jitter hi.

jitter_usec

```
uint32_t sl_iperf_udp_srv_hdr::jitter_usec
```

Jitter low.

lost_pkt_cnt

```
uint32_t sl_iperf_udp_srv_hdr::lost_pkt_cnt
```

Lost pkt cnt.

out_of_order_cnt

```
uint32_t sl_iperf_udp_srv_hdr::out_of_order_cnt
```

Rx pkt out of order cnt.

packet_cnt

```
uint32_t sl_iperf_udp_srv_hdr::packet_cnt
```

Packet count.

stop_sec

```
uint32_t sl_iperf_udp_srv_hdr::stop_sec
```

Stop time in sec.

stop_usec

```
uint32_t sl_iperf_udp_srv_hdr::stop_usec
```

Stop time in usec.

tot_len_l

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
uint32_t sl_iperf_udp_srv_hdr::tot_len_l
```

Tot bytes rx'd low part.

tot_len_u

```
uint32_t sl_iperf_udp_srv_hdr::tot_len_u
```

Tot bytes rx'd hi part.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Portable interface

Modules

sl_iperf_socket_addr

sl_iperf_socket_in6addr

sl_iperf_time

Enumerations

enum sl_iperf_protocol {

SL_IPERF_IPROTOV6_UDP = 0
SL_IPERF_IPROTOV6_TCP
SL_IPERF_IPROTOV4_UDP
SL_IPERF_IPROTOV4_TCP

}

iPerf socket IP protocol type definition enum

Typedefs

typedef enum sl_iperf_protocol — sl_iperf_protocol_t
iPerf socket IP protocol type definition enum

typedef struct sl_iperf_socket_addr — sl_iperf_socket_addr_t
Socket address type definition wrapper.

typedef struct sl_iperf_socket_in6addr — sl_iperf_socket_in6addr_t
Raw ip address byes.

typedef struct sl_iperf_time — sl_iperf_time_t
iPerf time data type definition

typedef uint64_t — sl_iperf_ts_ms_t
iPerf timestamp millisecond data type definition

Variables

const sl_iperf_socket_in6addr_t — sl_iperf_socket_inaddr_any
Anyaddress constant.

Functions

void sl_iperf_delay_ms(const uint32_t ms)
iPerf millisec delay

__STATIC_INLINE void sl_iperf_get_socket_addr(const sl_iperf_socket_addr_t *const src_addr, void *const dest_addr)
Get socket address.

Copyright © 2025 Silicon Laboratories. All rights reserved.

void	sl_iperf_get_socket_addr_ip(const sl_iperf_socket_addr_t *const addr, void *const ip) Get IP address bytes from iPerf address structure.
uint16_t	sl_iperf_get_socket_addr_port(const sl_iperf_socket_addr_t *const addr) Get port number of iperf address structure.
sl_iperf_ts_ms_t	sl_iperf_get_timestamp_ms(void) Get millisec timestamp.
const char *	sl_iperf_inet_ntop(const sl_iperf_socket_addr_t *const src_addr, char *const dst_str, const size_t size) Convert IPv4 and IPv6 addresses from binary to text form.
int32_t	sl_iperf_inet_pton(const char *src_str, sl_iperf_socket_addr_t *const dst_addr) Convert addresses from text to binary form.
int32_t	sl_iperf_join_multicast_group(const int32_t sockid, const sl_iperf_socket_addr_t *const multicast_addr) Join multicast group.
int32_t	sl_iperf_leave_multicast_group(const int32_t sockid, const sl_iperf_socket_addr_t *const multicast_addr) Leave multicast group.
uint32_t	sl_iperf_network_htonl(uint32_t val) iPerf host to network long
uint16_t	sl_iperf_network_htons(uint16_t val) iPerf host to network short
bool	sl_iperf_network_is_connected(void) Check network connection.
uint32_t	sl_iperf_network_ntohl(uint32_t val) iPerf network to host long
uint16_t	sl_iperf_network_ntohs(uint16_t val) iPerf network to host short
void	sl_iperf_network_wait_for_connection(void) Wait for network connection.
void	sl_iperf_nw_interface_init(void) Init iPerf Network Interface.
__STATIC_INLINE void	sl_iperf_set_socket_addr(sl_iperf_socket_addr_t *const dest_addr, const void *const src_addr) Set iPerf socket address.
void	sl_iperf_set_socket_addr_family(sl_iperf_socket_addr_t *const addr) Set iPerf socket address family.
void	sl_iperf_set_socket_addr_ip(sl_iperf_socket_addr_t *const addr, const void *const ip) Set IP address bytes in iPerf address structure.
void	sl_iperf_set_socket_addr_port(sl_iperf_socket_addr_t *const addr, const uint16_t port) Set port in iperf address structure.
int32_t	sl_iperf_socket_accept(int32_t sockid, sl_iperf_socket_addr_t *addr) Accept a connection on a socket.
int32_t	sl_iperf_socket_bind(int32_t sockid, const sl_iperf_socket_addr_t *addr) Bind a name to an iPerf socket.
int32_t	sl_iperf_socket_close(int32_t sockid) Close socket.

Copyright © 2025 Silicon Laboratories. All rights reserved.

int32_t sl_iperf_socket_connect(int32_t sockid, const sl_iperf_socket_addr_t *addr)
Initiate a connection on a socket.

int32_t sl_iperf_socket_create(sl_iperf_protocol_t protocol)
Create iPerf socket.

int32_t sl_iperf_socket_getsockopt(int32_t sockid, int32_t level, int32_t optname, void *optval, size_t *optlen)
Get socket option.

int32_t sl_iperf_socket_listen(int32_t sockid, int32_t backlog)
Listen for connections on an iPerf socket.

int32_t sl_iperf_socket_recv(int32_t sockid, void *buff, size_t len)
Receive a message from a socket.

int32_t sl_iperf_socket_recvfrom(int32_t sockid, void *buf, uint32_t len, sl_iperf_socket_addr_t *src_addr)
Receive messages from a socket.

int32_t sl_iperf_socket_send(int32_t sockid, const void *buff, size_t len)
Send a message on a socket,.

int32_t sl_iperf_socket_sendto(int32_t sockid, const void *buff, uint32_t len, const sl_iperf_socket_addr_t *dest_addr)
Send a message on a socket.

int32_t sl_iperf_socket_setsockopt(int32_t sockid, int32_t level, int32_t optname, const void *optval, size_t optlen)
Set socket option.

Macros

#define sl_iperf_assert_sock_addr_len (__type)
Assert socket address length macro function.

#define SL_IPERF_IN6ADDR_SIZE (16U)
Size definition of IPv6 address.

#define SL_IPERF_NW_API_ERROR (-1)
Define network api ERROR value.

Enumeration Documentation

sl_iperf_protocol

> sl_iperf_protocol

iPerf socket IP protocol type definition enum

Enumerator	
SL_IPERF_IPROTOV6_UDP	UDP IPv6 protocol.
SL_IPERF_IPROTOV6_TCP	TCP IPv6 protocol.
SL_IPERF_IPROTOV4_UDP	UDP IPv4 protocol.
SL_IPERF_IPROTOV4_TCP	TCP IPv4 protocol.

Typedef Documentation

sl_iperf_protocol_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef enum sl_iperf_protocol sl_iperf_protocol_t
```

iPerf socket IP protocol type definition enum

sl_iperf_socket_addr_t

```
typedef struct sl_iperf_socket_addr sl_iperf_socket_addr_t
```

Socket address type definition wrapper.

sl_iperf_socket_in6addr_t

```
typedef struct sl_iperf_socket_in6addr sl_iperf_socket_in6addr_t
```

Raw ip address byes.

sl_iperf_time_t

```
typedef struct sl_iperf_time sl_iperf_time_t
```

iPerf time data type definition

sl_iperf_ts_ms_t

```
typedef uint64_t sl_iperf_ts_ms_t
```

iPerf timestamp millisecond data type definition

Variable Documentation

sl_iperf_socket_inaddr_any

```
const sl_iperf_socket_in6addr_t sl_iperf_socket_inaddr_any
```

Anyaddress constant.

Function Documentation

sl_iperf_delay_ms

```
void sl_iperf_delay_ms (const uint32_t ms)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

iPerf millisec delay

Parameters

Type	Direction	Argument Name	Description
const uint32_t	[in]	ms	Millisec value

Customizable delay function

sl_iperf_get_socket_addr

> __STATIC_INLINE void sl_iperf_get_socket_addr (const sl_iperf_socket_addr_t *const src_addr, void *const dest_addr)

Get socket address.

Parameters

Type	Direction	Argument Name	Description
const sl_iperf_socket_addr_t *const	[in]	src_addr	Source address (iPerf type)
void *const	[inout]	dest_addr	Destination address (custom type)

Create a full copy from iperf source address into the custom destination address.

sl_iperf_get_socket_addr_ip

> void sl_iperf_get_socket_addr_ip (const sl_iperf_socket_addr_t *const addr, void *const ip)

Get IP address bytes from iPerf address structure.

Parameters

Type	Direction	Argument Name	Description
const sl_iperf_socket_addr_t *const	[in]	addr	iPerf address
void *const	[out]	ip	IP address destination buffer

Getter function of IP address

sl_iperf_get_socket_addr_port

> uint16_t sl_iperf_get_socket_addr_port (const sl_iperf_socket_addr_t *const addr)

Get port number of iperf address structure.

Parameters

Type	Direction	Argument Name	Description
const sl_iperf_socket_addr_t *const	[in]	addr	iPerf address

Getter function of port property **Returns**

- uint16_t Port number

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_get_timestamp_ms

```
sl_iperf_ts_ms_t sl_iperf_get_timestamp_ms (void )
```

Get millisec timestamp.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Get CPU tick convert to millisec **Returns**

- sl_iperf_ts_ms_t Timestamp in millisec

sl_iperf_inet_ntop

```
const char * sl_iperf_inet_ntop (const sl_iperf_socket_addr_t *const src_addr, char *const dst_str, const size_t size)
```

Convert IPv4 and IPv6 addresses from binary to text form.

Parameters

Type	Direction	Argument Name	Description
const sl_iperf_socket_addr_t *const	[in]	src_addr	Source address in byte form
char *const	[out]	dst_str	Destination buffer ptr
const size_t	[in]	size	Size of the destination buffer.

Converts the network address structure src in the af address family into a character string. **Returns**

- It returns a non-null pointer to dst. NULL is returned if there was an error

sl_iperf_inet_pton

```
int32_t sl_iperf_inet_pton (const char * src_str, sl_iperf_socket_addr_t *const dst_addr)
```

Convert addresses from text to binary form.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	src_str	Source string
sl_iperf_socket_addr_t *const	[out]	dst_addr	Destination address pointer

It converts the character string src into a network address structure. **Returns**

- 1 on success, -1 on error (POSIX described the 0 value too)

sl_iperf_join_multicast_group

```
int32_t sl_iperf_join_multicast_group (const int32_t sockid, const sl_iperf_socket_addr_t *const multicast_addr)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Join multicast group.

Parameters

Type	Direction	Argument Name	Description
const int32_t	[in]	sockid	Socket ID to join
const sl_iperf_socket_addr_t *const	[in]	multicast_addr	Multicast group address

Join socket to multicast group to get packets on group address **Returns**

- int32_t

sl_iperf_leave_multicast_group

> int32_t sl_iperf_leave_multicast_group (const int32_t sockid, const sl_iperf_socket_addr_t *const multicast_addr)

Leave multicast group.

Parameters

Type	Direction	Argument Name	Description
const int32_t	[in]	sockid	Socket ID to join
const sl_iperf_socket_addr_t *const	[in]	multicast_addr	Multicast group address

Remove socket from multicast group **Returns**

- int32_t

sl_iperf_network_htonl

> uint32_t sl_iperf_network_htonl (uint32_t val)

iPerf host to network long

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	val	Value

htonl implementation for iPerf **Returns**

- uint32_t Converted value

sl_iperf_network_htons

> uint16_t sl_iperf_network_htons (uint16_t val)

iPerf host to network short

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint16_t	[in]	val	Value

htons implementation for iPerf **Returns**

- uint16_t Converted value

sl_iperf_network_is_connected

```
bool sl_iperf_network_is_connected (void )
```

Check network connection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Checking the connection status. **Returns**

- true Network is connected
- false Network is NOT connected

sl_iperf_network_ntohl

```
uint32_t sl_iperf_network_ntohl (uint32_t val)
```

iPerf network to host long

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	val	Value

ntohl implementation for iPerf **Returns**

- uint32_t Converted value

sl_iperf_network_ntohs

```
uint16_t sl_iperf_network_ntohs (uint16_t val)
```

iPerf network to host short

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	val	Value

ntohs implementation for iPerf **Returns**

- uint16_t Converted value

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_network_wait_for_connection

```
void sl_iperf_network_wait_for_connection (void )
```

Wait for network connection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Waiting for the network connected status.

sl_iperf_nw_interface_init

```
void sl_iperf_nw_interface_init (void )
```

Init iPerf Network Interface.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Initialize network interface

sl_iperf_set_socket_addr

```
__STATIC_INLINE void sl_iperf_set_socket_addr (sl_iperf_socket_addr_t *const dest_addr, const void *const src_addr)
```

Set iPerf socket address.

Parameters

Type	Direction	Argument Name	Description
sl_iperf_socket_addr_t *const	[inout]	dest_addr	Destination address (custom type)
const void *const	[in]	src_addr	Source address (iPerf type)

Cerate a full copy of custom source address to the destination iperf address.

sl_iperf_set_socket_addr_family

```
void sl_iperf_set_socket_addr_family (sl_iperf_socket_addr_t *const addr)
```

Set iPerf socket address family.

Parameters

Type	Direction	Argument Name	Description
sl_iperf_socket_addr_t *const	[out]	addr	Socket address structure

Copyright © 2025 Silicon Laboratories. All rights reserved.

Set address family in socket address structure

sl_iperf_set_socket_addr_ip

```
void sl_iperf_set_socket_addr_ip (sl_iperf_socket_addr_t *const addr, const void *const ip)
```

Set IP address bytes in iPerf address structure.

Parameters

Type	Direction	Argument Name	Description
sl_iperf_socket_addr_t *const	[inout]	addr	iPerf address
const void *const	[in]	ip	IP address source buffer

Setter function of IP address

sl_iperf_set_socket_addr_port

```
void sl_iperf_set_socket_addr_port (sl_iperf_socket_addr_t *const addr, const uint16_t port)
```

Set port in iperf address structure.

Parameters

Type	Direction	Argument Name	Description
sl_iperf_socket_addr_t *const	[inout]	addr	iPerf address
const uint16_t	[out]	port	Port number

Setter function of port property

sl_iperf_socket_accept

```
int32_t sl_iperf_socket_accept (int32_t sockid, sl_iperf_socket_addr_t * addr)
```

Accept a connection on a socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
sl_iperf_socket_addr_t *	[inout]	addr	Is filled in with the address of the peer (remote) socket In Wi-SUN case, it shouldn't be NULL ptr, but for external sockets should be.

It is used with connection-based socket types (TCP). It extracts the first connection request on the queue of pending connections for the listening socket. **Returns**

- On success, return with the socket id for accepted socket, on error -1 is returned.

sl_iperf_socket_bind

Copyright © 2025 Silicon Laboratories. All rights reserved.

int32_t sl_iperf_socket_bind (int32_t sockid, const sl_iperf_socket_addr_t * addr)

Bind a name to an iPerf socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
const sl_iperf_socket_addr_t *	[in]	addr	address structure ptr

Assigns the address specified by addr to the socket referred to by the socket id. It is normally necessary to assign a local address using bind() before a SOCK_STREAM socket may receive connections **Returns**

- On success, zero is returned. On error, -1 is returned

sl_iperf_socket_close

int32_t sl_iperf_socket_close (int32_t sockid)

Close socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id

Close socket and remove from the socket handler storage **Returns**

- if any error occurred, return with -1, otherwise return with the socket id

sl_iperf_socket_connect

int32_t sl_iperf_socket_connect (int32_t sockid, const sl_iperf_socket_addr_t * addr)

Initiate a connection on a socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
const sl_iperf_socket_addr_t *	[in]	addr	If the socket sockid is of type SOCK_DGRAM, then 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.

Connects the socket referred to by the socketid to the address specified by addr. **Returns**

- If the connection or binding succeeds, zero is returned. On error, -1 is returned.

sl_iperf_socket_create

Copyright © 2025 Silicon Laboratories. All rights reserved.

int32_t sl_iperf_socket_create (sl_iperf_protocol_t protocol)

Create iPerf socket.

Parameters

Type	Direction	Argument Name	Description
sl_iperf_protocol_t	[in]	protocol	Protocol

Create a socket by selected protocol and return with the socket ID **Returns**

- int32_t Socket ID

sl_iperf_socket_getsockopt

int32_t sl_iperf_socket_getsockopt (int32_t sockid, int32_t level, int32_t optname, void * optval, size_t * optlen)

Get socket option.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
int32_t	[in]	level	Not used in Wi-SUN domain.
int32_t	[in]	optname	Option name.
void *	[out]	optval	Destination option value pointer
size_t *	[in]	optlen	Must be the size of sl_wisun_socket_option_data_t union

The function gets socket option by optname, and copy option data to optval ptr **Returns**

- Return 0 on success, other wise -1

sl_iperf_socket_listen

int32_t sl_iperf_socket_listen (int32_t sockid, int32_t backlog)

Listen for connections on an iPerf socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
int32_t	[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

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 **Returns**

- On success, zero is returned. On error, -1 is returned

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_socket_recv

```
int32_t sl_iperf_socket_recv (int32_t sockid, void * buff, size_t len)
```

Receive a message from a socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
void *	[out]	buff	destination buffer ptr
size_t	[in]	len	length of data to read

It should be used for connection-oriented protocol (TCP) **Returns**

- return the number of bytes received, or -1 if an error occurred

sl_iperf_socket_recvfrom

```
int32_t sl_iperf_socket_recvfrom (int32_t sockid, void * buf, uint32_t len, sl_iperf_socket_addr_t * src_addr)
```

Receive messages from a socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
void *	[out]	buf	destination buffer ptr
uint32_t	[in]	len	length of data to read
sl_iperf_socket_addr_t *	[in]	src_addr	Source address

It can be used to receive data on a socket whether or not it is connection-oriented. **Returns**

- return the number of bytes received, or -1 if an error occurred

sl_iperf_socket_send

```
int32_t sl_iperf_socket_send (int32_t sockid, const void * buff, size_t len)
```

Send a message on a socket,.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
const void *	[in]	buff	buffer pointer to send
size_t	[in]	len	length of buffer to send

It is preferred with connection-oriented sockets (TCP). **Returns**

- On success, these calls return the number of bytes sent. On error, -1 is returned

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_socket_sendto

> int32_t sl_iperf_socket_sendto (int32_t sockid, const void * buff, uint32_t len, const sl_iperf_socket_addr_t * dest_addr)

Send a message on a socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
const void *	[in]	buff	buffer pointer to send
uint32_t	[in]	len	length of buffer to send
const sl_iperf_socket_addr_t *	[in]	dest_addr	destination address ptr, the structure must be prepared for UDP sockets

It is preferred in datagram mode (UDP). **Returns**

- On success, these calls return the number of bytes sent. On error, -1 is returned

sl_iperf_socket_setsockopt

> int32_t sl_iperf_socket_setsockopt (int32_t sockid, int32_t level, int32_t optname, const void * optval, size_t optlen)

Set socket option.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
int32_t	[in]	level	Not used in Wi-SUN domain.
int32_t	[in]	optname	Option name.
const void *	[in]	optval	Option value structure pointer. For Wi-SUN it is casted to sl_wisun_socket_option_t
size_t	[in]	optlen	Must be the size of sl_wisun_socket_option_data_t union

This function can set socket properties. There are some limitation in Wi-SUN domain: optlen and level args are not used. optname and optval are casted to the corresponding Wi-SUN types: sl_wisun_socket_option_t and sl_wisun_socket_option_data_t. sl_wisun_socket_option_data_t is a union type to include all of implemented Wi-SUN socket options **Returns**

- Return 0 on success, other wise -1

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_socket_addr

Socket address type definition wrapper.

Public Attributes

uint8_t	addr Addres structure bytes.

Public Attribute Documentation

addr

```
uint8_t sl_iperf_socket_addr::addr[SL_IPERF_SOCKET_ADDR_LEN]
```

Addres structure bytes.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_socket_in6addr

Raw ip address byes.

Public Attributes

uint8_t	ip IP address bytes.

Public Attribute Documentation

ip

```
uint8_t sl_iperf_socket_in6addr::ip[SL_IPERF_IN6ADDR_SIZE]
```

IP address bytes.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_iperf_time

iPerf time data type definition

Public Attributes

uint32_t	sec Seconds.
uint32_t	usec Microseconds.

Public Attribute Documentation

sec

```
uint32_t sl_iperf_time::sec
```

Seconds.

usec

```
uint32_t sl_iperf_time::usec
```

Microseconds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

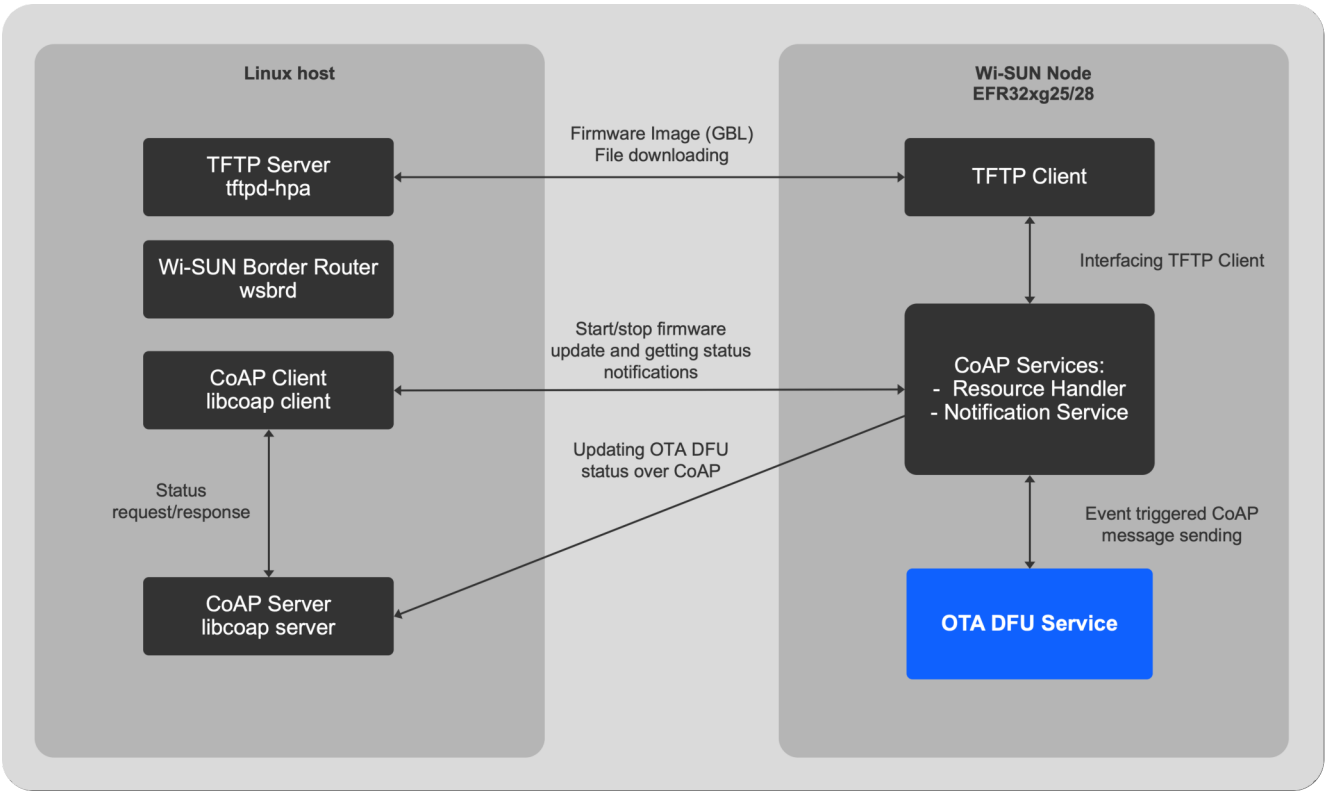
Over-The-Air Device Firmware Upgrade (Alpha)

Over-The-Air Device Firmware Upgrade (Alpha)

The Wi-SUN Over-The-Air Device Firmware Upgrade (OTA DFU) service is implemented to facilitate the updating of Wi-SUN device firmware. This component requires the Gecko Bootloader API to execute firmware writing, verification, and bootload operations.

To obtain a new Gecko Bootloader File (GBL) from a remote host Over-The-Air, a Trivial File Transfer Protocol (TFTP) client is employed, leveraging the Wi-SUN network. The entire firmware upgrade session is manageable over CoAP, with the service offering remote host connection settings, firmware file selection, notification, and status request capabilities. Configuration of the connection to the TFTP remote host can be accomplished within the component configuration file. To use this component, a bootloader must be flashed onto the device. The component relies on functions that reside within the bootloader. To choose the appropriate bootloader, consider factors such as flash location (internal or external) and size. To save on download time, we recommend applying compression on both the bootloader and device projects.

The following diagram shows the component architecture:



The example below shows how OTA DFU can be performed in any project.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```c
#include <stdio.h>
#include "cmsis_os2.h"
#include "sl_wisun_ota_dfu.h"

static void start_ota_dfu(void) {
  sl_status_t api_status = SL_STATUS_FAIL;
  uint32_t fw_update_status = 0;

  // Set remote host that stores the firmware image.
  // TFTP protocol is used to download the firmware.
  // 69 is the default TFTP Port
  api_status = sl_wisun_ota_dfu_set_host_addr("2001:db8::1", 69);
  if (api_status != SL_STATUS_OK) {
    // Error handling
    return;
  }

  // Set GBL file path
  // The path is relative to the preconfigured TFTP shared directory
  api_status = sl_wisun_ota_dfu_set_gbl_path("my_new_firmware.gbl");
  if (api_status != SL_STATUS_OK) {
    // Error handling
    return;
  }

  // Start firmware update
  api_status = sl_wisun_ota_dfu_start_fw_update();
  if (api_status != SL_STATUS_OK) {
    // Error handling
    return;
  }

  // Waiting for the firmware download to finish.
  while(1) {
    fw_update_status = sl_wisun_ota_dfu_get_fw_update_status();
    if (fw_update_status & (1 << SL_WISUN_OTA_DFU_STATUS_FW_DOWNLOADED)) {
      printf("Firmware downloaded. Preparing to bootload\n");
      break;
    } else if (fw_update_status & (1 << SL_WISUN_OTA_DFU_STATUS_FW_DOWNLOAD_ERROR)) {
      printf("Firmware download failed\n");
      break;
    }
    osDelay(1);
  }
}
```

The CoAP remote settings can be configured as follows:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
coap-client -m get -N -B 3 -t text coap://[$WISUN_NODE_IPV6_ADDR]:5683/ota/dfu -e "help"

Commands:

[P: POST (set), G: GET (get)]
[G] help
[G] (empty payload) status
[P] start
[P] stop
[P] install
[G|P] host_address <ipv6> <port>
[G|P] gbl <path>
[G|P] uri <path>
[G|P] notify_host_address <ipv6> <port>
[G|P] notify_uri <path>
[G|P] notify_chunk_cnt <count>
```

Modules

Type definitions

Configurations

Functions

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)
void	sl_wisun_ota_dfu_free_fw_update_status_json_str(const char *str) Free the status string buffer.
sl_status_t	sl_wisun_ota_dfu_get_fw_update_statistics(sl_wisun_ota_dfu_statistics_t *const stats) Get firmware update statistics.
uint32_t	sl_wisun_ota_dfu_get_fw_update_status(void) Get the status value.
bool	sl_wisun_ota_dfu_get_fw_update_status_flag(const sl_wisun_ota_dfu_status_t status_flag) Get the status flag value.
const char *	sl_wisun_ota_dfu_get_fw_update_status_json_str(void) Get the status string in JSON format.
sl_status_t	sl_wisun_ota_dfu_get_gbl_path(char *const dst_gbl_path, const uint16_t dst_gbl_path_size) Get GBL file path.
sl_status_t	sl_wisun_ota_dfu_get_host_addr(char *const dst_host, const uint16_t dst_host_size, uint16_t *dst_port) Get host addres and port.
sl_status_t	sl_wisun_ota_dfu_get_host_uri(char *const dst_uri, const uint16_t dst_uri_size) Get host uri path.
void	sl_wisun_ota_dfu_init(void) Initialize the device firmware upgrade service.
sl_status_t	sl_wisun_ota_dfu_reboot_and_install(void) Reboot device.
sl_status_t	sl_wisun_ota_dfu_set_gbl_path(const char *gbl_path) Set GBL file path.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_status_t sl_wisun_ota_dfu_set_host_addr(const char *host, const uint16_t port)
Set host address and port.

sl_status_t sl_wisun_ota_dfu_set_host_uri(const char *uri)
Set host uri path.

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.

Macros

#define SL_WISUN_OTA_DFU_COAP_URI_PATH_STR_BUF_LEN 128U
OTA DFU CoAP URI Path string buffer length.

#define SL_WISUN_OTA_DFU_GBL_PATH_STR_BUF_LEN 256U
OTA DFU GBL File name string buffer length.

#define SL_WISUN_OTA_DFU_IPV6_STR_BUF_LEN 40U
OTA DFU IPV6 string buffer length.

#define SL_WISUN_OTA_DFU_SERVICE_LOOP ()
OTA DFU Service loop.

Function Documentation

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

Type	Direction	Argument Name	Description
const sl_wisun_ota_dfu_error_code_t	[in]	error_code	Error code
sl_wisun_ota_dfu_error_ctx_t *const	[in]	ctx	Error context with error details

Catch error in different stages of boot load.

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

Type	Direction	Argument Name	Description
const char *	[in]	str	String ptr

Call CoAP free to release allocated memory

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_ota_dfu_get_fw_update_statistics

> sl_status_t sl_wisun_ota_dfu_get_fw_update_statistics (sl_wisun_ota_dfu_statistics_t *const stats)

Get firmware update statistics.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_ota_dfu_statistics_t *const	[out]	stats	Pointer to statistics structure

Get firmware update statistics **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise failure

sl_wisun_ota_dfu_get_fw_update_status

> uint32_t sl_wisun_ota_dfu_get_fw_update_status (void)

Get the status value.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returning the value of event flags **Returns**

- uint32_t Status

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

Type	Direction	Argument Name	Description
const sl_wisun_ota_dfu_status_t	[in]	status_flag	Status flag enum

Bool representation of status variable bit value **Returns**

- bool true if the flag is set, otherwise false

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
void	N/A		

String buffer is allocated in heap by CoAP allocator **Returns**

- const char * Allocated string pointer on success, otherwise NULL

sl_wisun_ota_dfu_get_gbl_path

```
sl_status_t sl_wisun_ota_dfu_get_gbl_path (char *const dst_gbl_path, const uint16_t dst_gbl_path_size)
```

Get GBL file path.

Parameters

Type	Direction	Argument Name	Description
char *const	[out]	dst_gbl_path	GBL file path
const uint16_t	[in]	dst_gbl_path_size	GBL path size

Get GBL file path **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_ota_dfu_get_host_addr

```
sl_status_t sl_wisun_ota_dfu_get_host_addr (char *const dst_host, const uint16_t dst_host_size, uint16_t * dst_port)
```

Get host addres and port.

Parameters

Type	Direction	Argument Name	Description
char *const	[out]	dst_host	Host address
const uint16_t	[in]	dst_host_size	Host address size
uint16_t *	[out]	dst_port	Host port

Get host address and port **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_ota_dfu_get_host_uri

```
sl_status_t sl_wisun_ota_dfu_get_host_uri (char *const dst_uri, const uint16_t dst_uri_size)
```

Get host uri path.

Parameters

Type	Direction	Argument Name	Description
char *const	[out]	dst_uri	Uri path
const uint16_t	[in]	dst_uri_size	Uri path size

Copyright © 2025 Silicon Laboratories. All rights reserved.

Get host uri path **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_ota_dfu_init

```
void sl_wisun_ota_dfu_init (void )
```

Initialize the device firmware upgrade service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Initialize Wi-SUN OTA Device Firmware Upgrade service.

sl_wisun_ota_dfu_reboot_and_install

```
sl_status_t sl_wisun_ota_dfu_reboot_and_install (void )
```

Reboot device.

Parameters

Type	Direction	Argument Name	Description
void	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

sl_wisun_ota_dfu_set_gbl_path

```
sl_status_t sl_wisun_ota_dfu_set_gbl_path (const char * gbl_path)
```

Set GBL file path.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	gbl_path	GBL file path

Set GBL file path **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_ota_dfu_set_host_addr

```
sl_status_t sl_wisun_ota_dfu_set_host_addr (const char * host, const uint16_t port)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Set host address and port.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	host	Host address
const uint16_t	[in]	port	Host port

Set host address and port **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_ota_dfu_set_host_uri

```
sl_status_t sl_wisun_ota_dfu_set_host_uri (const char * uri)
```

Set host uri path.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	uri	Uri path

Set host uri path **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_wisun_ota_dfu_start_fw_update

```
sl_status_t sl_wisun_ota_dfu_start_fw_update (void )
```

Start firmware update.

Parameters

Type	Direction	Argument Name	Description
void	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

sl_wisun_ota_dfu_stop_fw_update

```
sl_status_t sl_wisun_ota_dfu_stop_fw_update (void )
```

Stop firmware update.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Type definitions

Modules

- sl_wisun_ota_dfu_error_ctx
- sl_wisun_ota_dfu_error_ctx_btl_fw_set
- sl_wisun_ota_dfu_error_ctx_btl_fw_verify
- sl_wisun_ota_dfu_error_ctx_fw_download
- sl_wisun_ota_dfu_settings
- sl_wisun_ota_dfu_statistics

Enumerations

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.

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.

Typedefs

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_btl_fw_set — sl_wisun_ota_dfu_error_ctx_btl_fw_set_t
Wi-SUN OTA DFU set 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_-

Copyright © 2025 Silicon Laboratories. All rights reserved.

error_ctx_fw_download	sl_wisun_ota_dfu_error_ctx_fw_download_t Wi-SUN OTA DFU download 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.
typedef struct sl_wisun_ota_dfu_settings	sl_wisun_ota_dfu_settings_t Wi-SUN OTA DFU settings typedef.
typedef struct sl_wisun_ota_dfu_statistics	sl_wisun_ota_dfu_statistics_t Wi-SUN OTA DFU statistics typedef.
typedef enum sl_wisun_ota_dfu_status	sl_wisun_ota_dfu_status_t Wi-SUN OTA DFU Status enumeration.

Enumeration Documentation

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.

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.

Typedef Documentation

sl_wisun_ota_dfu_error_code_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef enum sl_wisun_ota_dfu_error_code sl_wisun_ota_dfu_error_code_t
```

Wi-SUN OTA DFU error code enumeration.

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.

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.

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.

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.

sl_wisun_ota_dfu_settings_t

```
typedef struct sl_wisun_ota_dfu_settings sl_wisun_ota_dfu_settings_t
```

Wi-SUN OTA DFU settings typedef.

sl_wisun_ota_dfu_statistics_t

```
typedef struct sl_wisun_ota_dfu_statistics sl_wisun_ota_dfu_statistics_t
```

Wi-SUN OTA DFU statistics typedef.

sl_wisun_ota_dfu_status_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef enum sl_wisun_ota_dfu_status sl_wisun_ota_dfu_status_t
```

Wi-SUN OTA DFU Status enumeration.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_set_t	set Set error context.
sl_wisun_ota_dfu_error_ctx_btl_fw_verify_t	verify Verify 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.

set

```
sl_wisun_ota_dfu_error_ctx_btl_fw_set_t sl_wisun_ota_dfu_error_ctx::set
```

Set error context.

verify

```
sl_wisun_ota_dfu_error_ctx_btl_fw_verify_t sl_wisun_ota_dfu_error_ctx::verify
```

Verify error context.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_ota_dfu_error_ctx_fw_download

Wi-SUN OTA DFU download error context definition.

Public Attributes

uint16_t	data_size Data size.
uint32_t	offset Offset of GBL file.
int32_t	ret_val Return value of API call.

Public Attribute Documentation

data_size

```
uint16_t sl_wisun_ota_dfu_error_ctx_fw_download::data_size
```

Data size.

offset

```
uint32_t sl_wisun_ota_dfu_error_ctx_fw_download::offset
```

Offset of GBL file.

ret_val

```
int32_t sl_wisun_ota_dfu_error_ctx_fw_download::ret_val
```

Return value of API call.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_ota_dfu_settings

Wi-SUN OTA DFU settings typedef.

Public Attributes

char	coap_uri_path CoAP URI path.
char	gbl_path_str GBL file path.
char	host_addr_str Host address string buffer.
uint16_t	host_port Host port.

Public Attribute Documentation

coap_uri_path

```
char sl_wisun_ota_dfu_settings::coap_uri_path[SL_WISUN_OTA_DFU_COAP_URI_PATH_STR_BUF_LEN]
```

CoAP URI path.

gbl_path_str

```
char sl_wisun_ota_dfu_settings::gbl_path_str[SL_WISUN_OTA_DFU_GBL_PATH_STR_BUF_LEN]
```

GBL file path.

host_addr_str

```
char sl_wisun_ota_dfu_settings::host_addr_str[SL_WISUN_OTA_DFU_IPV6_STR_BUF_LEN]
```

Host address string buffer.

host_port

```
uint16_t sl_wisun_ota_dfu_settings::host_port
```

Host port.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_ota_dfu_statistics

Wi-SUN OTA DFU statistics typedef.

Public Attributes

uint32_t	downl_bytes Downloaded bytes.
sl_sleeptimer_date_t	elapsed_time_since_rst Elapsed time since reset.
sl_sleeptimer_date_t	fw_downl_time Firmware download time.
sl_sleeptimer_date_t	fw_update_time Firmware update time.
const char *	global_ip_str Global IP String.
uint32_t	received_cnt Received packet count.
uint32_t	resent_cnt Resent packet count.
sl_sleeptimer_timestamp_t	start_time Start time reference in seconds.

Public Attribute Documentation

downl_bytes

```
uint32_t sl_wisun_ota_dfu_statistics::downl_bytes
```

Downloaded bytes.

elapsed_time_since_rst

```
sl_sleeptimer_date_t sl_wisun_ota_dfu_statistics::elapsed_time_since_rst
```

Elapsed time since reset.

fw_downl_time

```
sl_sleeptimer_date_t sl_wisun_ota_dfu_statistics::fw_downl_time
```

Firmware download time.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
sl_sleeptimer_date_t sl_wisun_ota_dfu_statistics::fw_update_time
```

Firmware update time.

global_ip_str

```
const char* sl_wisun_ota_dfu_statistics::global_ip_str
```

Global IP String.

received_cnt

```
uint32_t sl_wisun_ota_dfu_statistics::received_cnt
```

Received packet count.

resent_cnt

```
uint32_t sl_wisun_ota_dfu_statistics::resent_cnt
```

Resent packet count.

start_time

```
sl_sleeptimer_timestamp_t sl_wisun_ota_dfu_statistics::start_time
```

Start time reference in seconds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define SL_WISUN_OTA_DFU_AUTO_INSTALL_ENABLED 1U
Enable auto reboot and install.

#define SL_WISUN_OTA_DFU_GBL_FILE "wisun_firmware.gbl"
Firmware image (gbl) file name.

#define SL_WISUN_OTA_DFU_HOST_ADDR "2001:db8::1"
TFTP service listening address.

#define SL_WISUN_OTA_DFU_HOST_NOTIFY_ENABLED 1U
Enable host notification.

#define SL_WISUN_OTA_DFU_NOTIFY_DOWNLOAD_CHUNK_CNT 10U
Count of received chunk to notify in download task.

#define SL_WISUN_OTA_DFU_NOTIFY_HOST_ADDR "2001:db8::1"
OTA DFU notification server address.

#define SL_WISUN_OTA_DFU_NOTIFY_PORT 5683U
OTA DFU Host notification listener port.

#define SL_WISUN_OTA_DFU_NOTIFY_URI_PATH "/ota/dfu_notify"
OTA DFU notification resource URI path.

#define SL_WISUN_OTA_DFU_STACK_SIZE_WORD 360UL
OTA DFU service stack size in CPU word.

#define SL_WISUN_OTA_DFU_STORAGE_SLOT_ID 0U
Storage slot ID.

#define SL_WISUN_OTA_DFU_TFTP_DATA_BLOCK_SIZE 1228U
TFTP data block size.

#define SL_WISUN_OTA_DFU_TFTP_PORT 69U
TFTP service listener port.

#define SL_WISUN_OTA_DFU_TFTP_TIMEOUT_SEC 8U
TFTP server retransmission timeout in seconds.

#define SL_WISUN_OTA_DFU_URI_PATH "/ota/dfu"
OTA DFU resource URI path.

#define SL_WISUN_OTA_DFU_VERBOSE_MODE_ENABLED 0U
Enable verbose mode.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Trivial File Transfer Protocol (TFTP) Client (Alpha)

Trivial File Transfer Protocol (TFTP) Client (Alpha)

The TFTP Client facilitates file exchange between a client and any remote host over UDP. This component provides an easy-to-use solution for downloading and uploading files from a remote host. The TFTP client can be initialized with sl_tftp_clnt_init(), and the functions sl_tftp_clnt_get() and sl_tftp_clnt_put() can be used to download and upload files, respectively. Additionally, the block size and timeout options can be configured to negotiate between the server and the client, optimizing the transmission. The component implements the TFTP protocol as specified in RFC 1350.

To ensure proper functionality, a TFTP server, such as tftpd-hpa, must be prepared on the remote host.

The following code example shows how to download a file with TFTP Client:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```c
#include "sl_tftp_clnt.h"

// TFTP data callback
// In this callback you can implement storing data chunks to flash, heap or any other memory.
static void tftp_data_hnd(sl_tftp_clnt_t * const clnt,
                          const uint8_t * const data_ptr,
                          const uint16_t data_size) {
  (void)data_ptr;
  (void)data_size;
  printf("Data received chunk %u (%u bytes)\n",
         clnt->packet.content.data.block_num,
         clnt->packet.content.data.data_size);
}

// TFTP error callback
// You can implement error handling in this callback.
static void tftp_error_hnd(sl_tftp_clnt_t * const clnt,
                           const uint16_t error_code,
                           const char *error_msg) {
  (void)clnt;
  printf("Error (%u): %s\n", error_code, error_msg);
}

static void tftp_get_file(void) {
  static sl_tftp_clnt_t tftp_clnt = { 0 };

  // Initialize Client instance
  if (sl_tftp_clnt_init(&tftp_clnt,
                        "2001:db8::1",
                        SL_TFTP_DEFAULT_HOST_PORT,
                        tftp_data_hnd,
                        tftp_error_hnd) != SL_STATUS_OK) {
    // Error handling
    return;
  }

  // Set blocksize option
  if (sl_tftp_clnt_set_option(&tftp_clnt,
                              SL_TFTP_OPT_EXT_BLOCKSIZE,
                              1228) != SL_STATUS_OK) {
    // Error handling
    return;
  }

  // Start to download a file
  if (sl_tftp_clnt_get(&tftp_clnt,
                       "test.txt") != SL_STATUS_OK) {
    // Error handling
    return;
  }

  while (1) {
    if (sl_tftp_clnt_is_op_finished(&tftp_clnt)) {
      printf("File download finished\n");
      break;
    }
    osDelay(1);
  }
}
```

Modules

Configurations

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Portable interface

Typedefs

typedef void(* sl_tftp_clnt_data_hnd_t)(sl_tftp_clnt_t *const clnt, const uint8_t *const data_ptr, const uint16_t data_size)
Data handler callback definition.

typedef void(* sl_tftp_clnt_error_hnd_t)(sl_tftp_clnt_t *const clnt, const uint16_t error_code, const char *error_msg)
Error handler callback defintion.

Functions

__STATIC_INLINE sl_status_t sl_tftp_clnt_default_init(sl_tftp_clnt_t *const clnt)
Init TFTP Client default.

__STATIC_INLINE sl_status_t sl_tftp_clnt_get(sl_tftp_clnt_t *const clnt, const char *file)
TFT Client Get.

sl_status_t sl_tftp_clnt_init(sl_tftp_clnt_t *const clnt, const char *host, const uint16_t port, sl_tftp_clnt_data_hnd_t data_hnd, sl_tftp_clnt_error_hnd_t error_hnd)
Init TFTP Client.

bool sl_tftp_clnt_is_op_finished(const sl_tftp_clnt_t *const clnt)
TFTP Client is finished operation.

bool sl_tftp_clnt_is_op_get(const sl_tftp_clnt_t *const clnt)
TFTP Client is Get operation.

bool sl_tftp_clnt_is_op_put(const sl_tftp_clnt_t *const clnt)
TFTP Client is Put operation.

bool sl_tftp_clnt_is_op_rrq_wrq_failed(const sl_tftp_clnt_t *const clnt)
TFTP Client is RRQ or WRQ failed.

void sl_tftp_clnt_print_pkt(const sl_tftp_pkt_t *const pkt)
Print packet.

__STATIC_INLINE sl_status_t sl_tftp_clnt_put(sl_tftp_clnt_t *const clnt, const char *file)
TFT Client Put.

sl_status_t sl_tftp_clnt_request(sl_tftp_clnt_t *const clnt, const uint16_t opcode, const char *file, const char *mode)
TFTP Client request.

void sl_tftp_clnt_service_init(void)
Init TFTP Client service.

sl_status_t sl_tftp_clnt_set_option(sl_tftp_clnt_t *const clnt, const char *opt, const uint32_t value)
Set TFTP Client option.

sl_status_t sl_tftp_clnt_terminate_session(sl_tftp_clnt_t *const clnt)
TFTP Client terminate session.

sl_status_t sl_tftp_clnt_wait_op_finished(const sl_tftp_clnt_t *const clnt)
TFTP Client wait operation finished.

Macros

Copyright © 2025 Silicon Laboratories. All rights reserved.

#define SL_TFTP_DEFAULT_DATA_BLOCK_SIZE 512U
TFTP Default data block size.

#define SL_TFTP_DEFAULT_HOST_PORT 69U
Default TFTP host port.

#define SL_TFTP_DEFAULT_SRV_RET_TIMEOUT_SEC 1U
TFTP Default server retransmit timeout interval in seconds.

#define SL_TFTP_ERRORCODE_ACCVIOL 2U
Access violation.

#define SL_TFTP_ERRORCODE_DISKFULL 3U
Disk full or allocation exceeded.

#define SL_TFTP_ERRORCODE_FEXIST 6U
File already exists.

#define SL_TFTP_ERRORCODE_FNOTFOUND 1U
File not found.

#define SL_TFTP_ERRORCODE_ILLEGALOP 4U
Illegal TFTP operation.

#define SL_TFTP_ERRORCODE_NOTDEF 0U
Not defined, see error message (if any).

#define SL_TFTP_ERRORCODE_NOUSR 7U
No such user.

#define SL_TFTP_ERRORCODE_OPTNEGOTFAIL 8U
Terminate transfer due to option negotiation.

#define SL_TFTP_ERRORCODE_UNKNTID 5U
Unknown transfer ID.

#define SL_TFTP_MODE_NETASCII_STR "netascii"
Netascii mode string.

#define SL_TFTP_MODE_NETASCII_STR_LEN 8U
Netascii mode string length.

#define SL_TFTP_MODE_OCTET_STR "octet"
Octet mode string.

#define SL_TFTP_MODE_OCTET_STR_LEN 5U
Octet mode string length.

#define SL_TFTP_OPCODE_ACK 4U
TFTP Acknowledgement operation code.

#define SL_TFTP_OPCODE_DATA 3U
TFTP Data operation code.

#define SL_TFTP_OPCODE_ERROR 5U
TFTP Error operation code.

#define SL_TFTP_OPCODE_OACK 6U
TFTP Option Acknowledgement operation code.

#define SL_TFTP_OPCODE_RRQ 1U
TFTP Read Request operation code.

Copyright © 2025 Silicon Laboratories. All rights reserved.

#define SL_TFTP_OPCODE_WRQ 2U
TFTP Write Request operation code.

#define SL_TFTP_OPT_EXT_BLOCKSIZE "blksize"
TFTP Blocksize Option.

#define SL_TFTP_OPT_EXT_BLOCKSIZE_LEN 7U
TFTP Blocksize Option length.

#define SL_TFTP_OPT_EXT_MULTICAST "multicast"
TFTP Multicast Option.

#define SL_TFTP_OPT_EXT_MULTICAST_LEN 9U
TFTP Multicast Option length.

#define SL_TFTP_OPT_EXT_TIMEOUT_INTERVAL "timeout"
TFTP Timeout Interval Option.

#define SL_TFTP_OPT_EXT_TIMEOUT_INTERVAL_LEN 7U
TFTP Timeout Interval Option length.

#define SL_TFTP_OPT_EXT_TRANSFER_SIZE "tsize"
TFTP Transfer Size Option.

#define SL_TFTP_OPT_EXT_TRANSFER_SIZE_LEN 5U
TFTP Transfer Size Option length.

#define SL_TFTP_OPT_EXT_WINDOWSIZE "windowsize"
TFTP Windowsize Option.

#define SL_TFTP_OPT_EXT_WINDOWSIZE_LEN 10U
TFTP Windowsize Option length.

#define SL_TFTP_SERVICE_LOOP while (1)
TFTP Service loop definition.

#define SL_TFTP_STR_MAX_LEN 256UL
TFTP string max length (filename, mode)

Typedef Documentation

sl_tftp_clnt_data_hnd_t

```
typedef void(* sl_tftp_clnt_data_hnd_t) (sl_tftp_clnt_t *const clnt, const uint8_t *const data_ptr, const uint16_t data_size) )
(sl_tftp_clnt_t *const clnt, const uint8_t *const data_ptr, const uint16_t data_size)
```

Data handler callback definition.

sl_tftp_clnt_error_hnd_t

```
typedef void(* sl_tftp_clnt_error_hnd_t) (sl_tftp_clnt_t *const clnt, const uint16_t error_code, const char *error_msg) )
(sl_tftp_clnt_t *const clnt, const uint16_t error_code, const char *error_msg)
```

Error handler callback defintion.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Function Documentation

sl_tftp_clnt_default_init

> __STATIC_INLINE sl_status_t sl_tftp_clnt_default_init (sl_tftp_clnt_t *const clnt)

Init TFTP Client default.

Parameters

Type	Direction	Argument Name	Description
sl_tftp_clnt_t *const	[inout]	clnt	Client to initialize

Initialize Client with default values from config file parameters: host: SL_TFTP_CLNT_DEFAULT_HOST port: SL_TFTP_DEFAULT_HOST_PORT data_hnd: simple hex dump callback error_hnd: error code and message printer **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

References SL_TFTP_CLNT_DEFAULT_HOST , sl_tftp_clnt_init , and SL_TFTP_DEFAULT_HOST_PORT

sl_tftp_clnt_get

> __STATIC_INLINE sl_status_t sl_tftp_clnt_get (sl_tftp_clnt_t *const clnt, const char * file)

TFT Client Get.

Parameters

Type	Direction	Argument Name	Description
sl_tftp_clnt_t *const	[inout]	clnt	Client
const char *	[in]	file	File name

Get file from server in octet mode **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

References sl_tftp_clnt_request , SL_TFTP_MODE_OCTET_STR , and SL_TFTP_OPCODE_RRQ

sl_tftp_clnt_init

> sl_status_t sl_tftp_clnt_init (sl_tftp_clnt_t *const clnt, const char * host, const uint16_t port, sl_tftp_clnt_data_hnd_t data_hnd, sl_tftp_clnt_error_hnd_t error_hnd)

Init TFTP Client.

Parameters

Type	Direction	Argument Name	Description
sl_tftp_clnt_t *const	[inout]	clnt	Client to initialize
const char *	[in]	host	Host address string
const uint16_t	[in]	port	Port number
sl_tftp_clnt_data_hnd_t	[in]	data_hnd	Data handler callback
sl_tftp_clnt_error_hnd_t	[in]	error_hnd	Error handler callback

Copyright © 2025 Silicon Laboratories. All rights reserved.

Initialize TFTP Client context **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_tftp_clnt_is_op_finished

```
bool sl_tftp_clnt_is_op_finished (const sl_tftp_clnt_t *const clnt)
```

TFTP Client is finished operation.

Parameters

Type	Direction	Argument Name	Description
const sl_tftp_clnt_t *const	[in]	clnt	Client

Check read or write operations whether is in progress **Returns**

- true Is finished
- false Is in progress

sl_tftp_clnt_is_op_get

```
bool sl_tftp_clnt_is_op_get (const sl_tftp_clnt_t *const clnt)
```

TFTP Client is Get operation.

Parameters

Type	Direction	Argument Name	Description
const sl_tftp_clnt_t *const	[in]	clnt	Client

Check Read operation is in progress **Returns**

- true Is in progress otherwise false

sl_tftp_clnt_is_op_put

```
bool sl_tftp_clnt_is_op_put (const sl_tftp_clnt_t *const clnt)
```

TFTP Client is Put operation.

Parameters

Type	Direction	Argument Name	Description
const sl_tftp_clnt_t *const	[in]	clnt	Client

Check Write operation is in progress **Returns**

- true Is in progress otherwise false

sl_tftp_clnt_is_op_rrq_wrq_failed

Copyright © 2025 Silicon Laboratories. All rights reserved.

> bool sl_tftp_clnt_is_op_rrq_wrq_failed (const sl_tftp_clnt_t *const clnt)

TFTP Client is RRQ or WRQ failed.

Parameters

Type	Direction	Argument Name	Description
const sl_tftp_clnt_t *const	[in]	clnt	Client

Check Read or Write Request operation is failed **Returns**

- true Is failed otherwise false

sl_tftp_clnt_print_pkt

> void sl_tftp_clnt_print_pkt (const sl_tftp_pkt_t *const pkt)

Print packet.

Parameters

Type	Direction	Argument Name	Description
const sl_tftp_pkt_t *const	[in]	pkt	Packet to print

Print packet info in json format

sl_tftp_clnt_put

> __STATIC_INLINE sl_status_t sl_tftp_clnt_put (sl_tftp_clnt_t *const clnt, const char * file)

TFT Client Put.

Parameters

Type	Direction	Argument Name	Description
sl_tftp_clnt_t *const	[inout]	clnt	Client
const char *	[in]	file	File name to store data

Send buffer content to file on remote server **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

References sl_tftp_clnt_request , SL_TFTP_MODE_OCTET_STR , and SL_TFTP_OPCODE_WRQ

sl_tftp_clnt_request

> sl_status_t sl_tftp_clnt_request (sl_tftp_clnt_t *const clnt, const uint16_t opcode, const char * file, const char * mode)

TFTP Client request.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
sl_tftp_clnt_t *const	[inout]	clnt	Client
const uint16_t	[in]	opcode	Operation code: SL_TFTP_OPCODE_RRQ or SL_TFTP_OPCODE_WRQ
const char *	[in]	file	File name to put or get
const char *	[in]	mode	Mode string

Send request for Server (RRQ or WRQ) **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

Referenced by sl_tftp_clnt_get , and sl_tftp_clnt_put

sl_tftp_clnt_service_init

```
void sl_tftp_clnt_service_init (void )
```

Init TFTP Client service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Initialize TFTP client OS objects

sl_tftp_clnt_set_option

```
sl_status_t sl_tftp_clnt_set_option (sl_tftp_clnt_t *const clnt, const char * opt, const uint32_t value)
```

Set TFTP Client option.

Parameters

Type	Direction	Argument Name	Description
sl_tftp_clnt_t *const	[inout]	clnt	Client
const char *	[in]	opt	Option string
const uint32_t	[in]	value	Option value

Set TFTP Client option in the TFTP client instance. The supported options are:

- SL_TFTP_OPT_EXT_BLOCKSIZE: Blocksize option (8 - 1228 for Wi-SUN)
- SL_TFTP_OPT_EXT_TIMEOUT_INTERVAL: Timeout interval option (1 - 255) **Returns**
  - sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_tftp_clnt_terminate_session

```
sl_status_t sl_tftp_clnt_terminate_session (sl_tftp_clnt_t *const clnt)
```

TFTP Client terminate session.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
sl_tftp_clnt_t *const	[inout]	clnt	Client

Terminate TFTP session **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

sl_tftp_clnt_wait_op_finished

> sl_status_t sl_tftp_clnt_wait_op_finished (const sl_tftp_clnt_t *const clnt)

TFTP Client wait operation finished.

Parameters

Type	Direction	Argument Name	Description
const sl_tftp_clnt_t *const	[in]	clnt	Client

Wait for operation to finish **Returns**

- sl_status_t SL_STATUS_OK on success, otherwise SL_STATUS_FAIL

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define SL_FTP_ENABLE_FTP_PROTOCOL 0U
Enable FTP TCP based protocol.

#define SL_FTP_ENABLE_TFTP_PROTOCOL 1U
Enable TFTP UDP based protocol.

#define SL_TFTP_CLNT_DEFAULT_HOST "aabb:ccdd::eeff:0011:2233:4455"
TFTP Default remote host address string.

#define SL_TFTP_CLNT_RECV_TIMEOUT_MS 300000UL
TFTP default receive timeout (ms)

#define SL_TFTP_CLNT_STACK_SIZE_WORD 256UL
TFTP Client service stack size in word.

#define SL_TFTP_DEBUG 0U
Enable TFTP debug prints.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type definitions

Modules

sl_tftp_ack_pkt

sl_tftp_clnt

sl_tftp_clnt_opt

sl_tftp_data_pkt

sl_tftp_error_pkt

sl_tftp_oack_pkt

sl_tftp_pkt

sl_tftp_pkt_content

sl_tftp_req_pkt

Typedefs

typedef struct sl_tftp_ack_pkt	sl_tftp_ack_pkt_t TFTP ACK packet.
typedef struct sl_tftp_clnt_opt	sl_tftp_clnt_opt_t TFTP Client options The definition doesn't include the unsupported options.
typedef struct sl_tftp_clnt	sl_tftp_clnt_t TFTP Client.
typedef struct sl_tftp_data_pkt	sl_tftp_data_pkt_t TFTP data packet.
typedef struct sl_tftp_error_pkt	sl_tftp_error_pkt_t TFTP error packet.
typedef struct sl_tftp_oack_pkt	sl_tftp_oack_pkt_t TFTP OACK packet.
typedef union sl_tftp_pkt_content	sl_tftp_pkt_content_t TFTP Packet content union.
typedef struct sl_tftp_pkt	sl_tftp_pkt_t TFTP packet handler.
typedef struct sl_tftp_req_pkt	sl_tftp_req_pkt_t TFTP request packet.

Typedef Documentation

sl_tftp_ack_pkt_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_tftp_ack_pkt sl_tftp_ack_pkt_t
```

TFTP ACK packet.

sl_tftp_clnt_opt_t

```
typedef struct sl_tftp_clnt_opt sl_tftp_clnt_opt_t
```

TFTP Client options The definition doesn't include the unsupported options.

sl_tftp_clnt_t

```
typedef struct sl_tftp_clnt sl_tftp_clnt_t
```

TFTP Client.

sl_tftp_data_pkt_t

```
typedef struct sl_tftp_data_pkt sl_tftp_data_pkt_t
```

TFTP data packet.

sl_tftp_error_pkt_t

```
typedef struct sl_tftp_error_pkt sl_tftp_error_pkt_t
```

TFTP error packet.

sl_tftp_oack_pkt_t

```
typedef struct sl_tftp_oack_pkt sl_tftp_oack_pkt_t
```

TFTP OACK packet.

sl_tftp_pkt_content_t

```
typedef union sl_tftp_pkt_content sl_tftp_pkt_content_t
```

TFTP Packet content union.

sl_tftp_pkt_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_tftp_pkt sl_tftp_pkt_t
```

TFTP packet handler.

sl_tftp_req_pkt_t

```
typedef struct sl_tftp_req_pkt sl_tftp_req_pkt_t
```

TFTP request packet.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_ack_pkt

TFTP ACK packet.

Public Attributes

uint16_t	block_num Block number.

Public Attribute Documentation

block_num

```
uint16_t sl_tftp_ack_pkt::block_num
```

Block number.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_clnt

TFTP Client.

Public Attributes

void(*	data_hnd Data handler callback.
void(*	error_hnd Error handler callback.
osEventFlagsId_t	evt_flags Event flags.
const uint8_t *	ext_data External data pointer to WRQ.
uint32_t	ext_data_size External data size.
const char *	host Host address string.
sl_tftp_clnt_opt_t	options Options.
sl_tftp_pkt_t	packet Packet.
uint16_t	port Port.

Public Attribute Documentation

data_hnd

```
void(* sl_tftp_clnt::data_hnd) (struct sl_tftp_clnt *const clnt, const uint8_t *const data_ptr, const uint16_t data_size)
```

Data handler callback.

error_hnd

```
void(* sl_tftp_clnt::error_hnd) (struct sl_tftp_clnt *const clnt, const uint16_t error_code, const char *error_msg)
```

Error handler callback.

evt_flags

```
osEventFlagsId_t sl_tftp_clnt::evt_flags
```

Event flags.

Copyright © 2025 Silicon Laboratories. All rights reserved.

ext_data

```
const uint8_t* sl_tftp_clnt::ext_data
```

External data pointer to WRQ.

ext_data_size

```
uint32_t sl_tftp_clnt::ext_data_size
```

External data size.

host

```
const char* sl_tftp_clnt::host
```

Host address string.

options

```
sl_tftp_clnt_opt_t sl_tftp_clnt::options
```

Options.

packet

```
sl_tftp_pkt_t sl_tftp_clnt::packet
```

Packet.

port

```
uint16_t sl_tftp_clnt::port
```

Port.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_clnt_opt

TFTP Client options The definition doesn't include the unsupported options.

Public Attributes

uint32_t	blksize Blocksize option.
uint32_t	timeout_sec Timeout interval option.

Public Attribute Documentation

blksize

```
uint32_t sl_tftp_clnt_opt::blksize
```

Blocksize option.

timeout_sec

```
uint32_t sl_tftp_clnt_opt::timeout_sec
```

Timeout interval option.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_data_pkt

TFTP data packet.

Public Attributes

uint16_t	block_num Block number.
uint8_t *	data Data ptr.
uint16_t	data_size Data size.

Public Attribute Documentation

block_num

```
uint16_t sl_tftp_data_pkt::block_num
```

Block number.

data

```
uint8_t* sl_tftp_data_pkt::data
```

Data ptr.

data_size

```
uint16_t sl_tftp_data_pkt::data_size
```

Data size.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_error_pkt

TFTP error packet.

Public Attributes

uint16_t	errcode Error code.
const char *	errmsg Error message.

Public Attribute Documentation

errcode

```
uint16_t sl_tftp_error_pkt::errcode
```

Error code.

errmsg

```
const char* sl_tftp_error_pkt::errmsg
```

Error message.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_oack_pkt

TFTP OACK packet.

Public Attributes

const char *	opt_list Option list start ptr.
uint16_t	size Total size of option list.

Public Attribute Documentation

opt_list

```
const char* sl_tftp_oack_pkt::opt_list
```

Option list start ptr.

size

```
uint16_t sl_tftp_oack_pkt::size
```

Total size of option list.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_pkt

TFTP packet handler.

Public Attributes

sl_tftp_pkt_content_t	content Content.
uint16_t	opcode Operation code.

Public Attribute Documentation

content

```
sl_tftp_pkt_content_t sl_tftp_pkt::content
```

Content.

opcode

```
uint16_t sl_tftp_pkt::opcode
```

Operation code.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_pkt_content

TFTP Packet content union.

Public Attributes

sl_tftp_ack_pkt_t	ack ACK.
sl_tftp_data_pkt_t	data Data.
sl_tftp_error_pkt_t	error Error.
sl_tftp_oack_pkt_t	oack OACK.
sl_tftp_req_pkt_t	request Request.

Public Attribute Documentation

ack

```
sl_tftp_ack_pkt_t sl_tftp_pkt_content::ack
```

ACK.

data

```
sl_tftp_data_pkt_t sl_tftp_pkt_content::data
```

Data.

error

```
sl_tftp_error_pkt_t sl_tftp_pkt_content::error
```

Error.

oack

```
sl_tftp_oack_pkt_t sl_tftp_pkt_content::oack
```

OACK.

Copyright © 2025 Silicon Laboratories. All rights reserved.

request

```
sl_tftp_req_pkt_t sl_tftp_pkt_content::request
```

Request.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_req_pkt

TFTP request packet.

Public Attributes

const char *	filename File name.
const char *	mode Mode.

Public Attribute Documentation

filename

```
const char* sl_tftp_req_pkt::filename
```

File name.

mode

```
const char* sl_tftp_req_pkt::mode
```

Mode.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Portable interface

Portable interface

Functions

void	sl_ftp_delay_ms(const uint32_t delay_ms) Delay ms.
void	sl_ftp_dump_buff(const uint8_t *const data_ptr, const uint16_t data_size) Dump buffer.
const char *	sl_ftp_get_global_addr(void) FTP get global IP address.
void	sl_ftp_init(void) FTP Init.
bool	sl_ftp_is_network_connected(void) FTP network is connected.
int32_t	sl_ftp_socket_close(int32_t sockid) FTP Close.
void	sl_ftp_wait_for_connection(void) FTP wait for network connection.
__STATIC_INLINE void	sl_tftp_delay_ms(const uint32_t delay_ms) TFTP Delay ms.
__STATIC_INLINE void	sl_tftp_dump_buff(const uint8_t *const data_ptr, const uint16_t data_size) Define SL_TFTP_DEBUG.
__STATIC_INLINE const char *	sl_tftp_get_global_addr(void) TFTP get global IP address.
__STATIC_INLINE bool	sl_tftp_is_network_connected(void) TFTP network is connected.
__STATIC_INLINE int32_t	sl_tftp_socket_close(int32_t sockid) TFTP Close.
void	sl_tftp_udp_free_addr(void *addr) Free address.
void *	sl_tftp_udp_get_addr(const char *host, uint16_t port) Get UDP address structure.
int32_t	sl_tftp_udp_recvfrom(int32_t sockid, void *buff, uint32_t len, void *src_addr) TFTP Receive messages on a UDP socket.
int32_t	sl_tftp_udp_sendto(int32_t sockid, const void *buff, uint32_t len, const void *dest_addr) TFTP Send a message on a UDP socket.
int32_t	sl_tftp_udp_socket_create(void) TFTP udp socket create.
__STATIC_INLINE void	sl_tftp_wait_for_connection(void) TFTP wait for network connection.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Macros

#define sl_ftp_debug (format, ...)
FTP debug macro function.

#define SL_FTP_ERROR SL_FTP_INVALID_SOCKID
FTP error value.

#define SL_FTP_INVALID_SOCKID (-1)
FTP invalid socket ID.

#define SL_FTP_LOCAL_HOST_STR "::"
FTP Localhost string.

#define SL_TFTP_ERROR SL_FTP_ERROR
TFTP error value (alias)

#define SL_TFTP_INVALID_SOCKID SL_FTP_INVALID_SOCKID
TFTP invalid socket ID (alias)

#define SL_TFTP_LOCAL_HOST_STR SL_FTP_LOCAL_HOST_STR
TFTP Localhost string (alias)

Function Documentation

sl_ftp_delay_ms

```
void sl_ftp_delay_ms (const uint32_t delay_ms)
```

Delay ms.

Parameters

Type	Direction	Argument Name	Description
const uint32_t	N/A	delay_ms	Milliseconds

Portable function

sl_ftp_dump_buff

```
void sl_ftp_dump_buff (const uint8_t *const data_ptr, const uint16_t data_size)
```

Dump buffer.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *const	[in]	data_ptr	Data ptr
const uint16_t	[in]	data_size	Data size

Util function to dump buffer in hex format

sl_ftp_get_global_addr

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
const char * sl_ftp_get_global_addr (void )
```

FTP get global IP address.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Portable function **Returns**

- Global address string representation

sl_ftp_init

```
void sl_ftp_init (void )
```

FTP Init.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Initialize FTP service

sl_ftp_is_network_connected

```
bool sl_ftp_is_network_connected (void )
```

FTP network is connected.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Portable function **Returns**

- true if network is connected, otherwise false

sl_ftp_socket_close

```
int32_t sl_ftp_socket_close (int32_t sockid)
```

FTP Close.

Parameters

Type	Direction	Argument Name	Description
int32_t	N/A	sockid	Socket ID

Copyright © 2025 Silicon Laboratories. All rights reserved.

Portable function **Returns**

- int32_t Closed socket ID on success, -1 on error

sl_ftp_wait_for_connection

```
void sl_ftp_wait_for_connection (void )
```

FTP wait for network connection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Portable function

sl_tftp_delay_ms

```
__STATIC_INLINE void sl_tftp_delay_ms (const uint32_t delay_ms)
```

TFTP Delay ms.

Parameters

Type	Direction	Argument Name	Description
const uint32_t	N/A	delay_ms	Milliseconds

Alias function

sl_tftp_dump_buff

```
__STATIC_INLINE void sl_tftp_dump_buff (const uint8_t *const data_ptr, const uint16_t data_size)
```

Define SL_TFTP_DEBUG.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *const	[in]	data_ptr	Data ptr
const uint16_t	[in]	data_size	Data size

TFTP Dump buffer

Alias function

sl_tftp_get_global_addr

```
__STATIC_INLINE const char * sl_tftp_get_global_addr (void )
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

TFTP get global IP address.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Portable function **Returns**

- Global address string representation

sl_tftp_is_network_connected

```
__STATIC_INLINE bool sl_tftp_is_network_connected (void )
```

TFTP network is connected.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Alias function **Returns**

- true if network is connected, otherwise false

sl_tftp_socket_close

```
__STATIC_INLINE int32_t sl_tftp_socket_close (int32_t sockid)
```

TFTP Close.

Parameters

Type	Direction	Argument Name	Description
int32_t	N/A	sockid	Socket ID

Alias function **Returns**

- int32_t Closed socket ID on success, -1 on error

sl_tftp_udp_free_addr

```
void sl_tftp_udp_free_addr (void * addr)
```

Free address.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	addr	Address structure

Release the allocated address structure

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_udp_get_addr

> void * sl_tftp_udp_get_addr (const char * host, uint16_t port)

Get UDP address structure.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	host	Host string
uint16_t	[in]	port	Port

Allocate address structure and set host address and port **Returns**

- void * Address structure

sl_tftp_udp_recvfrom

> int32_t sl_tftp_udp_recvfrom (int32_t sockid, void * buff, uint32_t len, void * src_addr)

TFTP Receive messages on a UDP socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket id
void *	[out]	buff	destination buffer ptr
uint32_t	[in]	len	length of data to read
void *	[out]	src_addr	Source address

Receives data on a socket whether or not it is connection-oriented. **Returns**

- return the number of bytes received, or -1 if an error occurred

sl_tftp_udp_sendto

> int32_t sl_tftp_udp_sendto (int32_t sockid, const void * buff, uint32_t len, const void * dest_addr)

TFTP Send a message on a UDP socket.

Parameters

Type	Direction	Argument Name	Description
int32_t	[in]	sockid	socket ID
const void *	[in]	buff	buffer pointer to send
uint32_t	[in]	len	length of data to send
const void *	[in]	dest_addr	destination address ptr, the structure must be prepared for UDP sockets

Preferred in datagram mode (UDP). **Returns**

- On success, these calls return the number of bytes sent. On error, -1 is returned

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_tftp_udp_socket_create

```
int32_t sl_tftp_udp_socket_create (void )
```

TFTP udp socket create.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Portable function **Returns**

- int32_t Socket ID

sl_tftp_wait_for_connection

```
__STATIC_INLINE void sl_tftp_wait_for_connection (void )
```

TFTP wait for network connection.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Alias function

Copyright © 2025 Silicon Laboratories. All rights reserved.

Application Memory Statistic (Alpha)

Application Memory Statistic (Alpha)

The service systematically collects heap and stack usage data from active threads across various configurations involving FreeRTOS, MicriumOS, and different toolchains such as GCC and IAR. This critical information is then relayed to the Real-Time Transfer (RTT) console. To ensure accurate reporting, proper RTT setup is essential. The component includes a warning mechanism that provides hints about missing or incorrect configurations.

Within the component configuration, you can set the service thread parameters, the update period time, the maximum count of trackable threads, and enable heap usage statistics. A common scenario involves increasing the measurable thread stack size and the application heap size before beginning testing.

Based on the measurement results, adjustments can be made to the appropriate thread's stack and the application heap size. Stack size is calculated using the highest watermarks, and the last result always displays the maximum values.

The following RTT log demonstrates the component's operation:

```
[APP_OS_STAT][AppMain] Used/Tot: 584/1024 (58%)
[APP_OS_STAT][Ping] Used/Tot: 232/1024 (23%)
[APP_OS_STAT][WisunLedTask] Used/Tot: 192/320 (64%)
[APP_OS_STAT][AppOsStatThread] Used/Tot: 728/1024 (72%)
[APP_OS_STAT][Wi-SUN Event Task] Used/Tot: 500/2000 (25%)
[APP_OS_STAT][Wi-SUN Task] Used/Tot: 1164/2000 (58%)
[APP_OS_STAT][CoAP-Notification-Service] Used/Tot: 208/1024 (20%)
[APP_OS_STAT][CoAP-RHND-Service] Used/Tot: 240/1024 (24%)
[APP_OS_STAT][CLI instance vcom] Used/Tot: 340/1600 (21%)
[APP_OS_STAT][Kernel's Stat Task] Used/Tot: 188/1024 (18%)
[APP_OS_STAT][heap]  addr 0x2000BF80   Used/Tot: 65560/475264 (13%)
```

Modules

Configurations

void	app_os_stat_init(void) Initialize App OS statistic service.
sl_status_t	app_os_stat_register_thread(const osThreadId_t thr_id) Register thread to track.
sl_status_t	app_os_stat_remove_thread(const osThreadId_t thr_id) Remove thread from list.

app_os_stat_init

```
void app_os_stat_init (void )
```

Initialize App OS statistic service.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Copyright © 2025 Silicon Laboratories. All rights reserved.

Init thread and statistic storage

app_os_stat_register_thread

```
sl_status_t app_os_stat_register_thread (const osThreadId_t thr_id)
```

Register thread to track.

Parameters

Type	Direction	Argument Name	Description
const osThreadId_t	[in]	thr_id	Thread ID

Thread stack usage is tracked **Returns**

- sl_status_t SL_STATUS_OK on success, SL_STATUS_FAIL on error

app_os_stat_remove_thread

```
sl_status_t app_os_stat_remove_thread (const osThreadId_t thr_id)
```

Remove thread from list.

Parameters

Type	Direction	Argument Name	Description
const osThreadId_t	[in]	thr_id	Thread ID

Remove thread from tracked thread list **Returns**

- sl_status_t SL_STATUS_OK on success, SL_STATUS_FAIL on error

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define APP_OS_STAT_ASYNC_MODE_ENABLED 0U
Enable async mode.

#define APP_OS_STAT_HEAP_ENABLED 1U
Enable heap usage statistic.

#define APP_OS_STAT_MANUAL_REGISTER_THREADS 0U
Enable manual thread registering.

#define APP_OS_STAT_MAX_THREAD_CNT 20U
Maximum registerable thread count.

#define APP_OS_STAT_RTT_LOG_CHANNEL 0U
RTT Log channel for app os statistics.

#define APP_OS_STAT_STACK_SIZE_WORD 256U
App OS statistic thread stack size in word.

#define APP_OS_STAT_THREAD_PRIO osPriorityHigh
App OS statistic thread priority.

#define APP_OS_STAT_THREAD_STACK_ENABLED 1U
Enable thread stack usage statistic.

#define APP_OS_STAT_UPDATE_PERIOD_TIME_MS 1000U
App OS statistic update period time.

#define APP_OS_STAT_VERBOSE_MODE_ENABLED 0U
Enable verbose mode.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Application Status over CoAP (Alpha)

Application Status over CoAP (Alpha)

The component implements status notifications and resources over CoAP. The component can be used in any application to gather runtime information about devices. The component should be initialized with the sl_wisun_app_status_init() function. Conditional sending can be customized by implementing the sl_wisun_app_status_condition_cb() function. Server and notification parameters can be configured in the sl_wisun_app_status_config.h file or at runtime using the appropriate setters/getters.

When the application starts, the component is initialized automatically with the default values from the configuration header. The configuration interface provides options to enable or disable the expected statistics data from the nodes, such as device information, execution time statistics, and neighbor information.

Notifications can be received on any CoAP server, such as the libcoap server application. Over the prebuilt binaries, any custom library can be used to implement a CoAP server that receives the notifications from the Wi-SUN node.

CoAP resources can be accessed using the CoAP client, such as the libcoap client application to provide a solution for handling CoAP requests sent directly to the node. The application status resource is configurable through the configuration header file and can be optionally enabled or disabled at compile time.

The following JSON shows the payload of the status notification/resource:


```
{
"dev_info": {
  "hw": "BRD4270B A06 EFR32FG25B222F1920IM56",
  "sw": "Wi-SUN Ping Application v2.0.0 (stack: v2.3.0 FFN)",
  "mac": "60:A4:23:FF:FE:37:A7:57",
  "ip": "fd12:3456::62a4:23ff:fe37:a757"
},
"stat": {
  "run": "000-02:47:37",
  "conn_cnt": 2,
  "conn": "000-00:30:35",
  "tot_conn": "000-02:43:23",
  "disconn": "000-00:26:26",
  "tot_disconn": "000-00:04:26",
  "available": "97.35%"
},
"nb_info": {
  "ip":["t","lt","txc","txf","txms","txmsf","rpl","etx","rslo","rsli"],
  "fd12:3456::62a4:23ff:fe37:aed9":[0,2191,1352,0,0,0,128,134,133,143],
  "fd12:3456::32fb:10ff:feed:414e":[2,2080,29,0,0,0,65535,141,119,132],
  "fd12:3456::62a4:23ff:fe37:aee3":[2,545,14,0,0,0,65535,160,101,114]
}
}
```


Modules

Configurations

Functions

bool sl_wisun_app_status_condition_cb(const sl_wisun_coap_notify_t *notify)

Application Status notification callback.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sockaddr_in6_t * sl_wisun_app_status_get_remote_address(void)
Application Status notification remote address and port getter.

uint32_t sl_wisun_app_status_get_schedule_time_ms(void)
Application Status notification schedule time getter.

void sl_wisun_app_status_init(void)
Init Application Status notification content.

sl_status_t sl_wisun_app_status_set_remote_address(const char *remote_address, const uint16_t port)
Application Status notification remote address and port setter.

void sl_wisun_app_status_set_schedule_time_ms(const uint32_t new_schedule_time_ms)
Application Status notification schedule time setter.

Function Documentation

sl_wisun_app_status_condition_cb

> bool sl_wisun_app_status_condition_cb (const sl_wisun_coap_notify_t * notify)

Application Status notification callback.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_coap_notify_t *	N/A	notify	Notification context.

The function provides an implementable callback to create custom condition check to send notification. It's a weak implementation, by default it returns true always. **Returns**

- bool The result of the condition check.

sl_wisun_app_status_get_remote_address

> sockaddr_in6_t * sl_wisun_app_status_get_remote_address (void)

Application Status notification remote address and port getter.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The function provides a getter function for the address and port parameters that are static for the file **Returns**

- sockaddr_in6_t The pointer to the address setting structure

sl_wisun_app_status_get_schedule_time_ms

> uint32_t sl_wisun_app_status_get_schedule_time_ms (void)

Application Status notification schedule time getter.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
void	N/A		

The function provides a getter function for the schedule time parameters that are static for the file **Returns**

- uint32_t The schedule time in ms

sl_wisun_app_status_init

```
void sl_wisun_app_status_init (void )
```

Init Application Status notification content.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Init notification content and callbacks.

sl_wisun_app_status_set_remote_address

```
sl_status_t sl_wisun_app_status_set_remote_address (const char * remote_address, const uint16_t port)
```

Application Status notification remote address and port setter.

Parameters

Type	Direction	Argument Name	Description
const char *	N/A	remote_address	remote address to be selected
const uint16_t	N/A	port	remote port to be selected

The function provides a setter function for the address and port parameters that are static for the file **Returns**

- sl_status_t SL_STATUS_OK if setting was success or the proper sl_status enum of the error

sl_wisun_app_status_set_schedule_time_ms

```
void sl_wisun_app_status_set_schedule_time_ms (const uint32_t new_schedule_time_ms)
```

Application Status notification schedule time setter.

Parameters

Type	Direction	Argument Name	Description
const uint32_t	N/A	new_schedule_time_ms	The schedule time in ms

The function provides a setter function for the schedule time parameters that are static for the file

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configurations

Configurations

Macros

#define SL_WISUN_APP_STATUS_COAP_RESOURCE_ENABLE 1U
Enable CoAP resource.

#define SL_WISUN_APP_STATUS_DEFAULT_NOTIFICATION_ID "app-status"
Default notification ID string.

#define SL_WISUN_APP_STATUS_DEFAULT_REMOTE_ADDR "2001:db8::1"
Default remote address to send status report.

#define SL_WISUN_APP_STATUS_DEFAULT_REMOTE_PORT 5683U
Default port of the remote host to send status report.

#define SL_WISUN_APP_STATUS_DEFAULT_SCHEDULE_TIME_MS 10000UL
Default schedule time to send status report.

#define SL_WISUN_APP_STATUS_DEFAULT_URI_PATH "/app/status"
Default URI Path to use in status report packet.

#define SL_WISUN_APP_STATUS_DEVICE_INFO_ENABLE 1U
Enable execution time.

#define SL_WISUN_APP_STATUS_EXECUTION_TIME_ENABLE 1U
Enable execution time.

#define SL_WISUN_APP_STATUS_NEIGHBOR_INFO_ENABLE 1U
Enable Neighbor info.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Keychain

Keychain

Wi-SUN Keychain API is an interface for retrieving device credentials and trusted CA certificates from a set of keychains.

Supported keychain types:

- **built-in** Built-in keychain is created during compilation time and cannot be changed. The application may add its own credentials via template contributions using Silicon Labs Configurator. Device private keys are stored unencrypted.
- **NVM** NVM keychain is dynamically provisioned either during device production or while on the field. Device private keys are stored in PSA ITS, device certificates and trusted CA certificates are stored in NVM3. Depending on the device capabilities, PSA ITS may be encrypted.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
#include "sl_memory_manager.h"
#include "sl_wisun_api.h"
#include "sl_wisun_keychain.h"

uint8_t trustedca_count;
sl_wisun_keychain_entry_t *trustedca;
sl_wisun_keychain_credential_t *credential;
uint16_t certificate_options;

// Retrieve a device credential
credential = sl_wisun_keychain_get_credential(SL_WISUN_KEYCHAIN_AUTOMATIC, 0);

// Configure the device certificate
sl_wisun_set_device_certificate(SL_WISUN_CERTIFICATE_OPTION_NONE,
                                credential->certificate.data_length,
                                credential->certificate.data);

// Configure the device private key
if (credential->pk.type == SL_WISUN_KEYCHAIN_KEY_TYPE_PLAINTEXT) {
  sl_wisun_set_device_private_key(SL_WISUN_PRIVATE_KEY_OPTION_NONE,
                                  credential->pk.u.plaintext.data_length,
                                  credential->pk.u.plaintext.data);
} else {
  sl_wisun_set_device_private_key_id(credential->pk.u.key_id);
}

// Free the device credential, ownership was transferred
sl_free(credential);

// Enumerate the trusted CA certificates
trustedca_count = sl_wisun_keychain_get_trustedca_count();

// Loop through available trusted CA certificates
certificate_options = SL_WISUN_CERTIFICATE_OPTION_NONE;
for (uint8_t idx = 0; idx < trustedca_count; ++idx) {
  // Retrieve a trusted CA certificate
  trustedca = sl_wisun_keychain_get_trustedca(idx);

  // Configure the trusted CA certificate
  sl_wisun_set_trusted_certificate(certificate_options,
                                   trustedca->data_length,
                                   trustedca->data);

  // Free the trusted CA certificate, ownership was transferred
  sl_free(trustedca);

  // The trusted CA certificates after the first one must be
  // configured with SL_WISUN_CERTIFICATE_OPTION_APPEND.
  certificate_options |= SL_WISUN_CERTIFICATE_OPTION_APPEND;
}
```

It's possible to utilize Silicon Labs Configurator to add credentials/trusted CAs to the built-in keychain during compile time through template contributions.

- wisun_keychain_include::path is the name of the header file that contains the uint8_t array declarations or definitions.
- wisun_keychain_trustedca::certificate is the name of the uint8_t array for a trusted CA certificate. This variable can be defined multiple times.
- wisun_keychain_credential::certificate is the name of the uint8_t array for the device certificate.
- wisun_keychain_credential::key is the name of the uint8_t array for the device private key.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
template_contribution:
  - name: wisun_keychain_include
    value:
      path: certificate_example.h
  - name: wisun_keychain_trustedca
    value:
      certificate: EXAMPLE_ROOT_CERT
  - name: wisun_keychain_trustedca
    value:
      certificate: EXAMPLE_MCA_CERT
  - name: wisun_keychain_trustedca
    value:
      certificate: EXAMPLE_MICA_CERT
  - name: wisun_keychain_credential
    value:
      certificate: EXAMPLE_DEVICE_CERT
      key: EXAMPLE_DEVICE_KEY
```

In certificate_example.h

```
const uint8_t EXAMPLE_ROOT_CERT[] = {
  ...
};
const uint8_t EXAMPLE_MCA_CERT[] = {
  ...
};
const uint8_t EXAMPLE_MICA_CERT[] = {
  ...
};
const uint8_t EXAMPLE_DEVICE_CERT[] = {
  ...
};
const uint8_t EXAMPLE_DEVICE_KEY[] = {
  ...
};
```

Modules

sl_wisun_keychain_credential_t

sl_wisun_keychain_entry_t

sl_wisun_keychain_key_t

Enumerations

enum sl_wisun_keychain_key_type_t {

SL_WISUN_KEYCHAIN_KEY_TYPE_PLAINTEXT
SL_WISUN_KEYCHAIN_KEY_TYPE_ID

}

Enumeration for private key type.

enum sl_wisun_keychain_t {

SL_WISUN_KEYCHAIN_AUTOMATIC
SL_WISUN_KEYCHAIN_BUILTIN
SL_WISUN_KEYCHAIN_NVM

}

Enumeration for keychain.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Functions

sl_wisun_keychain_credential_t *	sl_wisun_keychain_get_credential(sl_wisun_keychain_t keychain, uint8_t index) Retrieve a device credential.
sl_wisun_keychain_entry_t *	sl_wisun_keychain_get_trustedca(uint8_t index) Retrieve a trusted CA certificate.
uint8_t	sl_wisun_keychain_get_trustedca_count() Retrieve the number of trusted CA certificates.

Enumeration Documentation

sl_wisun_keychain_key_type_t

> sl_wisun_keychain_key_type_t

Enumeration for private key type.

Enumerator	
SL_WISUN_KEYCHAIN_KEY_TYPE_PLAINTEXT	Private key is stored in plaintext.
SL_WISUN_KEYCHAIN_KEY_TYPE_ID	Private key is stored in PSA ITS.

sl_wisun_keychain_t

> sl_wisun_keychain_t

Enumeration for keychain.

Enumerator	
SL_WISUN_KEYCHAIN_AUTOMATIC	Automatic keychain selection.
SL_WISUN_KEYCHAIN_BUILTIN	Built-in keychain.
SL_WISUN_KEYCHAIN_NVM	NVM keychain.

Function Documentation

sl_wisun_keychain_get_credential

> sl_wisun_keychain_credential_t * sl_wisun_keychain_get_credential (sl_wisun_keychain_t keychain, uint8_t index)

Retrieve a device credential.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
sl_wisun_keychain_t	[in]	keychain	Keychain to use • SL_WISUN_KEYCHAIN_AUTOMATIC: Use NVM keychain if available, built-in otherwise • SL_WISUN_KEYCHAIN_BUILTIN: Use built-in keychain • SL_WISUN_KEYCHAIN_NVM: Use NVM keychain
uint8_t	[in]	index	Built-in credential index (0...max)

Returns

- Pointer to the device credential on success, NULL otherwise.

This function retrieves a device credential from the given keychain. Calling function is responsible for freeing the returned entry.

sl_wisun_keychain_get_trustedca

```
sl_wisun_keychain_entry_t * sl_wisun_keychain_get_trustedca (uint8_t index)
```

Retrieve a trusted CA certificate.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Trusted CA certificate index to retrieve (0...max)

Returns

- Pointer to the trusted CA certificate on success, NULL otherwise.

This function retrieves a trusted CA certificate from one of the keychains. The index must be below the value returned by sl_wisun_keychain_get_trustedca_count(). Calling function is responsible for freeing the returned entry.

sl_wisun_keychain_get_trustedca_count

```
uint8_t sl_wisun_keychain_get_trustedca_count ()
```

Retrieve the number of trusted CA certificates.

Returns

- Number of trusted CA certificates, 0 if none.

This function returns the number of trusted CA certificates in all available keychains.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_keychain_credential_t

Data structure for a device credential.

Public Attributes

sl_wisun_keychain_entry_t	certificate Device certificate.
sl_wisun_keychain_key_t	pk Device private key.

Public Attribute Documentation

certificate

```
sl_wisun_keychain_entry_t sl_wisun_keychain_credential_t::certificate
```

Device certificate.

pk

```
sl_wisun_keychain_key_t sl_wisun_keychain_credential_t::pk
```

Device private key.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_keychain_entry_t

Data structure for a keychain entry.

Public Attributes

const void *	data Pointer to data.
size_t	data_length Amount of data in bytes.
sl_wisun_keychain_t	keychain Keychain were the entry is stored.

Public Attribute Documentation

data

```
const void* sl_wisun_keychain_entry_t::data
```

Pointer to data.

When freeing a keychain entry, this pointer must not be freed.

data_length

```
size_t sl_wisun_keychain_entry_t::data_length
```

Amount of data in bytes.

keychain

```
sl_wisun_keychain_t sl_wisun_keychain_entry_t::keychain
```

Keychain were the entry is stored.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_keychain_key_t

Data structure for a device private key.

Public Attributes

uint32_t	key_id Private key when type is SL_WISUN_KEYCHAIN_KEY_TYPE_ID.
sl_wisun_keychain_t	keychain Keychain were the private keý is stored.
sl_wisun_keychain_entry_t	plaintext Private key when type is SL_WISUN_KEYCHAIN_KEY_TYPE_PLAINTEXT.
sl_wisun_keychain_key_type_t	type Type of the private key.
union sl_wisun_keychain_key_t::@4	u Private key.

Public Attribute Documentation

key_id

```
uint32_t sl_wisun_keychain_key_t::key_id
```

Private key when type is SL_WISUN_KEYCHAIN_KEY_TYPE_ID.

keychain

```
sl_wisun_keychain_t sl_wisun_keychain_key_t::keychain
```

Keychain were the private keý is stored.

plaintext

```
sl_wisun_keychain_entry_t sl_wisun_keychain_key_t::plaintext
```

Private key when type is SL_WISUN_KEYCHAIN_KEY_TYPE_PLAINTEXT.

type

```
sl_wisun_keychain_key_type_t sl_wisun_keychain_key_t::type
```

Type of the private key.

Copyright © 2025 Silicon Laboratories. All rights reserved.

u

```
union sl_wisun_keychain_key_t::@4 sl_wisun_keychain_key_t::u
```

Private key.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

RF Test

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_filter(const uint8_t filter[SL_WISUN_FILTER_BITFIELD_SIZE])
Set the trace filter.

sl_status_t sl_wisun_set_trace_level(uint8_t group_count, const sl_wisun_trace_group_config_t *trace_config)
Set the trace level.

sl_status_t sl_wisun_trace_debug(const char *fmt,...)
Generate application logs using Wi-SUN stack trace mechanism with DEBUG level.

sl_status_t sl_wisun_trace_error(const char *fmt,...)
Generate application logs using Wi-SUN stack trace mechanism with ERROR level.

sl_status_t sl_wisun_trace_info(const char *fmt,...)
Generate application logs using Wi-SUN stack trace mechanism with INFO level.

sl_status_t sl_wisun_trace_warn(const char *fmt,...)
Generate application logs using Wi-SUN stack trace mechanism with WARN level.

Function Documentation

sl_wisun_set_trace_filter

```
sl_status_t sl_wisun_set_trace_filter (const uint8_t filter)
```

Set the trace filter.

Parameters

Type	Direction	Argument Name	Description
const uint8_t	[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

sl_wisun_set_trace_level

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_status_t sl_wisun_set_trace_level (uint8_t group_count, const sl_wisun_trace_group_config_t * trace_config)

Set the trace level.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	group_count	Number of groups to configure. If 0, enable all levels for all groups. Maximum SL_WISUN_TRACE_GROUP_COUNT.
const sl_wisun_trace_group_config_t *	[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

sl_wisun_trace_debug

sl_status_t sl_wisun_trace_debug (const char * fmt, ...)

Generate application logs using Wi-SUN stack trace mechanism with DEBUG level.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	fmt	Format string (same syntax as printf)
...	[in]		Variable arguments related to fmt

Returns

- SL_STATUS_NOT_AVAILABLE if using the release library, SL_STATUS_OK otherwise

sl_wisun_trace_error

sl_status_t sl_wisun_trace_error (const char * fmt, ...)

Generate application logs using Wi-SUN stack trace mechanism with ERROR level.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	fmt	Format string (same syntax as printf)
...	[in]		Variable arguments related to fmt

Returns

- SL_STATUS_NOT_AVAILABLE if using the release library, SL_STATUS_OK otherwise

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_trace_info

```
sl_status_t sl_wisun_trace_info (const char * fmt, ... )
```

Generate application logs using Wi-SUN stack trace mechanism with INFO level.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	fmt	Format string (same syntax as printf)
...	[in]		Variable arguments related to fmt

Returns

- SL_STATUS_NOT_AVAILABLE if using the release library, SL_STATUS_OK otherwise

sl_wisun_trace_warn

```
sl_status_t sl_wisun_trace_warn (const char * fmt, ... )
```

Generate application logs using Wi-SUN stack trace mechanism with WARN level.

Parameters

Type	Direction	Argument Name	Description
const char *	[in]	fmt	Format string (same syntax as printf)
...	[in]		Variable arguments related to fmt

Returns

- SL_STATUS_NOT_AVAILABLE if using the release library, SL_STATUS_OK otherwise

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

bool	sl_wisun_is_running_rf_test() Return the current status of the RF test plugin.
sl_status_t	sl_wisun_set_test_tx_power(int8_t tx_power) Set transmit power.
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_start_tone(uint16_t channel) Start transmitting an unmodulated tone.
sl_status_t	sl_wisun_stop_stream() Stop a previously started stream of characters.
sl_status_t	sl_wisun_stop_tone() Stop a previously started tone.

Function Documentation

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.

sl_wisun_set_test_tx_power

```
sl_status_t sl_wisun_set_test_tx_power (int8_t tx_power)
```

Set transmit power.

Parameters

Type	Direction	Argument Name	Description
int8_t	[in]	tx_power	Transmit power in units of dBm, can be negative.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Returns

- always SL_STATUS_OK

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

Type	Direction	Argument Name	Description
uint16_t	[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 sl_rail_start_tx_stream() for more information.

sl_wisun_start_tone

```
sl_status_t sl_wisun_start_tone (uint16_t channel)
```

Start transmitting an unmodulated tone.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[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 sl_rail_start_tx_stream() for more information.

sl_wisun_stop_stream

```
sl_status_t sl_wisun_stop_stream ( channel)
```

Stop a previously started stream of characters.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
	[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 sl_rail_stop_tx_stream() for more information.

sl_wisun_stop_tone

```
sl_status_t sl_wisun_stop_tone ( channel)
```

Stop a previously started tone.

Parameters

Type	Direction	Argument Name	Description
	[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 sl_rail_stop_tx_stream() for more information.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Wi-SUN API availability

Wi-SUN Regional Regulation API

Border Router API

Socket API

Select API

Callbacks

void	sl_wisun_on_event(sl_wisun_evt_t *evt) Callback handler for a single event.

Functions

sl_status_t	sl_wisun_accept_direct_connect_link(in6_addr_t *link_local_ipv6) Accept a Direct Connect link.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_clear_credential_cache()
Clear the credential cache.

sl_status_t sl_wisun_config_concurrent_detection(bool enable_tx, uint8_t reserved)
Configure concurrent detection.

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_config_neighbor_table(uint8_t max_child_count, uint8_t max_neighbor_count, uint16_t max_security_neighbor_count)
Configure neighbor table size.

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_disconnect()
Disconnect from the Wi-SUN network.

sl_status_t sl_wisun_get_excluded_channel_mask(sl_wisun_channel_mask_type_t type, sl_wisun_channel_mask_t *channel_mask, uint32_t *channel_count)
Get the mask of channels excluded from channel plan.

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_get_join_state(sl_wisun_join_state_t *join_state)
Get the current join state.

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_get_neighbor_count(uint8_t *neighbor_count)
Get the number of RPL neighbors (parents and children).

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_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_network_info(sl_wisun_network_info_t *network_info)
Get Wi-SUN network information.

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_rpl_info(sl_wisun_rpl_info_t *rpl_info)
Get RPL information.

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_get_statistics(sl_wisun_statistics_type_t statistics_type, sl_wisun_statistics_t *statistics)
Read a set of statistics.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_status_t sl_wisun_reset_regulation_duty_cycle(void)
Reset all the the past hour transmissions counters.

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_set_advert_fragment_duration(uint32_t fragment_duration_ms)
Set the async transmission fragmentation parameters.

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_set_connection_parameters(const sl_wisun_connection_params_t *params)
Configure the FFN parameter set.

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 device private key used to authenticate to the authentication server.

sl_status_t sl_wisun_set_device_private_key_id(uint32_t key_id)
Set the device private key used to authenticate to the authentication server.

sl_status_t sl_wisun_set_device_type(sl_wisun_device_type_t device_type)
Set the device type.

sl_status_t sl_wisun_set_direct_connect_pmk(const uint32_t pmk_key_id)
Set the Pairwise Master Key (PMK) to use for Direct Connect.

sl_status_t sl_wisun_set_direct_connect_state(bool is_enabled)
Enable/disable Direct Connect.

sl_status_t sl_wisun_set_eap_identity(uint8_t identity_length, const uint8_t *identity)
Set EAP identity for authentication.

sl_status_t sl_wisun_set_leaf(bool is_leaf)
Set the RPL leaf mode.

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)
Configure LFN parenting support.

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_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_neighbor_table_size(uint8_t size) SL_DEPRECATED_API_SDK_2025_6
Set neighbor table size.

sl_status_t sl_wisun_set_phy_sensitivity(uint8_t phy_mode_id, int16_t sensitivity_dbm)
Set the radio sensitivity for the given PHY.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_set_preferred_pan(uint16_t pan_id)
Set the preferred PAN for this device.

sl_status_t sl_wisun_set_pti_state(bool pti_state)
Set the PTI state.

sl_status_t sl_wisun_set_regulation(sl_wisun_regulation_t regulation)
Set the regional regulation.

sl_status_t sl_wisun_set_regulation_parameters(const sl_wisun_regulation_params_t *params)
Configure the regional regulation parameter set.

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_security_state(uint32_t security_state)
Set the connection security state.

sl_status_t sl_wisun_set_trusted_certificate(uint16_t certificate_options, uint16_t certificate_length, const uint8_t *certificate)
Set a trusted CA certificate used to validate the authentication server certificate.

sl_status_t sl_wisun_set_tx_power(int8_t tx_power) SL_DEPRECATED_API_SDK_2024_6
Set the maximum TX power.

sl_status_t sl_wisun_set_tx_power_ddbm(int16_t tx_power_ddbm)
Set the maximum TX power.

sl_status_t sl_wisun_set_unicast_settings(uint8_t dwell_interval_ms)
Set unicast settings.

sl_status_t sl_wisun_trigger_frame(sl_wisun_frame_type_t frame_type)
Trigger a transmission of a periodic frame (FAN Discovery, RPL).

Callbacks Documentation

sl_wisun_on_event

> void sl_wisun_on_event (sl_wisun_evt_t * evt)

Callback handler for a single event.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_evt_t *	[out]	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

Function Documentation

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_accept_direct_connect_link

```
sl_status_t sl_wisun_accept_direct_connect_link (in6_addr_t * link_local_ipv6)
```

Accept a Direct Connect link.

Parameters

Type	Direction	Argument Name	Description
in6_addr_t *	[in]	link_local_ipv6	Link-Local IPv6 address of the client

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
const sl_wisun_mac_address_t *	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_config_concurrent_detection

```
sl_status_t sl_wisun_config_concurrent_detection (bool enable_tx, uint8_t reserved)
```

Configure concurrent detection.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable_tx	TX state • **true**: TX is allowed with an alternate PHY • **false**: TX is not allowed with an alternate PHY
uint8_t	[in]	reserved	Reserved for future use, set to zero.

Returns

- SL_STATUS_OK if the operation is successful, an error code otherwise.

This function configures whether an alternate PHY can be used concurrently with the base PHY. The alternate PHY must be enabled in the radio configuration to utilize this feature. When enabled, the alternate PHY is always accepted on receive. When TX state is set to true, the node will transmit to nodes that support the feature using the alternate PHY, instead of using the standard mode switch. TX is disabled by default.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
uint8_t	[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.
uint8_t	[in]	phy_mode_id	PhyModeId to use when mode is set to SL_WISUN_MODE_SWITCH_ENABLED, ignored otherwise.
const sl_wisun_mac_address_t *	[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.
bool	[in]	reserved	Reserved for future use, set to false.

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_config_neighbor_table

```
sl_status_t sl_wisun_config_neighbor_table (uint8_t max_child_count, uint8_t max_neighbor_count, uint16_t max_security_neighbor_count)
```

Configure neighbor table size.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	max_child_count	Maximum number of RPL children Increasing this parameter means a higher number of potential neighbors while lowering it means reduced RAM consumption. The default value is 22.
uint8_t	[in]	max_neighbor_count	Maximum number of neighbors including children, parent, and temporary neighbors The default value is 32
uint16_t	[in]	max_security_neighbor_count	Maximum number of neigbors in the security table. Entries in the security table are removed on key expirations. Default value is 300

Returns

- SL_STATUS_OK if successful, an error code otherwise

max_neighbor_count must be greater than max_child_count. max_neighbor_count - max_child_count represents the available neighbors for RPL parents and temporary neighbors (neighbors not yet registered, parent candidate or neighbors sending multicast). max_security_neighbor_count must be greater or equal than max_neighbor_count Each entry in the neighbor table consumes about 450 bytes of RAM. Each entry in the security neighbor table consumes about 50 bytes of RAM. Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
const sl_wisun_mac_address_t *	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_disconnect

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Available in libraries: Full, FFN, LFN (see Wi-SUN API availability)

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, uint32_t * channel_count)

Get the mask of channels excluded from channel plan.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_channel_mask_type_t	[in]	type	Type of channel mask to retrieve
sl_wisun_channel_mask_t *	[out]	channel_mask	Pointer where channel mask is written
uint32_t *	[out]	channel_count	Number of channels in the mask

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
sl_wisun_ip_address_type_t	[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

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
in6_addr_t *	[out]	address	IP address to read

Returns

- SL_STATUS_OK if successful, an error code otherwise.

Available in libraries: Full, FFN, LFN (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
sl_wisun_join_state_t *	[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.

Available in libraries: Full, FFN, LFN (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
sl_wisun_mac_address_t *	[out]	address	MAC address

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
uint8_t *	[out]	neighbor_count	Number of neighbors

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
const sl_wisun_mac_address_t *	[in]	neighbor_mac_address	Pointer to neighbor MAC address
sl_wisun_neighbor_info_t *	[out]	neighbor_info	Pointer to where the read information is stored

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
uint8_t *	[inout]	neighbor_count	Maximum number of neighbors to read on input, number of neighbors read on output
sl_wisun_mac_address_t *	[out]	neighbor_mac_addresses	Pointer to memory where to store neighbor MAC addresses

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_get_network_info

> sl_status_t sl_wisun_get_network_info (sl_wisun_network_info_t * network_info)

Get Wi-SUN network information.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_network_info_t *	[out]	network_info	Pointer where network information is written

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
uint8_t *	[out]	phy_mode_id_count	Number of PhyModeId retrieved
uint8_t *	[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.
uint8_t *	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_get_rpl_info

```
sl_status_t sl_wisun_get_rpl_info (sl_wisun_rpl_info_t * rpl_info)
```

Get RPL information.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_rpl_info_t *	[out]	rpl_info	Pointer where RPL information is written

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, BR (see Wi-SUN API availability)

sl_wisun_get_stack_version

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Type	Direction	Argument Name	Description
uint8_t *	[out]	major	Wi-SUN stack version major
uint8_t *	[out]	minor	Wi-SUN stack version minor
uint8_t *	[out]	patch	Wi-SUN stack version patch
uint16_t *	[out]	build	Build number, set to 0 in public versions

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
sl_wisun_statistics_type_t	[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
sl_wisun_statistics_t *	[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().

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_join

Copyright © 2025 Silicon Laboratories. All rights reserved.

> 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

Type	Direction	Argument Name	Description
const uint8_t *	[in]	name	Name of the Wi-SUN network as a zero-terminated string
sl_wisun_phy_config_t *	[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.

Available in libraries: Full, FFN, LFN (see Wi-SUN API availability)

sl_wisun_reset_regulation_duty_cycle

> sl_status_t sl_wisun_reset_regulation_duty_cycle (void)

Reset all the the past hour transmissions counters.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
sl_wisun_statistics_type_t	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
uint32_t	[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.

Available in libraries: Full, FFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
const sl_wisun_channel_mask_t *	[in]	channel_mask	Mask of operating channels

Returns

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Warnings

- By comparison to the Wi-SUN FAN specification, the channel mask logic is inverted. The specification references a mask of excluded channels.

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

Type	Direction	Argument Name	Description
const sl_wisun_channel_mask_t *	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
const sl_wisun_connection_params_t *	[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). Small profile will be used by default for all missing configurations. The function must be called before initiating a connection.

Available in libraries: Full, FFN (see Wi-SUN API availability)

sl_wisun_set_device_certificate

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Type	Direction	Argument Name	Description
uint16_t	[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 corresponding private key
uint16_t	[in]	certificate_length	Size of the certificate data
const uint8_t *	[in]	certificate	Pointer to the certificate data

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the device certificate used during the authentication process. The corresponding device private key must be set using sl_wisun_set_device_private_key() or sl_wisun_set_device_private_key_id(). The supported certificate formats depend on application's Mbed TLS configuration. The default configuration supports PEM and DER formats.

Available in libraries: Full, FFN, LFN (see Wi-SUN API availability)

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 device private key used to authenticate to the authentication server.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[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
uint16_t	[in]	key_length	Size of the private key data
const uint8_t *	[in]	key	Pointer to the private key data

Returns

- SL_STATUS_OK if successful, an error code otherwise

Copyright © 2025 Silicon Laboratories. All rights reserved.

This function sets the device private key used during the authentication process. The corresponding device certificate must be set using sl_wisun_set_device_certificate(). The supported key formats depend on application's Mbed TLS configuration. The default configuration supports unencrypted PKCS#8 and SEC1 keys in PEM and DER formats.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_set_device_private_key_id

```
sl_status_t sl_wisun_set_device_private_key_id (uint32_t key_id)
```

Set the device private key used to authenticate to the authentication server.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[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 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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
sl_wisun_device_type_t	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Note

- This is a no-op with BR, FFN and LFN libraries.

sl_wisun_set_direct_connect_pmk

```
sl_status_t sl_wisun_set_direct_connect_pmk (const uint32_t pmk_key_id)
```

Set the Pairwise Master Key (PMK) to use for Direct Connect.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
const uint32_t	[in]	pmk_key_id	The Pairwise Master Key (PMK) ID to use.

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the Direct Connect PMK using a key identifier, referencing a key stored in PSA cryptography module. The stored key must have correct PSA key attributes, see the Wi-SUN FAN Security Concepts and Design Considerations document for details.

Available in libraries: Full, FFN (see Wi-SUN API availability)

sl_wisun_set_direct_connect_state

```
sl_status_t sl_wisun_set_direct_connect_state (bool is_enabled)
```

Enable/disable Direct Connect.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	is_enabled	Direct Connect status • **true**: Direct Connect is enabled • **false**: Direct Connect is disabled

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN (see Wi-SUN API availability)

Note

- Enabling this feature may increase connection time.

sl_wisun_set_eap_identity

```
sl_status_t sl_wisun_set_eap_identity (uint8_t identity_length, const uint8_t * identity)
```

Set EAP identity for authentication.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	identity_length	Length of EAP identity in bytes
const uint8_t *	[in]	identity	EAP identity to use

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the identity used in the EAP Identity Response during EAP authentication. The function must be called before initiating a connection. No checks are performed on the identity, the caller is responsible for formatting it correctly.

Copyright © 2025 Silicon Laboratories. All rights reserved.

In particular, RFC3748 requires that the identity must not be NULL terminated. If not set, the identity defaults to "Anonymous".

Available in libraries: Full, FFN, LFN (see Wi-SUN API availability)

sl_wisun_set_leaf

```
sl_status_t sl_wisun_set_leaf (bool is_leaf)
```

Set the RPL leaf mode.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	is_leaf	RPL leaf mode • **true**: RPL leaf mode is enabled • **false**: RPL leaf mode is disabled

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function configures RPL leaf mode on FFNs. When enabled, it prevents the FFN from parenting other nodes (FFN or LFN).

Available in libraries: Full, FFN (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
const sl_wisun_lfn_params_t *	[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). Test profile will be used by default for all missing configurations. The function must be called before initiating a connection.

Available in libraries: Full, LFN (see Wi-SUN API availability)

sl_wisun_set_lfn_support

```
sl_status_t sl_wisun_set_lfn_support (uint8_t lfn_limit)
```

Configure LFN parenting support.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	lfn_limit	Maximum number of LFN children • 0: LFN parenting is disabled in the node • **> 0**: Maximum number of LFN children the node can parent

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the maximum number of LFN children this node can parent. Set sl_wisun_config_neighbor_table accordingly.

Available in libraries: Full, FFN (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
const sl_wisun_mac_address_t *	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
uint8_t	[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.
uint8_t	[in]	phy_mode_id	PhyModeId to use when mode is set to SL_WISUN_MODE_SWITCH_ENABLED, ignored otherwise.
const sl_wisun_mac_address_t *	[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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Returns

- SL_STATUS_OK if successful, an error code otherwise

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_set_neighbor_table_size

```
sl_status_t sl_wisun_set_neighbor_table_size (uint8_t size)
```

Set neighbor table size.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	size	Size of the neighbor table

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the limit of regular neighbors supported by the node [1, 245], without considering temporary or RPL parents. Increasing this parameter means a higher number of potential neighbors while lowering it means reduced RAM consumption. The default value is 22.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack, use sl_wisun_config_neighbor_table() instead.

sl_wisun_set_phy_sensitivity

```
sl_status_t sl_wisun_set_phy_sensitivity (uint8_t phy_mode_id, int16_t sensitivity_dbm)
```

Set the radio sensitivity for the given PHY.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	phy_mode_id	PHY mode ID
int16_t	[in]	sensitivity_dbm	Sensitivity in dBm

Returns

- SL_STATUS_OK if successful, SL_STATUS_INVALID_PARAMETER if the PHY mode ID is invalid, SL_STATUS_FULL if you a have already set SL_WISUN_MAX_PHY_MODE_ID_COUNT different entries

This function sets the radio sensitivity for RX state. It allows to fine tune it according to the user's hardware and needs. The stack possesses a set of default values for each PHY mode, but the user can override them with this function. User's modifications generated by this API are not saved in the NVM.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Warnings

- Changing the RX sensitivity can have a significant impact on the network's behavior and performance and should be done with caution. The stack follows the Wi-SUN specification, and as such, takes a margin around the given sensitivity

Copyright © 2025 Silicon Laboratories. All rights reserved.

(hysteresis). It is strongly advised to only use this API when using a Low-noise amplifier with an FEM.

sl_wisun_set_pom_ie

```
sl_status_t sl_wisun_set_pom_ie (uint8_t phy_mode_id_count, uint8_t phy_mode_ids, uint8_t is_mdr_command_capable)
```

Set the POM-IE configuration.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	phy_mode_id_count	Number of PhyModeId to configure
uint8_t	[in]	phy_mode_ids	List of phy_mode_id_count PhyModeId. It must not contain the base operating mode.
uint8_t	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_set_preferred_pan

```
sl_status_t sl_wisun_set_preferred_pan (uint16_t pan_id)
```

Set the preferred PAN for this device.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	pan_id	Preferred PAN ID. Set to 0xFFFF to disable.

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets a preferred PAN ID that the device will always select during connection if available, or seek to roam to while connected to another PAN.

Note

- Available in libraries: Full, FFN (see Wi-SUN API availability)

sl_wisun_set_pti_state

```
sl_status_t sl_wisun_set_pti_state (bool pti_state)
```

Set the PTI state.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
bool	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_set_regulation

> sl_status_t sl_wisun_set_regulation (sl_wisun_regulation_t regulation)

Set the regional regulation.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_regulation_t	[in]	regulation	Regional regulation

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function applies the appropriate parameter set from Predefined regional regulation parameter sets to be used for upcoming 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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Note

- This function uses a copy of the regulation parameter sets, modifying a predefined set in sl_wisun_regulation_api.h will not affect the regulation set used by this function. To change the regulation parameters, use sl_wisun_set_regulation_parameters().

sl_wisun_set_regulation_parameters

> sl_status_t sl_wisun_set_regulation_parameters (const sl_wisun_regulation_params_t * params)

Configure the regional regulation parameter set.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_regulation_params_t *	[in]	params	Parameter set to use

Returns

- SL_STATUS_OK if successful, an error code otherwise

Copyright © 2025 Silicon Laboratories. All rights reserved.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

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

Type	Direction	Argument Name	Description
int8_t	[in]	warning_threshold	Warning threshold in percent or -1 to disable
int8_t	[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.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_set_security_state

```
sl_status_t sl_wisun_set_security_state (uint32_t security_state)
```

Set the connection security state.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	security_state	Connection security state • **0**: Authentication and link encryption are disabled • **1**: Authentication and link encryption are enabled

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the connection security state. By default, authentication and link encryption are enabled. The function is intended for test purposes.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Note

- Authentication and link encryption can only be disabled when the application is linked against a library supporting this option.

sl_wisun_set_trusted_certificate

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_status_t sl_wisun_set_trusted_certificate (uint16_t certificate_options, uint16_t certificate_length, const uint8_t * certificate)

Set a trusted CA certificate used to validate the authentication server certificate.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[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
uint16_t	[in]	certificate_length	Size of the certificate data
const uint8_t *	[in]	certificate	Pointer to the certificate data

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets a trusted CA certificate for validiting the authentication server certificate during the authentication process. The supported certificate formats depend on application's Mbed TLS configuration. The default configuration supports PEM and DER formats. When PEM is used, the certificate data may contain multiple concatenated certificates.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

sl_wisun_set_tx_power

sl_status_t sl_wisun_set_tx_power (int8_t tx_power)

Set the maximum TX power.

Parameters

Type	Direction	Argument Name	Description
int8_t	[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. The function must be called before initiating a connection.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

DeprecatedThis function will be removed in the future versions of the Wi-SUN stack, use sl_wisun_set_tx_power_ddbm() instead.

sl_wisun_set_tx_power_ddbm

Copyright © 2025 Silicon Laboratories. All rights reserved.

> sl_status_t sl_wisun_set_tx_power_ddbm (int16_t tx_power_ddbm)

Set the maximum TX power.

Parameters

Type	Direction	Argument Name	Description
int16_t	[in]	tx_power_ddbm	TX power in deci-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. The function must be called before initiating a connection.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Note

- When replacing sl_wisun_set_tx_power() function calls with this function, note the change from dBm units to deci-dBm units.

sl_wisun_set_unicast_settings

> sl_status_t sl_wisun_set_unicast_settings (uint8_t dwell_interval_ms)

Set unicast settings.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[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.

Available in libraries: Full, FFN, BR (see Wi-SUN API availability)

sl_wisun_trigger_frame

> sl_status_t sl_wisun_trigger_frame (sl_wisun_frame_type_t frame_type)

Trigger a transmission of a periodic frame (FAN Discovery, RPL).

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
sl_wisun_frame_type_t	[in]	frame_type	Type of frame to transmit • SL_WISUN_FRAME_TYPE_PAS: Transmit a PAN Advertisement Solicit frame • SL_WISUN_FRAME_TYPE_PA: Transmit a PAN Advertisement frame • SL_WISUN_FRAME_TYPE_PCS: Transmit a PAN Configuration Solicit frame • SL_WISUN_FRAME_TYPE_PC: Transmit a PAN Configuration frame • SL_WISUN_FRAME_TYPE_DIS: Transmit a DODAG Information Solicitation frame • SL_WISUN_FRAME_TYPE_DIO: Transmit a DODAG Information Object frame

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function causes a periodic frame (FAN Discovery, RPL) to be transmitted immediately. The frame can only be transmitted in the appropriate connection state, i.e. the associated trickle timer must be running. The function is intended for test purposes.

Available in libraries: Full, FFN, LFN, BR (see Wi-SUN API availability)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN API events

Wi-SUN API events

Modules

- sl_wisun_evt_t
- sl_wisun_msg_connected_ind
- sl_wisun_msg_pan_defect_ind
- sl_wisun_msg_network_update_ind
- sl_wisun_msg_socket_data_ind
- sl_wisun_msg_socket_data_available_ind
- sl_wisun_msg_socket_connected_ind
- sl_wisun_msg_socket_connection_available_ind
- sl_wisun_msg_socket_closing_ind
- sl_wisun_msg_disconnected_ind
- sl_wisun_msg_connection_lost_ind
- sl_wisun_msg_socket_data_sent_ind
- sl_wisun_msg_error_ind
- sl_wisun_msg_join_state_ind
- sl_wisun_msg_regulation_tx_level_ind
- sl_wisun_mode_switch_fallback_level_ind
- sl_wisun_msg_rx_frame_ind
- sl_wisun_msg_lfn_wake_up_ind
- sl_wisun_msg_lfn_multicast_reg_ind
- sl_wisun_msg_dhcp_vendor_data_ind
- sl_wisun_msg_direct_connect_link_available_ind
- sl_wisun_msg_direct_connect_link_status_ind
- sl_wisun_br_msg_stopped_ind
- sl_wisun_br_msg_routing_table_update_ind

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
    SL_WISUN_MSG_SOCKET_CLOSING_IND_ID = 0x86
    SL_WISUN_MSG_DISCONNECTED_IND_ID = 0x87
    SL_WISUN_MSG_CONNECTION_LOST_IND_ID = 0x88
    SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID = 0x89
    SL_WISUN_MSG_ERROR_IND_ID = 0x8A
    SL_WISUN_MSG_JOIN_STATE_IND_ID = 0x8B
    SL_WISUN_MSG_NETWORK_UPDATE_IND_ID = 0x8C
    SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID = 0x8D
    SL_WISUN_MSG_MODE_SWITCH_FALLBACK_IND_ID = 0x8E
    SL_WISUN_MSG_RX_FRAME_IND_ID = 0x8F
    SL_WISUN_MSG_LFN_WAKE_UP_IND_ID = 0x90
    SL_WISUN_MSG_LFN_MULTICAST_REG_IND_ID = 0x91
    SL_WISUN_MSG_DHCP_VENDOR_DATA_IND_ID = 0x92
    SL_WISUN_MSG_PAN_DEFECT_IND_ID = 0x93
    SL_WISUN_MSG_DIRECT_CONNECT_LINK_AVAILABLE_IND_ID
    = 0x94
    SL_WISUN_MSG_DIRECT_CONNECT_LINK_STATUS_IND_ID =
    0x95
    SL_WISUN_BR_MSG_STOPPED_IND_ID = 0x96
    SL_WISUN_BR_MSG_ROUTING_TABLE_UPDATE_IND_ID =
    0x97
}
```

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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 reception.
SL_WISUN_MSG_LFN_WAKE_UP_IND_ID	This event is sent on LFN wake up.
SL_WISUN_MSG_LFN_MULTICAST_REG_IND_ID	This event is sent when an LFN multicast group registration finishes.
SL_WISUN_MSG_DHCP_VENDOR_DATA_IND_ID	This event is sent when DHCPv6 vendor-specific information is received.
SL_WISUN_MSG_PAN_DEFECT_IND_ID	This event is sent to state a PAN defect state change (true for defective PAN, false for a recovery).
SL_WISUN_MSG_DIRECT_CONNECT_LINK_AVAILABLE_IND_ID	This event is sent when a Direct Connect Link is available.
SL_WISUN_MSG_DIRECT_CONNECT_LINK_STATUS_IND_ID	This event is sent when the status of a Direct Connect Link changes.
SL_WISUN_BR_MSG_STOPPED_IND_ID	This event is sent when a BR stop request has been completed.
SL_WISUN_BR_MSG_ROUTING_TABLE_UPDATE_IND_ID	This event is sent once the routing table is updated (topology change).

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_evt_t

Wi-SUN event definitions.

This structure contains a Wi-SUN API event and its associated data.

Public Attributes

sl_wisun_br_msg_routing_table_update_ind_body_t	br_routing_table_update SL_WISUN_BR_MSG_ROUTING_TABLE_UPDATE_IND_ID event data
sl_wisun_br_msg_stopped_ind_body_t	br_stopped SL_WISUN_BR_MSG_STOPPED_IND_ID event data
sl_wisun_msg_connected_ind_body_t	connected SL_WISUN_MSG_CONNECTED_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_dhcp_vendor_data_ind_body_t	dhcp_vendor_data SL_WISUN_MSG_DHCP_VENDOR_DATA_IND_ID event data
sl_wisun_msg_direct_connect_link_available_ind_body_t	direct_connect_link_available SL_WISUN_MSG_DIRECT_CONNECT_LINK_AVAILABLE_IND_ID event data
sl_wisun_msg_direct_connect_link_status_ind_body_t	direct_connect_link_status SL_WISUN_MSG_DIRECT_CONNECT_LINK_STATUS_IND_ID event data
sl_wisun_msg_disconnected_ind_body_t	disconnected SL_WISUN_MSG_DISCONNECTED_IND_ID event data
sl_wisun_msg_error_ind_body_t	error SL_WISUN_MSG_ERROR_IND_ID event data
union sl_wisun_evt_t::@0	evt Event-specific data.
sl_wisun_msg_header_t	header Common event header.
sl_wisun_msg_join_state_ind_body_t	join_state SL_WISUN_MSG_JOIN_STATE_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
sl_wisun_msg_lfn_wake_up_ind_body_t	lfn_wake_up SL_WISUN_MSG_LFN_WAKE_UP_IND_ID event data

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_network_update_ind_body_t	network_update SL_WISUN_MSG_NETWORK_UPDATE_IND_ID event data
sl_wisun_msg_pan_defect_ind_body_t	pan_defect SL_WISUN_MSG_PAN_DEFECT_IND_ID event data
sl_wisun_msg_regulation_tx_level_ind_body_t	regulation_tx_level SL_WISUN_MSG_REGULATION_TX_LEVEL_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_socket_closing_ind_body_t	socket_closing SL_WISUN_MSG_SOCKET_CLOSING_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_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_data_sent_ind_body_t	socket_data_sent SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID event data

Public Attribute Documentation

br_routing_table_update

> sl_wisun_br_msg_routing_table_update_ind_body_t sl_wisun_evt_t::br_routing_table_update

SL_WISUN_BR_MSG_ROUTING_TABLE_UPDATE_IND_ID event data

br_stopped

> sl_wisun_br_msg_stopped_ind_body_t sl_wisun_evt_t::br_stopped

SL_WISUN_BR_MSG_STOPPED_IND_ID event data

Copyright © 2025 Silicon Laboratories. All rights reserved.

connected

> sl_wisun_msg_connected_ind_body_t sl_wisun_evt_t::connected

SL_WISUN_MSG_CONNECTED_IND_ID event data

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

dhcp_vendor_data

> sl_wisun_msg_dhcp_vendor_data_ind_body_t sl_wisun_evt_t::dhcp_vendor_data

SL_WISUN_MSG_DHCP_VENDOR_DATA_IND_ID event data

direct_connect_link_available

> sl_wisun_msg_direct_connect_link_available_ind_body_t sl_wisun_evt_t::direct_connect_link_available

SL_WISUN_MSG_DIRECT_CONNECT_LINK_AVAILABLE_IND_ID event data

direct_connect_link_status

> sl_wisun_msg_direct_connect_link_status_ind_body_t sl_wisun_evt_t::direct_connect_link_status

SL_WISUN_MSG_DIRECT_CONNECT_LINK_STATUS_IND_ID event data

disconnected

> sl_wisun_msg_disconnected_ind_body_t sl_wisun_evt_t::disconnected

SL_WISUN_MSG_DISCONNECTED_IND_ID event data

error

> sl_wisun_msg_error_ind_body_t sl_wisun_evt_t::error

SL_WISUN_MSG_ERROR_IND_ID event data

Copyright © 2025 Silicon Laboratories. All rights reserved.

evt

union sl_wisun_evt_t::@0 sl_wisun_evt_t::evt

Event-specific data.

This structure contains the event-specific data.

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.

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

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

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

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

network_update

Copyright © 2025 Silicon Laboratories. All rights reserved.

> sl_wisun_msg_network_update_ind_body_t sl_wisun_evt_t::network_update

SL_WISUN_MSG_NETWORK_UPDATE_IND_ID event data

pan_defect

> sl_wisun_msg_pan_defect_ind_body_t sl_wisun_evt_t::pan_defect

SL_WISUN_MSG_PAN_DEFECT_IND_ID event data

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

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

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

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

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

socket_data

Copyright © 2025 Silicon Laboratories. All rights reserved.

> sl_wisun_msg_socket_data_ind_body_t sl_wisun_evt_t::socket_data

SL_WISUN_MSG_SOCKET_DATA_IND_ID event data

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

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_connected_ind

sl_wisun_msg_connected_ind

Modules

- sl_wisun_msg_connected_ind_body_t
- sl_wisun_msg_connected_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_connected_ind_t

Indication message.

Public Attributes

sl_wisun_msg_connected_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_connected_ind_body_t sl_wisun_msg_connected_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_connected_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_pan_defect_ind

sl_wisun_msg_pan_defect_ind

Modules

sl_wisun_msg_pan_defect_ind_body_t

sl_wisun_msg_pan_defect_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_pan_defect_ind_body_t

Indication message body.

Public Attributes

uint32_t	pan_id PAN ID.
uint32_t	state State of the PAN defect.

Public Attribute Documentation

pan_id

```
uint32_t sl_wisun_msg_pan_defect_ind_body_t::pan_id
```

PAN ID.

state

```
uint32_t sl_wisun_msg_pan_defect_ind_body_t::state
```

State of the PAN defect.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_pan_defect_ind_t

Indication message.

Public Attributes

sl_wisun_msg_pan_defect_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_pan_defect_ind_body_t sl_wisun_msg_pan_defect_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_pan_defect_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_network_update_ind

sl_wisun_msg_network_update_ind

Modules

sl_wisun_msg_network_update_ind_body_t

sl_wisun_msg_network_update_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_network_update_ind_body_t

Indication message body.

Public Attributes

uint32_t	flags Bit mask indicating the changes.
uint32_t	status Status of the indication.

Public Attribute Documentation

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
- bit SL_WISUN_NETWORK_UPDATE_FLAGS_HOP_COUNT: estimated hop count has changed

status

```
uint32_t sl_wisun_msg_network_update_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_network_update_ind_t

Indication message.

Public Attributes

sl_wisun_msg_network_update_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_network_update_ind_body_t sl_wisun_msg_network_update_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_network_update_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_ind

sl_wisun_msg_socket_data_ind

Modules

sl_wisun_msg_socket_data_ind_body_t

sl_wisun_msg_socket_data_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_ind_body_t

Indication message body.

Public Attributes

uint8_t	data Received data.
uint16_t	data_length Amount of received data.
in6_addr_t	remote_address IP address of the sender.
uint16_t	remote_port Port number of the sender.
int32_t	socket_id ID of the socket.
uint32_t	status Status of the indication.

Public Attribute Documentation

data

```
uint8_t sl_wisun_msg_socket_data_ind_body_t::data[]
```

Received data.

data_length

```
uint16_t sl_wisun_msg_socket_data_ind_body_t::data_length
```

Amount of received data.

remote_address

```
in6_addr_t sl_wisun_msg_socket_data_ind_body_t::remote_address
```

IP address of the sender.

remote_port

```
uint16_t sl_wisun_msg_socket_data_ind_body_t::remote_port
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Port number of the sender.

socket_id

```
int32_t sl_wisun_msg_socket_data_ind_body_t::socket_id
```

ID of the socket.

status

```
uint32_t sl_wisun_msg_socket_data_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_ind_t

Indication message.

Public Attributes

sl_wisun_msg_socket_data_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_socket_data_ind_body_t sl_wisun_msg_socket_data_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_data_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_available_ind

sl_wisun_msg_socket_data_available_ind

Modules

sl_wisun_msg_socket_data_available_ind_body_t

sl_wisun_msg_socket_data_available_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_available_ind_body_t

Indication message body.

Public Attributes

uint16_t	data_length Amount of data that can be read.
uint16_t	reserved Reserved, set to zero.
int32_t	socket_id ID of the socket.
uint32_t	status Status of the indication.

Public Attribute Documentation

data_length

```
uint16_t sl_wisun_msg_socket_data_available_ind_body_t::data_length
```

Amount of data that can be read.

reserved

```
uint16_t sl_wisun_msg_socket_data_available_ind_body_t::reserved
```

Reserved, set to zero.

socket_id

```
int32_t sl_wisun_msg_socket_data_available_ind_body_t::socket_id
```

ID of the socket.

status

```
uint32_t sl_wisun_msg_socket_data_available_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_available_ind_t

Indication message.

Public Attributes

sl_wisun_msg_socket_data_available_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_socket_data_available_ind_body_t sl_wisun_msg_socket_data_available_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_data_available_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_connected_ind

sl_wisun_msg_socket_connected_ind

Modules

sl_wisun_msg_socket_connected_ind_body_t

sl_wisun_msg_socket_connected_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_connected_ind_body_t

Indication message body.

Public Attributes

int32_t	socket_id ID of the socket.
uint32_t	status Status of the indication.

Public Attribute Documentation

socket_id

```
int32_t sl_wisun_msg_socket_connected_ind_body_t::socket_id
```

ID of the socket.

status

```
uint32_t sl_wisun_msg_socket_connected_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_connected_ind_t

Indication message.

Public Attributes

sl_wisun_msg_socket_connected_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_socket_connected_ind_body_t sl_wisun_msg_socket_connected_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_connected_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_connection_available_ind

sl_wisun_msg_socket_connection_available_ind

Modules

sl_wisun_msg_socket_connection_available_ind_body_t

sl_wisun_msg_socket_connection_available_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_connection_available_ind_body_t

Indication message body.

Public Attributes

int32_t	socket_id ID of the socket.
uint32_t	status Status of the indication.

Public Attribute Documentation

socket_id

```
int32_t sl_wisun_msg_socket_connection_available_ind_body_t::socket_id
```

ID of the socket.

status

```
uint32_t sl_wisun_msg_socket_connection_available_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_connection_available_ind_t

Indication message.

Public Attributes

sl_wisun_msg_socket_connection_available_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_socket_connection_available_ind_body_t sl_wisun_msg_socket_connection_available_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_connection_available_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_closing_ind

sl_wisun_msg_socket_closing_ind

Modules

sl_wisun_msg_socket_closing_ind_body_t

sl_wisun_msg_socket_closing_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_closing_ind_body_t

Indication message body.

Public Attributes

int32_t	socket_id ID of the socket.
uint32_t	status Status of the indication.

Public Attribute Documentation

socket_id

```
int32_t sl_wisun_msg_socket_closing_ind_body_t::socket_id
```

ID of the socket.

status

```
uint32_t sl_wisun_msg_socket_closing_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_closing_ind_t

Indication message.

Public Attributes

sl_wisun_msg_socket_closing_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_socket_closing_ind_body_t sl_wisun_msg_socket_closing_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_closing_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_disconnected_ind

sl_wisun_msg_disconnected_ind

Modules

sl_wisun_msg_disconnected_ind_body_t

sl_wisun_msg_disconnected_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_disconnected_ind_t

Indication message.

Public Attributes

sl_wisun_msg_disconnected_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_disconnected_ind_body_t sl_wisun_msg_disconnected_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_disconnected_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_connection_lost_ind

sl_wisun_msg_connection_lost_ind

Modules

sl_wisun_msg_connection_lost_ind_body_t

sl_wisun_msg_connection_lost_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_connection_lost_ind_t

Indication message.

Public Attributes

sl_wisun_msg_connection_lost_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_connection_lost_ind_body_t sl_wisun_msg_connection_lost_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_connection_lost_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_sent_ind

sl_wisun_msg_socket_data_sent_ind

Modules

sl_wisun_msg_socket_data_sent_ind_body_t

sl_wisun_msg_socket_data_sent_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_sent_ind_body_t

Indication message body.

Public Attributes

int32_t	socket_id ID of the socket.
uint32_t	socket_space_left Amount of free space in the transmission buffer.
uint32_t	status Status of the indication.

Public Attribute Documentation

socket_id

```
int32_t sl_wisun_msg_socket_data_sent_ind_body_t::socket_id
```

ID of the socket.

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.

status

```
uint32_t sl_wisun_msg_socket_data_sent_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_socket_data_sent_ind_t

Indication message.

Public Attributes

sl_wisun_msg_socket_data_sent_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_socket_data_sent_ind_body_t sl_wisun_msg_socket_data_sent_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_socket_data_sent_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_error_ind

sl_wisun_msg_error_ind

Modules

sl_wisun_msg_error_ind_body_t

sl_wisun_msg_error_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_error_ind_t

Indication message.

Public Attributes

sl_wisun_msg_error_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_error_ind_body_t sl_wisun_msg_error_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_error_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_join_state_ind

sl_wisun_msg_join_state_ind

Modules

- sl_wisun_msg_join_state_ind_body_t
- sl_wisun_msg_join_state_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_join_state_ind_body_t

Indication message body.

Public Attributes

uint32_t	join_state Join state.
uint32_t	status Status of the indication.

Public Attribute Documentation

join_state

```
uint32_t sl_wisun_msg_join_state_ind_body_t::join_state
```

Join state.

status

```
uint32_t sl_wisun_msg_join_state_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_join_state_ind_t

Indication message.

Public Attributes

sl_wisun_msg_join_state_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_join_state_ind_body_t sl_wisun_msg_join_state_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_join_state_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_regulation_tx_level_ind

sl_wisun_msg_regulation_tx_level_ind

Modules

sl_wisun_msg_regulation_tx_level_ind_body_t

sl_wisun_msg_regulation_tx_level_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_regulation_tx_level_ind_body_t

Indication message body.

Public Attributes

uint8_t	reserved Reserved.
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.

Public Attribute Documentation

reserved

```
uint8_t sl_wisun_msg_regulation_tx_level_ind_body_t::reserved[3]
```

Reserved.

status

```
uint32_t sl_wisun_msg_regulation_tx_level_ind_body_t::status
```

Status of the indication.

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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_regulation_tx_level_ind_t

Indication message.

Public Attributes

sl_wisun_msg_regulation_tx_level_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_regulation_tx_level_ind_body_t sl_wisun_msg_regulation_tx_level_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_regulation_tx_level_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_mode_switch_fallback_level_ind

sl_wisun_mode_switch_fallback_level_ind

Modules

sl_wisun_msg_mode_switch_fallback_ind_body_t

sl_wisun_msg_mode_switch_fallback_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_mode_switch_fallback_ind_body_t

Indication message body.

Public Attributes

sl_wisun_mac_address_t	address MAC address of the peer triggering the fallback.
uint32_t	status Status of the indication.

Public Attribute Documentation

address

```
sl_wisun_mac_address_t sl_wisun_msg_mode_switch_fallback_ind_body_t::address
```

MAC address of the peer triggering the fallback.

status

```
uint32_t sl_wisun_msg_mode_switch_fallback_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_mode_switch_fallback_ind_t

Indication message.

Public Attributes

sl_wisun_msg_mode_switch_fallback_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_mode_switch_fallback_ind_body_t sl_wisun_msg_mode_switch_fallback_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_mode_switch_fallback_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_rx_frame_ind

sl_wisun_msg_rx_frame_ind

Modules

sl_wisun_msg_rx_frame_ind_body_t

sl_wisun_msg_rx_frame_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_rx_frame_ind_body_t

Indication message body.

Public Attributes

uint8_t	frame Received frame.
uint16_t	length Frame length in bytes.
uint32_t	status Status of the indication.
uint64_t	timestamp_us Timestamp in microseconds.

Public Attribute Documentation

frame

```
uint8_t sl_wisun_msg_rx_frame_ind_body_t::frame[]
```

Received frame.

length

```
uint16_t sl_wisun_msg_rx_frame_ind_body_t::length
```

Frame length in bytes.

status

```
uint32_t sl_wisun_msg_rx_frame_ind_body_t::status
```

Status of the indication.

timestamp_us

```
uint64_t sl_wisun_msg_rx_frame_ind_body_t::timestamp_us
```

Timestamp in microseconds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_rx_frame_ind_t

Indication message.

Public Attributes

sl_wisun_msg_rx_frame_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_rx_frame_ind_body_t sl_wisun_msg_rx_frame_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_rx_frame_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_lfn_wake_up_ind

sl_wisun_msg_lfn_wake_up_ind

Modules

sl_wisun_msg_lfn_wake_up_ind_body_t

sl_wisun_msg_lfn_wake_up_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_lfn_wake_up_ind_body_t

Indication message body.

Public Attributes

uint64_t	next_wup_us Expected time to next wake-up in microseconds.
uint32_t	status Status of the indication.
uint32_t	wup_duration_us Expected wake-up duration in microseconds.

Public Attribute Documentation

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.

status

```
uint32_t sl_wisun_msg_lfn_wake_up_ind_body_t::status
```

Status of the indication.

wup_duration_us

```
uint32_t sl_wisun_msg_lfn_wake_up_ind_body_t::wup_duration_us
```

Expected wake-up duration in microseconds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_lfn_wake_up_ind_t

Indication message.

Public Attributes

sl_wisun_msg_lfn_wake_up_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_lfn_wake_up_ind_body_t sl_wisun_msg_lfn_wake_up_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_lfn_wake_up_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_lfn_multicast_reg_ind

sl_wisun_msg_lfn_multicast_reg_ind

Modules

sl_wisun_msg_lfn_multicast_reg_ind_body_t

sl_wisun_msg_lfn_multicast_reg_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_lfn_multicast_reg_ind_body_t

Indication message body.

Public Attributes

in6_addr_t	ip_address Registered multicast IP address.
uint32_t	status Status of the indication.

Public Attribute Documentation

ip_address

```
in6_addr_t sl_wisun_msg_lfn_multicast_reg_ind_body_t::ip_address
```

Registered multicast IP address.

status

```
uint32_t sl_wisun_msg_lfn_multicast_reg_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_lfn_multicast_reg_ind_t

Indication message.

Public Attributes

sl_wisun_msg_lfn_multicast_reg_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_lfn_multicast_reg_ind_body_t sl_wisun_msg_lfn_multicast_reg_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_lfn_multicast_reg_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_dhcp_vendor_data_ind

sl_wisun_msg_dhcp_vendor_data_ind

Modules

sl_wisun_msg_dhcp_vendor_data_ind_body_t

sl_wisun_msg_dhcp_vendor_data_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_dhcp_vendor_data_ind_body_t

Indication message body.

Public Attributes

uint8_t	data Vendor specific data.
uint16_t	data_length Length of Vendor specific data.
uint32_t	enterprise_number Vendor Specific Enterprise Number.

Public Attribute Documentation

data

```
uint8_t sl_wisun_msg_dhcp_vendor_data_ind_body_t::data[]
```

Vendor specific data.

data_length

```
uint16_t sl_wisun_msg_dhcp_vendor_data_ind_body_t::data_length
```

Length of Vendor specific data.

enterprise_number

```
uint32_t sl_wisun_msg_dhcp_vendor_data_ind_body_t::enterprise_number
```

Vendor Specific Enterprise Number.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_dhcp_vendor_data_ind_t

Indication message.

Public Attributes

sl_wisun_msg_dhcp_vendor_data_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_dhcp_vendor_data_ind_body_t sl_wisun_msg_dhcp_vendor_data_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_dhcp_vendor_data_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_direct_connect_link_available_ind

sl_wisun_msg_direct_connect_link_available_ind

Modules

sl_wisun_msg_direct_connect_link_available_ind_body_t

sl_wisun_msg_direct_connect_link_available_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_direct_connect_link_available_ind_body_t

Indication message body.

Public Attributes

in6_addr_t	link_local_ipv6 LinkLocal IPv6 address of the client.

Public Attribute Documentation

link_local_ipv6

```
in6_addr_t sl_wisun_msg_direct_connect_link_available_ind_body_t::link_local_ipv6
```

LinkLocal IPv6 address of the client.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_direct_connect_link_available_ind_t

Indication message.

Public Attributes

sl_wisun_msg_direct_connect_link_available_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_direct_connect_link_available_ind_body_t sl_wisun_msg_direct_connect_link_available_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_direct_connect_link_available_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_direct_connect_link_status_ind

sl_wisun_msg_direct_connect_link_status_ind

Modules

sl_wisun_msg_direct_connect_link_status_ind_body_t

sl_wisun_msg_direct_connect_link_status_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_direct_connect_link_status_ind_body_t

Indication message body.

Public Attributes

in6_addr_t	link_local_ipv6 LinkLocal IPv6 address of the client.
uint32_t	link_status Link status.

Public Attribute Documentation

link_local_ipv6

```
in6_addr_t sl_wisun_msg_direct_connect_link_status_ind_body_t::link_local_ipv6
```

LinkLocal IPv6 address of the client.

link_status

```
uint32_t sl_wisun_msg_direct_connect_link_status_ind_body_t::link_status
```

Link status.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_direct_connect_link_status_ind_t

Indication message.

Public Attributes

sl_wisun_msg_direct_connect_link_status_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_msg_direct_connect_link_status_ind_body_t sl_wisun_msg_direct_connect_link_status_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_msg_direct_connect_link_status_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_msg_stopped_ind

sl_wisun_br_msg_stopped_ind

Modules

sl_wisun_br_msg_stopped_ind_body_t

sl_wisun_br_msg_stopped_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_msg_stopped_ind_body_t

Indication message body.

Public Attributes

uint32_t	status Status of the indication.

Public Attribute Documentation

status

```
uint32_t sl_wisun_br_msg_stopped_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_msg_stopped_ind_t

Indication message.

Public Attributes

sl_wisun_br_msg_stopped_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_br_msg_stopped_ind_body_t sl_wisun_br_msg_stopped_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_br_msg_stopped_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_msg_routing_table_update_ind

sl_wisun_br_msg_routing_table_update_ind

Modules

sl_wisun_br_msg_routing_table_update_ind_body_t

sl_wisun_br_msg_routing_table_update_ind_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_msg_routing_table_update_ind_body_t

Indication message body.

Public Attributes

uint32_t	event Routing table update event SL_WISUN_ROUTING_TABLE_UPDATE_ROUTE_CHANGED: route changed.
uint32_t	status Status of the indication.

Public Attribute Documentation

event

```
uint32_t sl_wisun_br_msg_routing_table_update_ind_body_t::event
```

Routing table update event SL_WISUN_ROUTING_TABLE_UPDATE_ROUTE_CHANGED: route changed.

status

```
uint32_t sl_wisun_br_msg_routing_table_update_ind_body_t::status
```

Status of the indication.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_msg_routing_table_update_ind_t

Indication message.

Public Attributes

sl_wisun_br_msg_routing_table_update_ind_body_t	body Indication message body.
sl_wisun_msg_header_t	header Common message header.

Public Attribute Documentation

body

```
sl_wisun_br_msg_routing_table_update_ind_body_t sl_wisun_br_msg_routing_table_update_ind_t::body
```

Indication message body.

header

```
sl_wisun_msg_header_t sl_wisun_br_msg_routing_table_update_ind_t::header
```

Common message header.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN API type definitions

Modules

- sl_wisun_br_connection_params_t
- sl_wisun_br_lfn_params_t
- sl_wisun_br_params_configuration_s
- sl_wisun_br_params_dhcp_s
- sl_wisun_br_params_discovery_s
- sl_wisun_br_params_eapol_s
- sl_wisun_br_params_lfn_parent_s
- sl_wisun_br_params_misc_s
- sl_wisun_br_params_mpl_s
- sl_wisun_br_params_rpl_s
- sl_wisun_br_params_traffic_s
- sl_wisun_br_routing_table_entry_t
- sl_wisun_channel_mask_t
- sl_wisun_connection_params_t
- sl_wisun_key_lifetimes_params_t
- sl_wisun_lfn_params_connection_t
- sl_wisun_lfn_params_data_layer_t
- sl_wisun_lfn_params_network_t
- sl_wisun_lfn_params_power_t
- sl_wisun_lfn_params_t
- sl_wisun_mac_address_t
- sl_wisun_mac_params_t
- sl_wisun_msg_header_t
- sl_wisun_neighbor_info_t
- sl_wisun_network_info_t
- sl_wisun_params_configuration_s
- sl_wisun_params_discovery_s
- sl_wisun_params_eapol_s
- sl_wisun_params_lfn_parent_s

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_misc_s

sl_wisun_params_mpl_s

sl_wisun_params_rpl_s

sl_wisun_params_traffic_s

sl_wisun_phy_config_custom_fsk_t

sl_wisun_phy_config_custom_ofdm_t

sl_wisun_phy_config_custom_oqpsk_t

sl_wisun_phy_config_explicit_t

sl_wisun_phy_config_fan10_t

sl_wisun_phy_config_fan11_t

sl_wisun_phy_config_ids_t

sl_wisun_phy_config_t

sl_wisun_rate_stats_t

sl_wisun_rate_t

sl_wisun_rpl_info_t

sl_wisun_socket_option_data_t

sl_wisun_socket_option_edfe_mode_t

sl_wisun_socket_option_event_mode_t

sl_wisun_socket_option_multicast_group_t

sl_wisun_socket_option_multicast_hop_limit

sl_wisun_socket_option_send_buffer_limit_t

sl_wisun_socket_option_unicast_hop_limit

sl_wisun_statistics_arib_regulation_t

sl_wisun_statistics_fhss_t

sl_wisun_statistics_heap_t

sl_wisun_statistics_mac_t

sl_wisun_statistics_network_t

sl_wisun_statistics_phy_t

sl_wisun_statistics_regulation_t

sl_wisun_statistics_t

sl_wisun_statistics_wisun_t

sl_wisun_trace_group_config_t

sl_wisun_trickle_params_t

Predefined FFN parameter sets

Copyright © 2025 Silicon Laboratories. All rights reserved.

Predefined LFN parameter sets

Predefined BR parameter sets

Predefined BR LFN-parenting parameter sets

Enumerations

enum sl_direct_connect_link_status_t {

SL_WISUN_DIRECT_CONNECT_LINK_STATUS_CONNECTED
SL_WISUN_DIRECT_CONNECT_LINK_STATUS_ERROR
SL_WISUN_DIRECT_CONNECT_LINK_STATUS_DISCONNECTED

}

Enumeration for Direct Connect Link status.

enum sl_wisun_br_state_t {

SL_WISUN_BR_STATE_INITIALIZED = 0
SL_WISUN_BR_STATE_OPERATIONAL = 1

}

Border Router state.

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_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_channel_mask_type_t {

SL_WISUN_CHANNEL_MASK_TYPE_REGULATORY
SL_WISUN_CHANNEL_MASK_TYPE_REGIONAL = SL_WISUN_CHANNEL_MASK_TYPE_REGULATORY
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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_device_type_t {

SL_WISUN_ROUTER = 0
SL_WISUN_LFN = 1
SL_WISUN_BORDER_ROUTER = 2

}

Enumerations for device type.

enum sl_wisun_fan_version_t {

SL_WISUN_FAN_VERSION_DEFAULT = 0
SL_WISUN_FAN_VERSION_1_0 = 1
SL_WISUN_FAN_VERSION_1_1 = 2

}

Enumeration for FAN versions to which the node is certified.

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_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_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_DISCONNECTING = 6
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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_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_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_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_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_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
SL_WISUN_NETWORK_UPDATE_FLAGS_HOP_COUNT = 3

}

Enumerations for network update flags.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_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_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_regulation_t {

SL_WISUN_REGULATION_NONE = 0
SL_WISUN_REGULATION_ARIB = 1
SL_WISUN_REGULATION_WPC = 2
SL_WISUN_REGULATION_ETSI = 3

}

Enumerations for regional regulation.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_routing_table_update_event_t {

SL_WISUN_ROUTING_TABLE_UPDATE_ROUTE_CHANGED = 0

}

Enumeration for routing table update types.

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 since v2.0 MUST NOT be used with POSIX socket API.

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 since v2.0 MUST NOT be used with POSIX socket API.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_RALG = 39
SL_WISUN_TRACE_GROUP_FSM = 40
SL_WISUN_TRACE_GROUP_APP = 41
SL_WISUN_TRACE_GROUP_DC = 42
SL_WISUN_TRACE_GROUP_REG = 43
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.

Typedefs

typedef struct sl_wisun_br_params_configuration sl_wisun_br_params_configuration_s

PAN configuration parameter set.

Copyright © 2025 Silicon Laboratories. All rights reserved.

typedef struct sl_wisun_br_params_configuration_s — sl_wisun_br_params_configuration_t
PAN configuration parameter set.

typedef struct sl_wisun_br_params_dhcp_s — sl_wisun_br_params_dhcp
DHCP server parameter set.

typedef struct sl_wisun_br_params_dhcp_s — sl_wisun_br_params_dhcp_t
DHCP server parameter set.

typedef struct sl_wisun_br_params_discovery_s — sl_wisun_br_params_discovery
PAN discovery parameter set.

typedef struct sl_wisun_br_params_discovery_s — sl_wisun_br_params_discovery_t
PAN discovery parameter set.

typedef struct sl_wisun_br_params_eapol_s — sl_wisun_br_params_eapol
Authentication parameter set.

typedef struct sl_wisun_br_params_eapol_s — sl_wisun_br_params_eapol_t
Authentication parameter set.

typedef struct sl_wisun_br_params_lfn_parent_s — sl_wisun_br_params_lfn_parent
LFN parent parameter set.

typedef struct sl_wisun_br_params_lfn_parent_s — sl_wisun_br_params_lfn_parent_t
LFN parent parameter set.

typedef struct sl_wisun_br_params_misc_s — sl_wisun_br_params_misc
Misc parameter set.

typedef struct sl_wisun_br_params_misc_s — sl_wisun_br_params_misc_t
Misc parameter set.

typedef struct sl_wisun_br_params_mpl_s — sl_wisun_br_params_mpl
MPL parameter set.

typedef struct sl_wisun_br_params_mpl_s — sl_wisun_br_params_mpl_t
MPL parameter set.

typedef struct sl_wisun_br_params_rpl_s — sl_wisun_br_params_rpl
RPL parameter set.

typedef struct sl_wisun_br_params_rpl_s — sl_wisun_br_params_rpl_t
RPL parameter set.

typedef struct sl_wisun_br_params_traffic_s — sl_wisun_br_params_traffic_t
Traffic parameter set.

Copyright © 2025 Silicon Laboratories. All rights reserved.

typedef in6_addr_t	sl_wisun_ip_address_t IPv6 address structure.
typedef sl_status_t(*	sl_wisun_ipv6_up_handler_t)(const uint8_t *data, size_t data_length) Handler called for an IPv6 packet from Wi-SUN network.
typedef struct sl_wisun_params_configuration_s	sl_wisun_params_configuration PAN configuration parameter set.
typedef struct sl_wisun_params_configuration_s	sl_wisun_params_configuration_t PAN configuration parameter set.
typedef struct sl_wisun_params_discovery_s	sl_wisun_params_discovery PAN discovery parameter set.
typedef struct sl_wisun_params_discovery_s	sl_wisun_params_discovery_t PAN discovery parameter set.
typedef struct sl_wisun_params_eapol_s	sl_wisun_params_eapol Authentication parameter set.
typedef struct sl_wisun_params_eapol_s	sl_wisun_params_eapol_t Authentication parameter set.
typedef struct sl_wisun_params_lfn_parent_s	sl_wisun_params_lfn_parent LFN parent parameter set.
typedef struct sl_wisun_params_lfn_parent_s	sl_wisun_params_lfn_parent_t LFN parent parameter set.
typedef struct sl_wisun_params_misc_s	sl_wisun_params_misc Misc parameter set.
typedef struct sl_wisun_params_misc_s	sl_wisun_params_misc_t Misc parameter set.
typedef struct sl_wisun_params_mpl_s	sl_wisun_params_mpl MPL parameter set.
typedef struct sl_wisun_params_mpl_s	sl_wisun_params_mpl_t MPL parameter set.
typedef struct sl_wisun_params_rpl_s	sl_wisun_params_rpl RPL parameter set.
typedef struct sl_wisun_params_rpl_s	sl_wisun_params_rpl_t RPL parameter set.
typedef struct sl_wisun_params_traffic_s	sl_wisun_params_traffic_t Traffic parameter set.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Variables

const sl_wisun_mac_address_t **sl_wisun_broadcast_mac**
Broadcast MAC address.

Macros

#define **SL_WISUN_ADVERT_FRAGMENT_DISABLE** UINT32_MAX
Maximum fragment duration. Disables advert fragmentation.

#define **SL_WISUN_CHANNEL_MASK_SIZE** 32
Size of a channel mask.

#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_EAP_IDENTITY_SIZE** 32
Maximum size of EAP identity.

#define **SL_WISUN_FILTER_BITFIELD_SIZE** ((SL_WISUN_TRACE_GROUP_COUNT + 7) / 8)
Size of the filter bitfield.

#define **SL_WISUN_GAK_LEN** 16
Length of a GAK in bytes.

#define **SL_WISUN_GTK_LEN** 16
Length of a GTK in bytes.

#define **SL_WISUN_GTK_NUM** 4
Number of GTKs returned by sl_wisun_get_gtks()

#define **SL_WISUN_LGTK_NUM** 3
Number of LGTKs returned by sl_wisun_get_gtks()

#define **SL_WISUN_MAC_ADDRESS_SIZE** 8
Size of a MAC address.

#define **SL_WISUN_MAX_PHY_MODE_ID_COUNT** 15
Maximum number of PhyModeId allowed in POM-IE.

#define **SL_WISUN_NETWORK_NAME_SIZE** 32
Maximum size of the Wi-SUN network name.

#define **SL_WISUN_PMK_LEN** 32
Length of a PMK in bytes.

#define **SL_WISUN_TRACE_THREAD_EVENT_LOOP** "EVL"
Thread identifier "Wi-SUN Event Loop Task".

Copyright © 2025 Silicon Laboratories. All rights reserved.

#define SL_WISUN_TRACE_THREAD_EVENT_TASK "EVT"
Thread identifier "Wi-SUN Event Task".

#define SL_WISUN_TRACE_THREAD_MAC "MAC"
Thread identifier "Wi-SUN RF Task".

#define SL_WISUN_TRACE_THREAD_WISUN "WS"
Thread identifier "Wi-SUN Task".

Enumeration Documentation

sl_direct_connect_link_status_t

> sl_direct_connect_link_status_t

Enumeration for Direct Connect Link status.

Enumerator	
SL_WISUN_DIRECT_CONNECT_LINK_STATUS_CONNECTED	Direct Connect Link connected.
SL_WISUN_DIRECT_CONNECT_LINK_STATUS_ERROR	Direct Connect Link error.
SL_WISUN_DIRECT_CONNECT_LINK_STATUS_DISCONNECTED	Direct Connect Link disconnected.

sl_wisun_br_state_t

> sl_wisun_br_state_t

Border Router state.

Enumerator	
SL_WISUN_BR_STATE_INITIALIZED	Border Router is initialized.
SL_WISUN_BR_STATE_OPERATIONAL	Border Router is started.

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.

sl_wisun_channel_exclusion_mode_t

> sl_wisun_channel_exclusion_mode_t

Enumeration for channel exlusion modes.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

sl_wisun_channel_mask_type_t

> sl_wisun_channel_mask_type_t

Enumeration for channel mask types.

Enumerator	
SL_WISUN_CHANNEL_MASK_TYPE_REGULATORY	Regulatory excluded channel mask (not advertised)
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.

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

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)

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_WISUN_2_BYTES_CRC	2 bytes CRC
SL_WISUN_4_BYTES_CRC	4 bytes CRC

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.

sl_wisun_fan_version_t

sl_wisun_fan_version_t

Enumeration for FAN versions to which the node is certified.

Enumerator	
SL_WISUN_FAN_VERSION_DEFAULT	Keep node's default FAN TPS version.
SL_WISUN_FAN_VERSION_1_0	FAN 1.0 certified node.
SL_WISUN_FAN_VERSION_1_1	FAN 1.1 certified node.

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	PAN Advertisement Solicit.
SL_WISUN_FRAME_TYPE_PA	PAN Advertisement.
SL_WISUN_FRAME_TYPE_PCS	PAN Configuration Solicit.
SL_WISUN_FRAME_TYPE_PC	PAN Configuration.
SL_WISUN_FRAME_TYPE_DIS	DODAG Information Solicitation.
SL_WISUN_FRAME_TYPE_DIO	DODAG Information Object.

sl_wisun_ip_address_type_t

sl_wisun_ip_address_type_t

Enumerations for IP address type.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

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_DISCONNECTING	Disconnecting.
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.

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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

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.

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.

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.

sl_wisun_network_update_flags_t

> sl_wisun_network_update_flags_t

Enumerations for network update flags.

Enumerator	

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.
SL_WISUN_NETWORK_UPDATE_FLAGS_HOP_COUNT	Hop count modification flag bit.

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.

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.

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 PHY configuration.
SL_WISUN_PHY_CONFIG_FAN11	FAN1.1 PHY configuration.

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_WISUN_PHY_CONFIG_EXPLICIT	Explicit PHY configuration.
SL_WISUN_PHY_CONFIG_IDS	Explicit RAIL configuration.
SL_WISUN_PHY_CONFIG_CUSTOM_FSK	Custom FSK PHY configuration.
SL_WISUN_PHY_CONFIG_CUSTOM_OFDM	Custom OFDM PHY configuration.
SL_WISUN_PHY_CONFIG_CUSTOM_OQPSK	Custom OQPSK PHY configuration.

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.

sl_wisun_regulation_t

> sl_wisun_regulation_t

Enumerations for regional regulation.

Enumerator	
SL_WISUN_REGULATION_NONE	No regulation.
SL_WISUN_REGULATION_ARIB	ARIB, should only be used with JP regulatory domain.
SL_WISUN_REGULATION_WPC	WPC, should only be used with IN regulatory domain.
SL_WISUN_REGULATION_ETSI	ETSI, should only be used with EU regulatory domain.

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.

sl_wisun_regulatory_domain_t

> sl_wisun_regulatory_domain_t

Enumerations for regulatory domain.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

sl_wisun_routing_table_update_event_t

> sl_wisun_routing_table_update_event_t

Enumeration for routing table update types.

Enumerator	
SL_WISUN_ROUTING_TABLE_UPDATE_ROUTE_CHANGED	At least one route is updated in the routing table.

sl_wisun_socket_option_t

> sl_wisun_socket_option_t

Enumerations for socket option Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_protocol_t

sl_wisun_socket_protocol_t

Enumerations for socket protocol Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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)

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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_RALG	Adaptive rate algorithms.
SL_WISUN_TRACE_GROUP_FSM	Finite state machine.
SL_WISUN_TRACE_GROUP_APP	Application.
SL_WISUN_TRACE_GROUP_DC	Direct Connect.
SL_WISUN_TRACE_GROUP_REG	Regional regulation.
SL_WISUN_TRACE_GROUP_INT	Internal usage.
SL_WISUN_TRACE_GROUP_COUNT	Max number of trace group in this enum.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_WISUN_TRACE_LEVEL_INFO	Info + warning + error.
SL_WISUN_TRACE_LEVEL_DEBUG	Debug + info + warning + error.

Typedef Documentation

sl_wisun_br_params_configuration

```
typedef struct sl_wisun_br_params_configuration_s sl_wisun_br_params_configuration
```

PAN configuration parameter set.

sl_wisun_br_params_configuration_t

```
typedef struct sl_wisun_br_params_configuration_s sl_wisun_br_params_configuration_t
```

PAN configuration parameter set.

sl_wisun_br_params_dhcp

```
typedef struct sl_wisun_br_params_dhcp_s sl_wisun_br_params_dhcp
```

DHCP server parameter set.

sl_wisun_br_params_dhcp_t

```
typedef struct sl_wisun_br_params_dhcp_s sl_wisun_br_params_dhcp_t
```

DHCP server parameter set.

sl_wisun_br_params_discovery

```
typedef struct sl_wisun_br_params_discovery_s sl_wisun_br_params_discovery
```

PAN discovery parameter set.

sl_wisun_br_params_discovery_t

```
typedef struct sl_wisun_br_params_discovery_s sl_wisun_br_params_discovery_t
```

PAN discovery parameter set.

sl_wisun_br_params_eapol

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_wisun_br_params_eapol_s sl_wisun_br_params_eapol
```

Authentication parameter set.

sl_wisun_br_params_eapol_t

```
typedef struct sl_wisun_br_params_eapol_s sl_wisun_br_params_eapol_t
```

Authentication parameter set.

sl_wisun_br_params_lfn_parent

```
typedef struct sl_wisun_br_params_lfn_parent_s sl_wisun_br_params_lfn_parent
```

LFN parent parameter set.

sl_wisun_br_params_lfn_parent_t

```
typedef struct sl_wisun_br_params_lfn_parent_s sl_wisun_br_params_lfn_parent_t
```

LFN parent parameter set.

sl_wisun_br_params_misc

```
typedef struct sl_wisun_br_params_misc_s sl_wisun_br_params_misc
```

Misc parameter set.

sl_wisun_br_params_misc_t

```
typedef struct sl_wisun_br_params_misc_s sl_wisun_br_params_misc_t
```

Misc parameter set.

sl_wisun_br_params_mpl

```
typedef struct sl_wisun_br_params_mpl_s sl_wisun_br_params_mpl
```

MPL parameter set.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_mpl_t

```
typedef struct sl_wisun_br_params_mpl_s sl_wisun_br_params_mpl_t
```

MPL parameter set.

sl_wisun_br_params_rpl

```
typedef struct sl_wisun_br_params_rpl_s sl_wisun_br_params_rpl
```

RPL parameter set.

sl_wisun_br_params_rpl_t

```
typedef struct sl_wisun_br_params_rpl_s sl_wisun_br_params_rpl_t
```

RPL parameter set.

sl_wisun_br_params_traffic_t

```
typedef struct sl_wisun_br_params_traffic_s sl_wisun_br_params_traffic_t
```

Traffic parameter set.

sl_wisun_ip_address_t

```
typedef in6_addr_t sl_wisun_ip_address_t
```

IPv6 address structure.

sl_wisun_ipv6_up_handler_t

```
typedef sl_status_t(* sl_wisun_ipv6_up_handler_t) (const uint8_t *data, size_t data_length) )(const uint8_t *data, size_t data_length)
```

Handler called for an IPv6 packet from Wi-SUN network.

Parameters

Type	Direction	Argument Name	Description
	[in]	data	Pointer to IPv6 header
	[out]	data_length	Length of IPv6 packet data in bytes

Returns

-

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_STATUS_OK if successful, an error code otherwise

sl_wisun_params_configuration

```
typedef struct sl_wisun_params_configuration_s sl_wisun_params_configuration
```

PAN configuration parameter set.

sl_wisun_params_configuration_t

```
typedef struct sl_wisun_params_configuration_s sl_wisun_params_configuration_t
```

PAN configuration parameter set.

sl_wisun_params_discovery

```
typedef struct sl_wisun_params_discovery_s sl_wisun_params_discovery
```

PAN discovery parameter set.

sl_wisun_params_discovery_t

```
typedef struct sl_wisun_params_discovery_s sl_wisun_params_discovery_t
```

PAN discovery parameter set.

sl_wisun_params_eapol

```
typedef struct sl_wisun_params_eapol_s sl_wisun_params_eapol
```

Authentication parameter set.

sl_wisun_params_eapol_t

```
typedef struct sl_wisun_params_eapol_s sl_wisun_params_eapol_t
```

Authentication parameter set.

sl_wisun_params_lfn_parent

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_wisun_params_lfn_parent_s sl_wisun_params_lfn_parent
```

LFN parent parameter set.

sl_wisun_params_lfn_parent_t

```
typedef struct sl_wisun_params_lfn_parent_s sl_wisun_params_lfn_parent_t
```

LFN parent parameter set.

sl_wisun_params_misc

```
typedef struct sl_wisun_params_misc_s sl_wisun_params_misc
```

Misc parameter set.

sl_wisun_params_misc_t

```
typedef struct sl_wisun_params_misc_s sl_wisun_params_misc_t
```

Misc parameter set.

sl_wisun_params_mpl

```
typedef struct sl_wisun_params_mpl_s sl_wisun_params_mpl
```

MPL parameter set.

sl_wisun_params_mpl_t

```
typedef struct sl_wisun_params_mpl_s sl_wisun_params_mpl_t
```

MPL parameter set.

sl_wisun_params_rpl

```
typedef struct sl_wisun_params_rpl_s sl_wisun_params_rpl
```

RPL parameter set.

sl_wisun_params_rpl_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct sl_wisun_params_rpl_s sl_wisun_params_rpl_t
```

RPL parameter set.

sl_wisun_params_traffic_t

```
typedef struct sl_wisun_params_traffic_s sl_wisun_params_traffic_t
```

Traffic parameter set.

Variable Documentation

sl_wisun_broadcast_mac

```
const sl_wisun_mac_address_t sl_wisun_broadcast_mac
```

Broadcast MAC address.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_connection_params_t

BR parameter set.

Public Attributes

sl_wisun_br_params_configuration_t	configuration PAN configuration parameter set.
sl_wisun_br_params_dhcp_t	dhcp DHCP server parameter set.
sl_wisun_br_params_discovery_t	discovery PAN discovery parameter set.
sl_wisun_br_params_eapol_t	eapol Authentication parameter set.
uint8_t	enable_ffn10 Enable authentication of FAN 1.0 routers.
sl_wisun_br_params_lfn_parent_t	lfn_parent LFN parent parameter set.
sl_wisun_mac_params_t	mac MAC parameter set.
sl_wisun_br_params_misc_t	misc Misc parameter set.
sl_wisun_br_params_mpl_t	mpl MPL parameter set.
uint32_t	pan_capacity Maximum number of nodes supported by border router at once.
uint8_t	reserved Reserved, set to zero.
sl_wisun_br_params_rpl_t	rpl RPL parameter set.
sl_wisun_br_params_traffic_t	traffic Traffic parameter set.
uint32_t	version Version of this API.

Public Attribute Documentation

configuration

> sl_wisun_br_params_configuration_t sl_wisun_br_connection_params_t::configuration

PAN configuration parameter set.

dhcp

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_dhcp_t sl_wisun_br_connection_params_t::dhcp

DHCP server parameter set.

discovery

sl_wisun_br_params_discovery_t sl_wisun_br_connection_params_t::discovery

PAN discovery parameter set.

eapol

sl_wisun_br_params_eapol_t sl_wisun_br_connection_params_t::eapol

Authentication parameter set.

enable_ffn10

uint8_t sl_wisun_br_connection_params_t::enable_ffn10

Enable authentication of FAN 1.0 routers.

lfn_parent

sl_wisun_br_params_lfn_parent_t sl_wisun_br_connection_params_t::lfn_parent

LFN parent parameter set.

mac

sl_wisun_mac_params_t sl_wisun_br_connection_params_t::mac

MAC parameter set.

misc

sl_wisun_br_params_misc_t sl_wisun_br_connection_params_t::misc

Misc parameter set.

mpl

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_mpl_t sl_wisun_br_connection_params_t::mpl

MPL parameter set.

pan_capacity

uint32_t sl_wisun_br_connection_params_t::pan_capacity

Maximum number of nodes supported by border router at once.

reserved

uint8_t sl_wisun_br_connection_params_t::reserved[3]

Reserved, set to zero.

rpl

sl_wisun_br_params_rpl_t sl_wisun_br_connection_params_t::rpl

RPL parameter set.

traffic

sl_wisun_br_params_traffic_t sl_wisun_br_connection_params_t::traffic

Traffic parameter set.

version

uint32_t sl_wisun_br_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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_lfn_params_t

BR LFN parenting parameter set.

Public Attributes

uint32_t	lfn_broadcast_interval_ms Time between LFN Broadcast windows (millisecond) Specification range [10000, 600000] (10s to 10m)
uint8_t	lfn_broadcast_sync_period Number of LFN Broadcast Intervals between broadcast sync messages Specification range [1, 60].
uint8_t	reserved Reserved, set to zero.
uint32_t	version Version of this API, must be the first field.

Public Attribute Documentation

lfn_broadcast_interval_ms

```
uint32_t sl_wisun_br_lfn_params_t::lfn_broadcast_interval_ms
```

Time between LFN Broadcast windows (millisecond) Specification range [10000, 600000] (10s to 10m)

lfn_broadcast_sync_period

```
uint8_t sl_wisun_br_lfn_params_t::lfn_broadcast_sync_period
```

Number of LFN Broadcast Intervals between broadcast sync messages Specification range [1, 60].

reserved

```
uint8_t sl_wisun_br_lfn_params_t::reserved[3]
```

Reserved, set to zero.

version

```
uint32_t sl_wisun_br_lfn_params_t::version
```

Version of this API, must be the first field.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_configuration_s

PAN configuration parameter set.

Public Attributes

sl_wisun_trickle_params_t	trickle_pc PAN configuration trickle timer.

Public Attribute Documentation

trickle_pc

```
sl_wisun_trickle_params_t sl_wisun_br_params_configuration_s::trickle_pc
```

PAN configuration trickle timer.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_dhcp_s

DHCP server parameter set.

Public Attributes

uint32_t	dhcp_address_lifetime_s Lease time (seconds)

Public Attribute Documentation

dhcp_address_lifetime_s

```
uint32_t sl_wisun_br_params_dhcp_s::dhcp_address_lifetime_s
```

Lease time (seconds)

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_discovery_s

PAN discovery parameter set.

Public Attributes

sl_wisun_trickle_params_t	trickle_pa PAN advertisement trickle timer.

Public Attribute Documentation

trickle_pa

```
sl_wisun_trickle_params_t sl_wisun_br_params_discovery_s::trickle_pa
```

PAN advertisement trickle timer.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_eapol_s

Authentication parameter set.

Public Attributes

sl_wisun_key_lifetimes_params_t	key_lifetimes Security key lifetimes.
uint8_t	reserved Reserved, set to zero.
sl_wisun_trickle_params_t	sec_prot_trickle Deprecated.
uint8_t	sec_prot_trickle_expirations Deprecated.
uint16_t	temp_min_timeout_s Deprecated.

Public Attribute Documentation

key_lifetimes

```
sl_wisun_key_lifetimes_params_t sl_wisun_br_params_eapol_s::key_lifetimes
```

Security key lifetimes.

reserved

```
uint8_t sl_wisun_br_params_eapol_s::reserved
```

Reserved, set to zero.

sec_prot_trickle

```
sl_wisun_trickle_params_t sl_wisun_br_params_eapol_s::sec_prot_trickle
```

Deprecated.

sec_prot_trickle_expirations

```
uint8_t sl_wisun_br_params_eapol_s::sec_prot_trickle_expirations
```

Deprecated.

Copyright © 2025 Silicon Laboratories. All rights reserved.

temp_min_timeout_s

uint16_t sl_wisun_br_params_eapol_s::temp_min_timeout_s

Deprecated.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_lfn_parent_s

LFN parent parameter set.

Public Attributes

uint8_t	lfn_lpc_retry_count Number of broadcast LFN Pan Config retries when LFN Version is incremented Referred to as LFN_MAINTAIN_PARENT_TIME in FAN TPS 1.1.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

lfn_lpc_retry_count

```
uint8_t sl_wisun_br_params_lfn_parent_s::lfn_lpc_retry_count
```

Number of broadcast LFN Pan Config retries when LFN Version is incremented Referred to as LFN_MAINTAIN_PARENT_TIME in FAN TPS 1.1.

reserved

```
uint8_t sl_wisun_br_params_lfn_parent_s::reserved[3]
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_misc_s

Misc parameter set.

Public Attributes

uint16_t	reserved Reserved, set to zero.
uint16_t	temp_link_min_timeout_s Temporary neighbor link minimum timeout.

Public Attribute Documentation

reserved

```
uint16_t sl_wisun_br_params_misc_s::reserved
```

Reserved, set to zero.

temp_link_min_timeout_s

```
uint16_t sl_wisun_br_params_misc_s::temp_link_min_timeout_s
```

Temporary neighbor link minimum timeout.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_mpl_s

MPL parameter set.

Public Attributes

uint8_t	reserved Reserved, set to zero.
uint16_t	seed_set_entry_lifetime_s MPL seed set entry lifetime.
sl_wisun_trickle_params_t	trickle Warning: MPL parameters may be overwritten for border router MPL trickle timer.
uint8_t	trickle_expirations MPL trickle timer expirations.

Public Attribute Documentation

reserved

```
uint8_t sl_wisun_br_params_mpl_s::reserved
```

Reserved, set to zero.

seed_set_entry_lifetime_s

```
uint16_t sl_wisun_br_params_mpl_s::seed_set_entry_lifetime_s
```

MPL seed set entry lifetime.

trickle

```
sl_wisun_trickle_params_t sl_wisun_br_params_mpl_s::trickle
```

Warning: MPL parameters may be overwritten for border router MPL trickle timer.

trickle_expirations

```
uint8_t sl_wisun_br_params_mpl_s::trickle_expirations
```

MPL trickle timer expirations.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_rpl_s

RPL parameter set.

Public Attributes

uint8_t	reserved Reserved, set to zero.
uint16_t	rpl_dag_max_rank_increase RPL allowable increase in rank in support of local repair RFC 6550 6.7.6.
uint8_t	rpl_default_lifetime RPL default lifetime for routes (lifetime units) RFC6550 6.7.6.
uint16_t	rpl_default_lifetime_unit RPL default lifetime unit RFC6550 6.7.6.
uint8_t	rpl_dio_interval_doublings RPL trickle Imax (DIOIntervalDoublings)
uint8_t	rpl_dio_interval_min RPL trickle Imin equals (2^rpl_dio_interval_min) ms.
uint8_t	rpl_dio_redundancy_constant RPL trickle k DIORedundancyConstant.
uint16_t	rpl_min_hop_rank_increase RPL MinHopRankIncrease RFC6550 3.5.1.

Public Attribute Documentation

reserved

```
uint8_t sl_wisun_br_params_rpl_s::reserved[2]
```

Reserved, set to zero.

rpl_dag_max_rank_increase

```
uint16_t sl_wisun_br_params_rpl_s::rpl_dag_max_rank_increase
```

RPL allowable increase in rank in support of local repair RFC 6550 6.7.6.

rpl_default_lifetime

```
uint8_t sl_wisun_br_params_rpl_s::rpl_default_lifetime
```

RPL default lifetime for routes (lifetime units) RFC6550 6.7.6.

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint16_t sl_wisun_br_params_rpl_s::rpl_default_lifetime_unit

RPL default lifetime unit RFC6550 6.7.6.

rpl_dio_interval_doublings

uint8_t sl_wisun_br_params_rpl_s::rpl_dio_interval_doublings

RPL trickle Imax (DIOIntervalDoublings)

rpl_dio_interval_min

uint8_t sl_wisun_br_params_rpl_s::rpl_dio_interval_min

RPL trickle Imin equals (2^rpl_dio_interval_min) ms.

rpl_dio_redundancy_constant

uint8_t sl_wisun_br_params_rpl_s::rpl_dio_redundancy_constant

RPL trickle k DIORedundancyConstant.

rpl_min_hop_rank_increase

uint16_t sl_wisun_br_params_rpl_s::rpl_min_hop_rank_increase

RPL MinHopRankIncrease RFC6550 3.5.1.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_params_traffic_s

Traffic parameter set.

Public Attributes

uint16_t	ipv6_mru Maximum Receive Unit (MRU) for fragmented IPv6 packets in bytes. A larger packet will be silently discarded.
uint16_t	lowpan_mtu Maximum Transmission Unit (MTU) for 6LoWPAN packets in bytes. A larger packet will be fragmented using 6LoWPAN fragmentation.
uint8_t	max_edfe_fragment_count Maximum number of 6LoWPAN/IPv6 fragments to send in a single EDFE transaction (0 to 10). 0 disables EDFE for fragmented packets.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

ipv6_mru

```
uint16_t sl_wisun_br_params_traffic_s::ipv6_mru
```

Maximum Receive Unit (MRU) for fragmented IPv6 packets in bytes. A larger packet will be silently discarded.

lowpan_mtu

```
uint16_t sl_wisun_br_params_traffic_s::lowpan_mtu
```

Maximum Transmission Unit (MTU) for 6LoWPAN packets in bytes. A larger packet will be fragmented using 6LoWPAN fragmentation.

max_edfe_fragment_count

```
uint8_t sl_wisun_br_params_traffic_s::max_edfe_fragment_count
```

Maximum number of 6LoWPAN/IPv6 fragments to send in a single EDFE transaction (0 to 10). 0 disables EDFE for fragmented packets.

reserved

```
uint8_t sl_wisun_br_params_traffic_s::reserved[3]
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_routing_table_entry_t

Border Router routing table entry.

Public Attributes

in6_addr_t	backup GUA/ULA of the routed node's backup parent. INADDR_ANY if not set.
in6_addr_t	preferred GUA/ULA of the routed node's preferred parent.
in6_addr_t	target GUA/ULA of the routed node.

Public Attribute Documentation

backup

```
in6_addr_t sl_wisun_br_routing_table_entry_t::backup
```

GUA/ULA of the routed node's backup parent. INADDR_ANY if not set.

preferred

```
in6_addr_t sl_wisun_br_routing_table_entry_t::preferred
```

GUA/ULA of the routed node's preferred parent.

target

```
in6_addr_t sl_wisun_br_routing_table_entry_t::target
```

GUA/ULA of the routed node.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_connection_params_t

FFN parameter set.

Public Attributes

sl_wisun_params_configuration_t	configuration PAN configuration parameter set.
sl_wisun_params_eapol_t	direct_connect_eapol Direct Connect Authentication parameter set.
sl_wisun_params_discovery_t	discovery PAN discovery parameter set.
sl_wisun_params_eapol_t	eapol Authentication parameter set.
sl_wisun_params_lfn_parent_t	lfn_parent LFN parent parameter set.
sl_wisun_mac_params_t	mac MAC parameter set.
sl_wisun_params_misc_t	misc Misc parameter set.
sl_wisun_params_mpl_t	mpl MPL parameter set.
sl_wisun_params_rpl_t	rpl RPL parameter set.
sl_wisun_params_traffic_t	traffic Traffic parameter set.
uint32_t	version Version of this API.

Public Attribute Documentation

configuration

```
sl_wisun_params_configuration_t sl_wisun_connection_params_t::configuration
```

PAN configuration parameter set.

direct_connect_eapol

```
sl_wisun_params_eapol_t sl_wisun_connection_params_t::direct_connect_eapol
```

Direct Connect Authentication parameter set.

Copyright © 2025 Silicon Laboratories. All rights reserved.

> sl_wisun_params_discovery_t sl_wisun_connection_params_t::discovery

PAN discovery parameter set.

eapol

> sl_wisun_params_eapol_t sl_wisun_connection_params_t::eapol

Authentication parameter set.

lfn_parent

> sl_wisun_params_lfn_parent_t sl_wisun_connection_params_t::lfn_parent

LFN parent parameter set.

mac

> sl_wisun_mac_params_t sl_wisun_connection_params_t::mac

MAC parameter set.

misc

> sl_wisun_params_misc_t sl_wisun_connection_params_t::misc

Misc parameter set.

mpl

> sl_wisun_params_mpl_t sl_wisun_connection_params_t::mpl

MPL parameter set.

rpl

> sl_wisun_params_rpl_t sl_wisun_connection_params_t::rpl

RPL parameter set.

traffic

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
sl_wisun_params_traffic_t sl_wisun_connection_params_t::traffic
```

Traffic parameter set.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_key_lifetimes_params_t

Key lifetimes parameter set.

Public Attributes

uint32_t **ffn_revocation_lifetime_reduction**
FFN_REVOCATION_LIFETIME_REDUCTION Factor by which the active GTK lifetime is reduced during node revocation procedures.

uint32_t **gtk_expire_offset_m**
GTK_EXPIRE_OFFSET (minutes) The expiration time of a GTK is calculated as the expiration time of the GTK most recently installed at the Border Router plus GTK_EXPIRE_OFFSET.

uint32_t **gtk_new_activation_time**
GTK_NEW_ACTIVATION_TIME The time at which the Border Router activates the next GTK prior to expiration of the currently activated GTK.

uint32_t **gtk_new_install_required**
GTK_NEW_INSTALL_REQUIRED The percentage of time elapsed in the active GTK's lifetime at which a new GTK must be installed on the Border Router.

uint32_t **lfn_revocation_lifetime_reduction**
LFN_REVOCATION_LIFETIME_REDUCTION Factor by which the active LGTK lifetime is reduced during node revocation procedures.

uint32_t **lgtk_expire_offset_m**
LGTK_EXPIRE_OFFSET (minutes) The expiration time of a LGTK is calculated as the expiration time of the GTK most recently installed at the Border Router plus LGTK_EXPIRE_OFFSET.

uint32_t **lgtk_new_activation_time**
LGTK_NEW_ACTIVATION_TIME The time at which the Border Router activates the next LGTK prior to expiration of the currently activated LGTK.

uint32_t **lgtk_new_install_required**
LGTK_NEW_INSTALL_REQUIRED The percentage of time elapsed in the active LGTK's lifetime at which a new LGTK must be installed on the Border Router.

uint32_t **lpmk_lifetime_m**
LPMK lifetime (minutes)

uint32_t **lptk_lifetime_m**
LPTK lifetime (minutes)

uint32_t **pmk_lifetime_m**
PMK lifetime (minutes)

uint32_t **ptk_lifetime_m**
PTK lifetime (minutes)

Public Attribute Documentation

ffn_revocation_lifetime_reduction

> uint32_t sl_wisun_key_lifetimes_params_t::ffn_revocation_lifetime_reduction

Copyright © 2025 Silicon Laboratories. All rights reserved.

FFN_REVOCATION_LIFETIME_REDUCTION Factor by which the active GTK lifetime is reduced during node revocation procedures.

gtk_expire_offset_m

```
uint32_t sl_wisun_key_lifetimes_params_t::gtk_expire_offset_m
```

GTK_EXPIRE_OFFSET (minutes) The expiration time of a GTK is calculated as the expiration time of the GTK most recently installed at the Border Router plus GTK_EXPIRE_OFFSET.

gtk_new_activation_time

```
uint32_t sl_wisun_key_lifetimes_params_t::gtk_new_activation_time
```

GTK_NEW_ACTIVATION_TIME The time at which the Border Router activates the next GTK prior to expiration of the currently activated GTK.

gtk_new_install_required

```
uint32_t sl_wisun_key_lifetimes_params_t::gtk_new_install_required
```

GTK_NEW_INSTALL_REQUIRED The percentage of time elapsed in the active GTK's lifetime at which a new GTK must be installed on the Border Router.

lfn_revocation_lifetime_reduction

```
uint32_t sl_wisun_key_lifetimes_params_t::lfn_revocation_lifetime_reduction
```

LFN_REVOCATION_LIFETIME_REDUCTION Factor by which the active LGTK lifetime is reduced during node revocation procedures.

lgtk_expire_offset_m

```
uint32_t sl_wisun_key_lifetimes_params_t::lgtk_expire_offset_m
```

LGTK_EXPIRE_OFFSET (minutes) The expiration time of a LGTK is calculated as the expiration time of the GTK most recently installed at the Border Router plus LGTK_EXPIRE_OFFSET.

lgtk_new_activation_time

```
uint32_t sl_wisun_key_lifetimes_params_t::lgtk_new_activation_time
```

LGTK_NEW_ACTIVATION_TIME The time at which the Border Router activates the next LGTK prior to expiration of the currently activated LGTK.

Copyright © 2025 Silicon Laboratories. All rights reserved.

lgtk_new_install_required

```
uint32_t sl_wisun_key_lifetimes_params_t::lgtk_new_install_required
```

LGTK_NEW_INSTALL_REQUIRED The percentage of time elapsed in the active LGTK's lifetime at which a new LGTK must be installed on the Border Router.

lpmk_lifetime_m

```
uint32_t sl_wisun_key_lifetimes_params_t::lpmk_lifetime_m
```

LPMK lifetime (minutes)

lptk_lifetime_m

```
uint32_t sl_wisun_key_lifetimes_params_t::lptk_lifetime_m
```

LPTK lifetime (minutes)

pmk_lifetime_m

```
uint32_t sl_wisun_key_lifetimes_params_t::pmk_lifetime_m
```

PMK lifetime (minutes)

ptk_lifetime_m

```
uint32_t sl_wisun_key_lifetimes_params_t::ptk_lifetime_m
```

PTK lifetime (minutes)

Copyright © 2025 Silicon Laboratories. All rights reserved.

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].

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].

reserved

```
uint8_t sl_wisun_lfn_params_connection_t::reserved[2]
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_lfn_params_data_layer_t

LFN data layer parameters.

Public Attributes

uint32_t	eapol_unicast_interval_ms Shorter LFN Unicast interval used during the EAPOL exchange (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.
uint32_t	unicast_interval_max_ms Maximum acceptable LFN unicast interval (milliseconds)
uint32_t	unicast_interval_min_ms Minimum acceptable LFN unicast interval (milliseconds)
uint32_t	unicast_interval_ms Initial LFN Unicast interval proposed by the LFN (milliseconds).

Public Attribute Documentation

eapol_unicast_interval_ms

```
uint32_t sl_wisun_lfn_params_data_layer_t::eapol_unicast_interval_ms
```

Shorter LFN Unicast interval used during the EAPOL exchange (milliseconds).

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]

reserved

```
uint8_t sl_wisun_lfn_params_data_layer_t::reserved[3]
```

Reserved, set to zero.

unicast_interval_max_ms

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint32_t sl_wisun_lfn_params_data_layer_t::unicast_interval_max_ms

Maximum acceptable LFN unicast interval (milliseconds)

unicast_interval_min_ms

uint32_t sl_wisun_lfn_params_data_layer_t::unicast_interval_min_ms

Minimum acceptable LFN unicast interval (milliseconds)

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_lfn_params_network_t

LFN network parameters.

Public Attributes

uint8_t	lfn_na_wait_duration_m Duration for which an LFN waits for a registration confirmation (minutes).
uint16_t	lfn_registration_lifetime_m Address registration lifetime (IPv6 lease duration) the LFN requires to the Border Router (minutes).
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

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).

This value is administratively configured and should be consistent throughout the network. Make sure that the value is equal to or greather than the corresponding field on FFNs. Specification range [30, 120] (set lower for test purposes)

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]

reserved

```
uint8_t sl_wisun_lfn_params_network_t::reserved
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_lfn_params_power_t

LFN power parameters.

Public Attributes

uint8_t	broadcast_lts_only If true, the LFN wakes up only for broadcast slots containing synchronization information.
uint16_t	listening_window_min_us Minimum duration of the listening window.
uint8_t	reserved Reserved, set to zero.
uint16_t	window_margin_min_us Minimum margin added to the listening window (before and after).

Public Attribute Documentation

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.

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.

reserved

```
uint8_t sl_wisun_lfn_params_power_t::reserved[3]
```

Reserved, set to zero.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_lfn_params_t

LFN parameter set.

Public Attributes

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.
uint32_t	version Version of this API.

Public Attribute Documentation

connection

```
sl_wisun_lfn_params_connection_t sl_wisun_lfn_params_t::connection
```

LFN connection parameters.

data_layer

```
sl_wisun_lfn_params_data_layer_t sl_wisun_lfn_params_t::data_layer
```

LFN data layer parameters.

network

```
sl_wisun_lfn_params_network_t sl_wisun_lfn_params_t::network
```

LFN network parameters.

power

```
sl_wisun_lfn_params_power_t sl_wisun_lfn_params_t::power
```

LFN power parameters.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_mac_params_t

MAC parameter set.

Public Attributes

uint16_t	backoff_period_us Length of one backoff period in microseconds.
uint8_t	max_be Maximum value of CSMA-CA backoff exponent. The default value is 5.
uint8_t	max_cca_retries Maximum number of CCA retries.
uint8_t	max_frame_retries Maximum number of transmission retries.
uint8_t	min_be Minimum value of CSMA-CA backoff exponent. The default value is 3.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

backoff_period_us

```
uint16_t sl_wisun_mac_params_t::backoff_period_us
```

Length of one backoff period in microseconds.

If 0, the length will be calculated based on the PHY. The default value is 0.

max_be

```
uint8_t sl_wisun_mac_params_t::max_be
```

Maximum value of CSMA-CA backoff exponent. The default value is 5.

max_cca_retries

```
uint8_t sl_wisun_mac_params_t::max_cca_retries
```

Maximum number of CCA retries.

The transmission is aborted if the channel is still busy after 1 + max_cca_retries attempts. The default value is 8.

max_frame_retries

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint8_t sl_wisun_mac_params_t::max_frame_retries

Maximum number of transmission retries.

The transmission is aborted if no acknowledgment has been received after 1 + max_frame_retries attempts. The default value is 19.

min_be

uint8_t sl_wisun_mac_params_t::min_be

Minimum value of CSMA-CA backoff exponent. The default value is 3.

reserved

uint8_t sl_wisun_mac_params_t::reserved[2]

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_msg_header_t

Wi-SUN Message API common header.

Public Attributes

uint8_t	id ID (request, confirmation, indication) of the message.
uint8_t	info Processing metadata for the message.
uint16_t	length Total length of the message in bytes, this field included.

Public Attribute Documentation

id

```
uint8_t sl_wisun_msg_header_t::id
```

ID (request, confirmation, indication) of the message.

info

```
uint8_t sl_wisun_msg_header_t::info
```

Processing metadata for the message.

length

```
uint16_t sl_wisun_msg_header_t::length
```

Total length of the message in bytes, this field included.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_neighbor_info_t

RPL neighbor information.

Public Attributes

uint16_t	etx Measured ETX value if known (0xffff if unknown)
in6_addr_t	global_address ULA/GUA address (unspecified address :: if unknown)
uint8_t	is_lfn Indicate if the device is an LFN. 1 = LFN, 0 = FFN.
uint8_t	is_mdr_command_capable Indicate if the neighbor supports MAC mode switch.
uint32_t	lifetime Remaining lifetime (Link lifetime for parents, EARO lifetime for children) in seconds.
in6_addr_t	link_local_address Link-local address.
uint32_t	mac_rx_count MAC RX packet count.
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.
uint16_t	pan_size Number devices connected to Border Router.
uint8_t	phy_mode_id_count Number of PhyModeId supported.
uint8_t	phy_mode_ids List of phy_mode_id_count PhyModeId.
uint16_t	routing_cost ETX to Border Router.
uint16_t	rpl_rank RPL Rank value for parents (0xffff if unknown or child)
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.
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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

int8_t	rssi RSSI of the last received packet in integer dBm. */.
uint32_t	type Neighbor type (sl_wisun_neighbor_type_t)

Public Attribute Documentation

etx

```
uint16_t sl_wisun_neighbor_info_t::etx
```

Measured ETX value if known (0xffff if unknown)

global_address

```
in6_addr_t sl_wisun_neighbor_info_t::global_address
```

ULA/GUA address (unspecified address :: if unknown)

is_lfn

```
uint8_t sl_wisun_neighbor_info_t::is_lfn
```

Indicate if the device is an LFN. 1 = LFN, 0 = FFN.

is_mdr_command_capable

```
uint8_t sl_wisun_neighbor_info_t::is_mdr_command_capable
```

Indicate if the neighbor supports MAC mode switch.

lifetime

```
uint32_t sl_wisun_neighbor_info_t::lifetime
```

Remaining lifetime (Link lifetime for parents, EARO lifetime for children) in seconds.

link_local_address

```
in6_addr_t sl_wisun_neighbor_info_t::link_local_address
```

Link-local address.

Copyright © 2025 Silicon Laboratories. All rights reserved.

mac_rx_count

```
uint32_t sl_wisun_neighbor_info_t::mac_rx_count
```

MAC RX packet count.

mac_tx_count

```
uint32_t sl_wisun_neighbor_info_t::mac_tx_count
```

MAC TX packet count.

mac_tx_failed_count

```
uint32_t sl_wisun_neighbor_info_t::mac_tx_failed_count
```

MAC TX failed count.

mac_tx_ms_count

```
uint32_t sl_wisun_neighbor_info_t::mac_tx_ms_count
```

MAC TX packet count using mode switch.

mac_tx_ms_failed_count

```
uint32_t sl_wisun_neighbor_info_t::mac_tx_ms_failed_count
```

MAC TX failed count using mode switch.

pan_size

```
uint16_t sl_wisun_neighbor_info_t::pan_size
```

Number devices connected to Border Router.

phy_mode_id_count

```
uint8_t sl_wisun_neighbor_info_t::phy_mode_id_count
```

Number of PhyModeId supported.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

routing_cost

```
uint16_t sl_wisun_neighbor_info_t::routing_cost
```

ETX to Border Router.

rpl_rank

```
uint16_t sl_wisun_neighbor_info_t::rpl_rank
```

RPL Rank value for parents (0xffff if unknown or child)

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.

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.

rssi

```
int8_t sl_wisun_neighbor_info_t::rssi
```

RSSI of the last received packet in integer dBm. */.

type

```
uint32_t sl_wisun_neighbor_info_t::type
```

Neighbor type (sl_wisun_neighbor_type_t)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_network_info_t

Wi-SUN network information.

Public Attributes

uint8_t	hop_count Hop count.
uint16_t	pan_id PAN ID.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

hop_count

```
uint8_t sl_wisun_network_info_t::hop_count
```

Hop count.

pan_id

```
uint16_t sl_wisun_network_info_t::pan_id
```

PAN ID.

reserved

```
uint8_t sl_wisun_network_info_t::reserved
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_configuration_s

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_s::trickle_pc
```

PAN Configuration trickle timer.

trickle_pcs

```
sl_wisun_trickle_params_t sl_wisun_params_configuration_s::trickle_pcs
```

PAN Configuration Solicit trickle timer.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_discovery_s

PAN discovery parameter set.

Public Attributes

uint8_t	allow_skip If true, allow join state 1 to be skipped using cached information from the previous connection.
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	reserved Reserved, set to zero.
sl_wisun_trickle_params_t	trickle_pa PAN Advertisement trickle timer.
sl_wisun_trickle_params_t	trickle_pas PAN Advertisement Solicit trickle timer.

Public Attribute Documentation

allow_skip

```
uint8_t sl_wisun_params_discovery_s::allow_skip
```

If true, allow join state 1 to be skipped using cached information from the previous connection.

eapol_target_min_sens

```
uint8_t sl_wisun_params_discovery_s::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.

reserved

```
uint8_t sl_wisun_params_discovery_s::reserved[2]
```

Reserved, set to zero.

trickle_pa

```
sl_wisun_trickle_params_t sl_wisun_params_discovery_s::trickle_pa
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

PAN Advertisement trickle timer.

trickle_pas

```
sl_wisun_trickle_params_t sl_wisun_params_discovery_s::trickle_pas
```

PAN Advertisement Solicit trickle timer.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_eapol_s

Authentication parameter set.

Public Attributes

uint8_t	allow_skip If true, allow join state 2 to be skipped using cached credentials from the previous connection.
uint16_t	gtk_max_mismatch_m GTK_MAX_MISMATCH (minutes) Maximum time between a SUP detecting a GTKHASH mismatch and the SUP initiating Msg1 of the authentication flow.
uint16_t	gtk_request_imax_m Deprecated.
uint16_t	gtk_request_imin_m Deprecated.
uint16_t	initial_key_max_s Max delay of first key request (seconds)
uint16_t	initial_key_min_s Deprecated.
uint8_t	initial_key_retry_limit Maximum retransmission count.
uint16_t	initial_key_retry_max_limit_s Maximum duration for retransmissions (seconds)
uint16_t	initial_key_retry_max_s Deprecated.
uint16_t	initial_key_retry_min_s Initial retransmission time (seconds)
uint16_t	lgtk_max_mismatch_m LGTK_MAX_MISMATCH (minutes) Maximum time between a SUP detecting a LGTKHASH mismatch and the SUP initiating Msg1 of the authentication flow.
uint32_t	pmk_lifetime_m PMK lifetime (minutes)
uint32_t	ptk_lifetime_m PTK lifetime (minutes)
uint8_t	reserved Reserved, set to zero.
uint16_t	sec_prot_retry_timeout_s Deprecated.
sl_wisun_trickle_params_t	sec_prot_trickle Deprecated.
uint8_t	sec_prot_trickle_expirations Deprecated.
uint16_t	temp_min_timeout_s Deprecated.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Public Attribute Documentation

allow_skip

```
uint8_t sl_wisun_params_eapol_s::allow_skip
```

If true, allow join state 2 to be skipped using cached credentials from the previous connection.

gtk_max_mismatch_m

```
uint16_t sl_wisun_params_eapol_s::gtk_max_mismatch_m
```

GTK_MAX_MISMATCH (minutes) Maximum time between a SUP detecting a GTKHASH mismatch and the SUP initiating Msg1 of the authentication flow.

gtk_request_imax_m

```
uint16_t sl_wisun_params_eapol_s::gtk_request_imax_m
```

Deprecated.

gtk_request_imin_m

```
uint16_t sl_wisun_params_eapol_s::gtk_request_imin_m
```

Deprecated.

initial_key_max_s

```
uint16_t sl_wisun_params_eapol_s::initial_key_max_s
```

Max delay of first key request (seconds)

initial_key_min_s

```
uint16_t sl_wisun_params_eapol_s::initial_key_min_s
```

Deprecated.

initial_key_retry_limit

```
uint8_t sl_wisun_params_eapol_s::initial_key_retry_limit
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Maximum retransmission count.

initial_key_retry_max_limit_s

```
uint16_t sl_wisun_params_eapol_s::initial_key_retry_max_limit_s
```

Maximum duration for retransmissions (seconds)

initial_key_retry_max_s

```
uint16_t sl_wisun_params_eapol_s::initial_key_retry_max_s
```

Deprecated.

initial_key_retry_min_s

```
uint16_t sl_wisun_params_eapol_s::initial_key_retry_min_s
```

Initial retransmission time (seconds)

lgtk_max_mismatch_m

```
uint16_t sl_wisun_params_eapol_s::lgtk_max_mismatch_m
```

LGTK_MAX_MISMATCH (minutes) Maximum time between a SUP detecting a LGTKHASH mismatch and the SUP initiating Msg1 of the authentication flow.

pmk_lifetime_m

```
uint32_t sl_wisun_params_eapol_s::pmk_lifetime_m
```

PMK lifetime (minutes)

ptk_lifetime_m

```
uint32_t sl_wisun_params_eapol_s::ptk_lifetime_m
```

PTK lifetime (minutes)

reserved

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint8_t sl_wisun_params_eapol_s::reserved[3]

Reserved, set to zero.

sec_prot_retry_timeout_s

uint16_t sl_wisun_params_eapol_s::sec_prot_retry_timeout_s

Deprecated.

sec_prot_trickle

sl_wisun_trickle_params_t sl_wisun_params_eapol_s::sec_prot_trickle

Deprecated.

sec_prot_trickle_expirations

uint8_t sl_wisun_params_eapol_s::sec_prot_trickle_expirations

Deprecated.

temp_min_timeout_s

uint16_t sl_wisun_params_eapol_s::temp_min_timeout_s

Deprecated.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_lfn_parent_s

LFN parent parameter set.

Public Attributes

uint8_t	lfn_lpc_retry_count Number of broadcast LFN Pan Config retries when LFN Version is incremented Referred to as LFN_MAINTAIN_PARENT_TIME in FAN TPS 1.1.
uint8_t	lfn_na_wait_duration_m Duration for which an FFN retransmits DAO on behalf of a registering LFN (minutes).
uint16_t	lfn_pan_timeout_m Interval after which an FFN parent must disable LFN broadcast messaging if the FFN parent is unable communicate with its Border Router.

Public Attribute Documentation

lfn_lpc_retry_count

```
uint8_t sl_wisun_params_lfn_parent_s::lfn_lpc_retry_count
```

Number of broadcast LFN Pan Config retries when LFN Version is incremented Referred to as LFN_MAINTAIN_PARENT_TIME in FAN TPS 1.1.

lfn_na_wait_duration_m

```
uint8_t sl_wisun_params_lfn_parent_s::lfn_na_wait_duration_m
```

Duration for which an FFN retransmits DAO on behalf of a registering LFN (minutes).

This value is administratively configured and should be consistent throughout the network. Make sure that the value is equal to or less than the corresponding field on LFNs. Specification range [30, 120] (set lower for test purposes)

lfn_pan_timeout_m

```
uint16_t sl_wisun_params_lfn_parent_s::lfn_pan_timeout_m
```

Interval after which an FFN parent must disable LFN broadcast messaging if the FFN parent is unable communicate with its Border Router.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_misc_s

Misc parameter set.

Public Attributes

uint8_t	pan_timeout_m Border router communication timeout PAN_TIMEOUT.
uint8_t	reserved Reserved, set to zero.
uint16_t	temp_link_min_timeout_s Temporary neighbor link minimum timeout.

Public Attribute Documentation

pan_timeout_m

```
uint8_t sl_wisun_params_misc_s::pan_timeout_m
```

Border router communication timeout PAN_TIMEOUT.

reserved

```
uint8_t sl_wisun_params_misc_s::reserved
```

Reserved, set to zero.

temp_link_min_timeout_s

```
uint16_t sl_wisun_params_misc_s::temp_link_min_timeout_s
```

Temporary neighbor link minimum timeout.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_mpl_s

MPL parameter set.

Public Attributes

uint8_t	seed_id_type MPL seed-id type, 0 indicates the seed-id is elided and is the IPv6 source address, 3 indicates the seed-id is the 128-bit ULA/GUA of the source.
uint16_t	seed_set_entry_lifetime_s MPL seed set entry lifetime (seconds)
sl_wisun_trickle_params_t	trickle MPL trickle timer.
uint8_t	trickle_expirations MPL trickle timer expirations.

Public Attribute Documentation

seed_id_type

```
uint8_t sl_wisun_params_mpl_s::seed_id_type
```

MPL seed-id type, 0 indicates the seed-id is elided and is the IPv6 source address, 3 indicates the seed-id is the 128-bit ULA/GUA of the source.

seed_set_entry_lifetime_s

```
uint16_t sl_wisun_params_mpl_s::seed_set_entry_lifetime_s
```

MPL seed set entry lifetime (seconds)

trickle

```
sl_wisun_trickle_params_t sl_wisun_params_mpl_s::trickle
```

MPL trickle timer.

trickle_expirations

```
uint8_t sl_wisun_params_mpl_s::trickle_expirations
```

MPL trickle timer expirations.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_rpl_s

RPL parameter set.

Public Attributes

uint16_t	address_registration_lifetime_s Amount of time a parent should retain a registered GUA/ULA (seconds)
uint8_t	candidate_parents_max RPL max candidate parents.
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	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.
uint16_t	init_parent_selection_s Delay for preferred parent selection after first DIO reception (seconds)
uint8_t	parents_max RPL max parents.

Public Attribute Documentation

address_registration_lifetime_s

```
uint16_t sl_wisun_params_rpl_s::address_registration_lifetime_s
```

Amount of time a parent should retain a registered GUA/ULA (seconds)

candidate_parents_max

```
uint8_t sl_wisun_params_rpl_s::candidate_parents_max
```

RPL max candidate parents.

dis_max_delay_first_s

```
uint16_t sl_wisun_params_rpl_s::dis_max_delay_first_s
```

RPL first DIS maximum delay (seconds)

Copyright © 2025 Silicon Laboratories. All rights reserved.

dis_max_delay_s

```
uint16_t sl_wisun_params_rpl_s::dis_max_delay_s
```

RPL DIS maximum delay (seconds)

etx_probe_period_max_s

```
uint16_t sl_wisun_params_rpl_s::etx_probe_period_max_s
```

Maximum period of NS probes used to get samples for ETX calculation (seconds)

etx_samples_init

```
uint8_t sl_wisun_params_rpl_s::etx_samples_init
```

Number of samples used to calculate ETX during join state 4.

etx_samples_refresh

```
uint8_t sl_wisun_params_rpl_s::etx_samples_refresh
```

Number of samples used to refresh ETX.

init_parent_selection_s

```
uint16_t sl_wisun_params_rpl_s::init_parent_selection_s
```

Delay for preferred parent selection after first DIO reception (seconds)

parents_max

```
uint8_t sl_wisun_params_rpl_s::parents_max
```

RPL max parents.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_params_traffic_s

Traffic parameter set.

Public Attributes

uint16_t	ipv6_mru Maximum Receive Unit (MRU) for fragmented IPv6 packets in bytes. A larger packet will be silently discarded.
uint16_t	lowpan_mtu Maximum Transmission Unit (MTU) for 6LoWPAN packets in bytes. A larger packet will be fragmented using 6LoWPAN fragmentation.
uint8_t	max_edfe_fragment_count Maximum number of 6LoWPAN/IPv6 fragments to send in a single EDFE transaction (0 to 10). 0 disables EDFE for fragmented packets.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

ipv6_mru

```
uint16_t sl_wisun_params_traffic_s::ipv6_mru
```

Maximum Receive Unit (MRU) for fragmented IPv6 packets in bytes. A larger packet will be silently discarded.

lowpan_mtu

```
uint16_t sl_wisun_params_traffic_s::lowpan_mtu
```

Maximum Transmission Unit (MTU) for 6LoWPAN packets in bytes. A larger packet will be fragmented using 6LoWPAN fragmentation.

max_edfe_fragment_count

```
uint8_t sl_wisun_params_traffic_s::max_edfe_fragment_count
```

Maximum number of 6LoWPAN/IPv6 fragments to send in a single EDFE transaction (0 to 10). 0 disables EDFE for fragmented packets.

reserved

```
uint8_t sl_wisun_params_traffic_s::reserved[3]
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.
uint8_t	crc_type FSK CRC type (sl_wisun_crc_type_t)
uint16_t	number_of_channels Number of channels.
uint8_t	phy_mode_id PHY mode ID.
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.

channel_spacing_khz

```
uint16_t sl_wisun_phy_config_custom_fsk_t::channel_spacing_khz
```

Channel spacing in kHz.

crc_type

```
uint8_t sl_wisun_phy_config_custom_fsk_t::crc_type
```

FSK CRC type (sl_wisun_crc_type_t)

number_of_channels

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint16_t sl_wisun_phy_config_custom_fsk_t::number_of_channels

Number of channels.

phy_mode_id

uint8_t sl_wisun_phy_config_custom_fsk_t::phy_mode_id

PHY mode ID.

preamble_length

uint8_t sl_wisun_phy_config_custom_fsk_t::preamble_length

FSK preamble length in bits.

reserved

uint8_t sl_wisun_phy_config_custom_fsk_t::reserved[1]

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.
uint8_t	crc_type OFDM CRC type (sl_wisun_crc_type_t)
uint16_t	number_of_channels Number of channels.
uint8_t	phy_mode_id PHY mode ID.
uint8_t	reserved Reserved, set to zero.
uint8_t	stf_length STF length in number of symbols.

Public Attribute Documentation

ch0_frequency_khz

```
uint32_t sl_wisun_phy_config_custom_ofdm_t::ch0_frequency_khz
```

Ch0 center frequency in kHz.

channel_spacing_khz

```
uint16_t sl_wisun_phy_config_custom_ofdm_t::channel_spacing_khz
```

Channel spacing in kHz.

crc_type

```
uint8_t sl_wisun_phy_config_custom_ofdm_t::crc_type
```

OFDM CRC type (sl_wisun_crc_type_t)

number_of_channels

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint16_t sl_wisun_phy_config_custom_ofdm_t::number_of_channels

Number of channels.

phy_mode_id

uint8_t sl_wisun_phy_config_custom_ofdm_t::phy_mode_id

PHY mode ID.

reserved

uint8_t sl_wisun_phy_config_custom_ofdm_t::reserved[1]

Reserved, set to zero.

stf_length

uint8_t sl_wisun_phy_config_custom_ofdm_t::stf_length

STF length in number of symbols.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_phy_config_custom_oqpsk_t

Custom OQPSK PHY configuration.

Public Attributes

uint32_t	ch0_frequency_khz Ch0 center frequency in kHz.
uint16_t	channel_spacing_khz Channel spacing in kHz.
uint8_t	crc_type OFDM CRC type (sl_wisun_crc_type_t)
uint16_t	number_of_channels Number of channels.
uint8_t	phy_mode_id PHY mode ID.
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.

channel_spacing_khz

```
uint16_t sl_wisun_phy_config_custom_oqpsk_t::channel_spacing_khz
```

Channel spacing in kHz.

crc_type

```
uint8_t sl_wisun_phy_config_custom_oqpsk_t::crc_type
```

OFDM CRC type (sl_wisun_crc_type_t)

number_of_channels

Copyright © 2025 Silicon Laboratories. All rights reserved.

uint16_t sl_wisun_phy_config_custom_oqpsk_t::number_of_channels

Number of channels.

phy_mode_id

uint8_t sl_wisun_phy_config_custom_oqpsk_t::phy_mode_id

PHY mode ID.

preamble_length

uint8_t sl_wisun_phy_config_custom_oqpsk_t::preamble_length

OQPSK preamble length in bits.

reserved

uint8_t sl_wisun_phy_config_custom_oqpsk_t::reserved[1]

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_phy_config_explicit_t

Explicit PHY configuration.

Public Attributes

uint32_t	ch0_frequency_khz Ch0 center frequency in kHz.
uint8_t	channel_spacing Channel spacing (sl_wisun_channel_spacing_t)
uint16_t	number_of_channels Number of channels.
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.

channel_spacing

```
uint8_t sl_wisun_phy_config_explicit_t::channel_spacing
```

Channel spacing (sl_wisun_channel_spacing_t)

number_of_channels

```
uint16_t sl_wisun_phy_config_explicit_t::number_of_channels
```

Number of channels.

phy_mode_id

```
uint8_t sl_wisun_phy_config_explicit_t::phy_mode_id
```

PHY mode ID.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_phy_config_fan10_t

FAN1.0 PHY configuration.

Public Attributes

uint8_t	fec 1 if FEC is enabled, 0 if not
uint8_t	op_class Operating class (sl_wisun_operating_class_t)
uint8_t	op_mode Operating mode (sl_wisun_operating_mode_t)
uint8_t	reg_domain Regulatory domain (sl_wisun_regulatory_domain_t)

Public Attribute Documentation

fec

```
uint8_t sl_wisun_phy_config_fan10_t::fec
```

1 if FEC is enabled, 0 if not

op_class

```
uint8_t sl_wisun_phy_config_fan10_t::op_class
```

Operating class (sl_wisun_operating_class_t)

op_mode

```
uint8_t sl_wisun_phy_config_fan10_t::op_mode
```

Operating mode (sl_wisun_operating_mode_t)

reg_domain

```
uint8_t sl_wisun_phy_config_fan10_t::reg_domain
```

Regulatory domain (sl_wisun_regulatory_domain_t)

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_phy_config_fan11_t

FAN1.1 PHY configuration.

Public Attributes

uint8_t	chan_plan_id Channel plan ID.
uint8_t	phy_mode_id PHY mode ID.
uint8_t	reg_domain Regulatory domain (sl_wisun_regulatory_domain_t)

Public Attribute Documentation

chan_plan_id

```
uint8_t sl_wisun_phy_config_fan11_t::chan_plan_id
```

Channel plan ID.

phy_mode_id

```
uint8_t sl_wisun_phy_config_fan11_t::phy_mode_id
```

PHY mode ID.

reg_domain

```
uint8_t sl_wisun_phy_config_fan11_t::reg_domain
```

Regulatory domain (sl_wisun_regulatory_domain_t)

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_phy_config_ids_t

Explicit RAIL configuration.

Public Attributes

uint16_t	channel_id Channel ID.
uint8_t	phy_mode_id PHY mode ID.
uint16_t	protocol_id Protocol ID.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

channel_id

```
uint16_t sl_wisun_phy_config_ids_t::channel_id
```

Channel ID.

phy_mode_id

```
uint8_t sl_wisun_phy_config_ids_t::phy_mode_id
```

PHY mode ID.

protocol_id

```
uint16_t sl_wisun_phy_config_ids_t::protocol_id
```

Protocol ID.

reserved

```
uint8_t sl_wisun_phy_config_ids_t::reserved[3]
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_phy_config_t

PHY configuration.

Public Attributes

union sl_wisun_phy_config_t::@3	config Configuration.
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.
sl_wisun_phy_config_explicit_t	explicit_plan Configuration for SL_WISUN_PHY_CONFIG_EXPLICIT type.
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_ids_t	ids Configuration for SL_WISUN_PHY_CONFIG_IDS type.
uint32_t	type Configuration type (sl_wisun_phy_config_type_t)

Public Attribute Documentation

config

```
union sl_wisun_phy_config_t::@3 sl_wisun_phy_config_t::config
```

Configuration.

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.

custom_ofdm

```
sl_wisun_phy_config_custom_ofdm_t sl_wisun_phy_config_t::custom_ofdm
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Configuration for SL_WISUN_PHY_CONFIG_CUSTOM_OFDM type.

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.

explicit_plan

```
sl_wisun_phy_config_explicit_t sl_wisun_phy_config_t::explicit_plan
```

Configuration for SL_WISUN_PHY_CONFIG_EXPLICIT type.

fan10

```
sl_wisun_phy_config_fan10_t sl_wisun_phy_config_t::fan10
```

Configuration for SL_WISUN_PHY_CONFIG_FAN10 type.

fan11

```
sl_wisun_phy_config_fan11_t sl_wisun_phy_config_t::fan11
```

Configuration for SL_WISUN_PHY_CONFIG_FAN11 type.

ids

```
sl_wisun_phy_config_ids_t sl_wisun_phy_config_t::ids
```

Configuration for SL_WISUN_PHY_CONFIG_IDS type.

type

```
uint32_t sl_wisun_phy_config_t::type
```

Configuration type (sl_wisun_phy_config_type_t)

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_rate_stats_t

Adaptive rate algorithm rate stats.

Public Attributes

uint16_t	fail_count Number of tx without acknowledgement.
uint32_t	frame_duration_us Frame tx time.
int8_t	probe_limit Indicates if the rate can be probed.
uint8_t	reserved Reserved, set to zero.
uint32_t	sample_count Number of times the rate was selected.
uint8_t	skipped_sample_count Number of times this rate was skipped.
uint16_t	success_count Number of successful transmissions.
float	success_probability Estimated chance of success.
float	throughput Estimated troughput in kb/s.

Public Attribute Documentation

fail_count

```
uint16_t sl_wisun_rate_stats_t::fail_count
```

Number of tx without acknowledgement.

frame_duration_us

```
uint32_t sl_wisun_rate_stats_t::frame_duration_us
```

Frame tx time.

probe_limit

```
int8_t sl_wisun_rate_stats_t::probe_limit
```

Indicates if the rate can be probed.

Copyright © 2025 Silicon Laboratories. All rights reserved.

reserved

```
uint8_t sl_wisun_rate_stats_t::reserved[2]
```

Reserved, set to zero.

sample_count

```
uint32_t sl_wisun_rate_stats_t::sample_count
```

Number of times the rate was selected.

skipped_sample_count

```
uint8_t sl_wisun_rate_stats_t::skipped_sample_count
```

Number of times this rate was skipped.

success_count

```
uint16_t sl_wisun_rate_stats_t::success_count
```

Number of successful transmissions.

success_probability

```
float sl_wisun_rate_stats_t::success_probability
```

Estimated chance of success.

throughput

```
float sl_wisun_rate_stats_t::throughput
```

Estimated troughput in kb/s.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_rate_t

Adaptive rate algorithm rates.

Public Attributes

uint8_t	index Rate index on the rate list.
uint8_t	is_supported Indicates if rate is supported by both the node and the neighbor.
uint8_t	max_retries Maximum amount of retries with current rate.
uint8_t	phy_mode_id Rate phy mode id.
uint32_t	rate Rate in kb/s.
sl_wisun_rate_stats_t	stats Struct containig the rate's tx statistics.

Public Attribute Documentation

index

```
uint8_t sl_wisun_rate_t::index
```

Rate index on the rate list.

is_supported

```
uint8_t sl_wisun_rate_t::is_supported
```

Indicates if rate is supported by both the node and the neighbor.

max_retries

```
uint8_t sl_wisun_rate_t::max_retries
```

Maximum amount of retries with current rate.

phy_mode_id

```
uint8_t sl_wisun_rate_t::phy_mode_id
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Rate phy mode id.

rate

```
uint32_t sl_wisun_rate_t::rate
```

Rate in kb/s.

stats

```
sl_wisun_rate_stats_t sl_wisun_rate_t::stats
```

Struct containig the rate's tx statistics.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_rpl_info_t

RPL information.

Public Attributes

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)
uint8_t	default_lifetime Default lifetime, lifetime that is used as default for all RPL routes.
uint8_t	dio_interval_doublings DIO interval doublings, used to configure Imax of the DIO Trickle timer.
uint8_t	dio_interval_min DIO minimum interval, used to configure Imin of the DIO Trickle timer.
uint8_t	dio_redundancy_constant DIO redundancy constant, used to configure k of the DIO Trickle timer.
uint8_t	dodag_dtsn Destination Advertisement Trigger Sequence Number (DTSN)
uint8_t	dodag_preference DODAG Preference, defines how preferable the root of this DODAG is compared to other DODAG roots within the instance.
uint16_t	dodag_rank DODAG rank or the node.
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	instance_id Instance ID, set by the DODAG root, it indicates of which RPL Instance the DODAG is a part.
uint16_t	lifetime_unit Lifetime unit, unit in seconds that is used to express route lifetimes in RPL.
uint16_t	min_hop_rank_increase Min hop rank increase, minimum increase in Rank between a node and any of its DODAG parents.
uint8_t	mode_of_operation Mode of Operation (MOP), must be 1 for Non-Storing Mode of Operation.
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

dag_max_rank_increase

Copyright © 2025 Silicon Laboratories. All rights reserved.

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)

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.

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.

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.

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.

dodag_dtsn

uint8_t sl_wisun_rpl_info_t::dodag_dtsn

Destination Advertisement Trigger Sequence Number (DTSN)

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).

Copyright © 2025 Silicon Laboratories. All rights reserved.

dodag_rank

```
uint16_t sl_wisun_rpl_info_t::dodag_rank
```

DODAG rank or the node.

dodag_version_number

```
uint8_t sl_wisun_rpl_info_t::dodag_version_number
```

DODAG version number, set by the DODAG root.

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.

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.

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.

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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

reserved

```
uint8_t sl_wisun_rpl_info_t::reserved[2]
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_option_data_t

Socket options Deprecated since v2.0 MUST NOT be used with POSIX socket API.

Public Attributes

sl_wisun_socket_option_edfe_mode_t	edfe_mode Socket EDFE mode Deprecated.
sl_wisun_socket_option_event_mode_t	event_mode Socket event mode Deprecated.
in6_addr_t	ipv6_address IPv6 address.
sl_wisun_socket_option_multicast_group_t	multicast_group Socket multicast group Deprecated.
sl_wisun_socket_option_multicast_hop_limit	multicast_hop_limit Socket multicast hop limit Deprecated.
sl_wisun_socket_option_send_buffer_limit_t	send_buffer_limit Socket send buffer limit Deprecated.
sl_wisun_socket_option_unicast_hop_limit	unicast_hop_limit Socket unicast hop limit Deprecated.
int32_t	value Option-specific value.

Public Attribute Documentation

edfe_mode

```
sl_wisun_socket_option_edfe_mode_t sl_wisun_socket_option_data_t::edfe_mode
```

Socket EDFE mode Deprecated.

event_mode

```
sl_wisun_socket_option_event_mode_t sl_wisun_socket_option_data_t::event_mode
```

Socket event mode Deprecated.

ipv6_address

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
in6_addr_t sl_wisun_socket_option_data_t::ipv6_address
```

IPv6 address.

multicast_group

```
sl_wisun_socket_option_multicast_group_t sl_wisun_socket_option_data_t::multicast_group
```

Socket multicast group Deprecated.

multicast_hop_limit

```
sl_wisun_socket_option_multicast_hop_limit sl_wisun_socket_option_data_t::multicast_hop_limit
```

Socket multicast hop limit Deprecated.

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.

unicast_hop_limit

```
sl_wisun_socket_option_unicast_hop_limit sl_wisun_socket_option_data_t::unicast_hop_limit
```

Socket unicast hop limit Deprecated.

value

```
int32_t sl_wisun_socket_option_data_t::value
```

Option-specific value.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_option_edfe_mode_t

Socket option for EDFE mode Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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)

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_option_event_mode_t

Socket option for event mode Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_option_multicast_group_t

Socket option for multicast group Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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.

address

```
sl_wisun_ip_address_t sl_wisun_socket_option_multicast_group_t::address
```

Multicast group address.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_option_multicast_hop_limit

Socket option for socket multicast hop limit Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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)

reserved

```
uint16_t sl_wisun_socket_option_multicast_hop_limit::reserved
```

Reserved, set to 0.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_option_send_buffer_limit_t

Socket option for send buffer limit Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_socket_option_unicast_hop_limit

Socket option for socket unicast hop limit Deprecated since v2.0 MUST NOT be used with POSIX socket API.

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)

reserved

```
uint16_t sl_wisun_socket_option_unicast_hop_limit::reserved
```

Reserved, set to 0.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.
int16_t	prev_avg_synch_fix Deprecated.
uint16_t	synch_interval Number of seconds since last timing information from the parent.
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.

hop_count

```
uint16_t sl_wisun_statistics_fhss_t::hop_count
```

Estimated number of hops to the border router based on RPL rank.

prev_avg_synch_fix

```
int16_t sl_wisun_statistics_fhss_t::prev_avg_synch_fix
```

Deprecated.

synch_interval

```
uint16_t sl_wisun_statistics_fhss_t::synch_interval
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Number of seconds since last timing information from the parent.

synch_lost

> uint32_t sl_wisun_statistics_fhss_t::synch_lost

Deprecated.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_statistics_heap_t

Heap usage statistics.

Public Attributes

uint32_t	arena Highest heap usage in bytes.
uint32_t	free Current remaining heap in bytes.
uint32_t	total Total available heap in bytes.
uint32_t	uordblks Current heap usage in bytes.

Public Attribute Documentation

arena

```
uint32_t sl_wisun_statistics_heap_t::arena
```

Highest heap usage in bytes.

free

```
uint32_t sl_wisun_statistics_heap_t::free
```

Current remaining heap in bytes.

total

```
uint32_t sl_wisun_statistics_heap_t::total
```

Total available heap in bytes.

uordblks

```
uint32_t sl_wisun_statistics_heap_t::uordblks
```

Current heap usage in bytes.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_statistics_mac_t

MAC statistics.

Public Attributes

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	cca_attempts_count Number of MAC CCA attempts.
uint32_t	failed_cca_count Number of failed MAC transmissions due to CCA.
uint32_t	idle_duration_s Cumulated idle duration in seconds.
uint32_t	radio_tx_duration_ms Radio TX duration in milliseconds.
uint8_t	reserved Reserved, set to zero.
uint32_t	retry_count Number of retried MAC transmissions.
uint8_t	rx_availability_percentage MAC RX availability percentage.
uint32_t	rx_bytes Amount of successfully received MAC data in bytes. FCS is not included.
uint32_t	rx_count Number of successfully received MAC frames.
uint32_t	rx_drop_count Number of successfully received MAC frames discarded during processing.
uint32_t	rx_ms_count Number of successfully received 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_bytes Amount of transmitted MAC data in bytes. FCS is not included.
uint32_t	tx_count Number of transmitted MAC frames.
uint32_t	tx_failed_count Number of failed MAC transmissions.
uint32_t	tx_ms_count Number of transmitted MAC frames using mode switch.
uint32_t	tx_ms_failed_count

Copyright © 2025 Silicon Laboratories. All rights reserved.

Number of failed MAC frames transmission using mode switch.

uint16_t tx_queue_peak
Highest number of frames in the MAC transmission queue.

uint16_t tx_queue_size
Current number of frames in the MAC transmission queue.

Public Attribute Documentation

bc_rx_count

```
uint32_t sl_wisun_statistics_mac_t::bc_rx_count
```

Number of successfully received broadcast MAC frames.

bc_tx_count

```
uint32_t sl_wisun_statistics_mac_t::bc_tx_count
```

Number of transmitted broadcast MAC frames.

cca_attempts_count

```
uint32_t sl_wisun_statistics_mac_t::cca_attempts_count
```

Number of MAC CCA attempts.

failed_cca_count

```
uint32_t sl_wisun_statistics_mac_t::failed_cca_count
```

Number of failed MAC transmissions due to CCA.

idle_duration_s

```
uint32_t sl_wisun_statistics_mac_t::idle_duration_s
```

Cumulated idle duration in seconds.

radio_tx_duration_ms

```
uint32_t sl_wisun_statistics_mac_t::radio_tx_duration_ms
```

Radio TX duration in milliseconds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

reserved

```
uint8_t sl_wisun_statistics_mac_t::reserved[3]
```

Reserved, set to zero.

retry_count

```
uint32_t sl_wisun_statistics_mac_t::retry_count
```

Number of retried MAC transmissions.

rx_availability_percentage

```
uint8_t sl_wisun_statistics_mac_t::rx_availability_percentage
```

MAC RX availability percentage.

rx_bytes

```
uint32_t sl_wisun_statistics_mac_t::rx_bytes
```

Amount of successfully received MAC data in bytes. FCS is not included.

rx_count

```
uint32_t sl_wisun_statistics_mac_t::rx_count
```

Number of successfully received MAC frames.

rx_drop_count

```
uint32_t sl_wisun_statistics_mac_t::rx_drop_count
```

Number of successfully received MAC frames discarded during processing.

rx_ms_count

```
uint32_t sl_wisun_statistics_mac_t::rx_ms_count
```

Number of successfully received MAC frames using mode_switch.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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).

tx_bytes

```
uint32_t sl_wisun_statistics_mac_t::tx_bytes
```

Amount of transmitted MAC data in bytes. FCS is not included.

tx_count

```
uint32_t sl_wisun_statistics_mac_t::tx_count
```

Number of transmitted MAC frames.

tx_failed_count

```
uint32_t sl_wisun_statistics_mac_t::tx_failed_count
```

Number of failed MAC transmissions.

tx_ms_count

```
uint32_t sl_wisun_statistics_mac_t::tx_ms_count
```

Number of transmitted MAC frames using mode switch.

tx_ms_failed_count

```
uint32_t sl_wisun_statistics_mac_t::tx_ms_failed_count
```

Number of failed MAC frames transmission using mode switch.

tx_queue_peak

```
uint16_t sl_wisun_statistics_mac_t::tx_queue_peak
```

Highest number of frames in the MAC transmission queue.

Copyright © 2025 Silicon Laboratories. All rights reserved.

tx_queue_size

```
uint16_t sl_wisun_statistics_mac_t::tx_queue_size
```

Current number of frames in the MAC transmission queue.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_statistics_network_t

6LoWPAN/IP stack statistics

Public Attributes

uint16_t	adapt_layer_tx_queue_peak Highest number of frames in the adaptation layer transmission queue.
uint16_t	adapt_layer_tx_queue_size Current number of frames in the adaptation layer transmission queue.
uint32_t	buf_alloc Number of data buffer allocation attempts.
uint32_t	buf_headroom_fail Number of times data buffer resizing has failed.
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.
uint16_t	etx_1st_parent ETX of the primary parent.
uint16_t	etx_2nd_parent ETX of the secondary parent.
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	ip_cksum_error Number of discarded IPv6 packets due to a checksum error.
uint32_t	ip_no_route Number of discarded IPv6 packets due to lack routing information.
uint32_t	ip_routed_up Amount of forwarded IPv6 data in bytes.
uint32_t	ip_routeloop_detect Number of RPL packet forwarding errors due to inconsistent routing information.
uint32_t	ip_rx_bytes Amount received IPv6 data in bytes.
uint32_t	ip_rx_count Number of received IP6 packets.
uint32_t	ip_rx_drop Number of discarded IPv6 packets during processing.
uint32_t	ip_tx_bytes Amount of transmitted IPv6 data in bytes.
uint32_t	ip_tx_count

Copyright © 2025 Silicon Laboratories. All rights reserved.

Number of transmitted IPv6 packets.

uint16_t	mpl_forwarded_messages_count	Number of forwarded MPL messages.
uint16_t	mpl_freed_messages_count	Number of freed MPL messages.
uint16_t	mpl_new_messages_count	Number of new MPL messages.
uint16_t	mpl_not_tx_count	Number of deleted MPL messages that were never sent.
uint16_t	mpl_rx_count	Number of received MPL messages.
uint32_t	rpl_global_repair	Number of times a global repair has been triggered by the border router.
uint32_t	rpl_local_repair	Number of times a local repair procedure has been triggered by the node.
uint32_t	rpl_malformed_message	Number of discarded RPL packets due to malformed content.
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_route_routecost_better_change	Number of RPL parent changes due to better route cost.
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	rpl_unknown_instance	Number of discarded RPL packets due to an unknown DODAG instance.

Public Attribute Documentation

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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

buf_alloc

```
uint32_t sl_wisun_statistics_network_t::buf_alloc
```

Number of data buffer allocation attempts.

buf_headroom_fail

```
uint32_t sl_wisun_statistics_network_t::buf_headroom_fail
```

Number of times data buffer resizing has failed.

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.

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.

etx_1st_parent

```
uint16_t sl_wisun_statistics_network_t::etx_1st_parent
```

ETX of the primary parent.

etx_2nd_parent

```
uint16_t sl_wisun_statistics_network_t::etx_2nd_parent
```

ETX of the secondary parent.

frag_rx_errors

```
uint32_t sl_wisun_statistics_network_t::frag_rx_errors
```

Number of fragmentation errors in received IPv6 packets.

Copyright © 2025 Silicon Laboratories. All rights reserved.

frag_tx_errors

uint32_t sl_wisun_statistics_network_t::frag_tx_errors

Number of fragmentation errors in transmitted IPv6 packets.

ip_cksum_error

uint32_t sl_wisun_statistics_network_t::ip_cksum_error

Number of discarded IPv6 packets due to a checksum error.

ip_no_route

uint32_t sl_wisun_statistics_network_t::ip_no_route

Number of discarded IPv6 packets due to lack routing information.

ip_routed_up

uint32_t sl_wisun_statistics_network_t::ip_routed_up

Amount of forwarded IPv6 data in bytes.

ip_routeloop_detect

uint32_t sl_wisun_statistics_network_t::ip_routeloop_detect

Number of RPL packet forwarding errors due to inconsistent routing information.

ip_rx_bytes

uint32_t sl_wisun_statistics_network_t::ip_rx_bytes

Amount received IPv6 data in bytes.

ip_rx_count

uint32_t sl_wisun_statistics_network_t::ip_rx_count

Number of received IP6 packets.

Copyright © 2025 Silicon Laboratories. All rights reserved.

ip_rx_drop

```
uint32_t sl_wisun_statistics_network_t::ip_rx_drop
```

Number of discarded IPv6 packets during processing.

ip_tx_bytes

```
uint32_t sl_wisun_statistics_network_t::ip_tx_bytes
```

Amount of transmitted IPv6 data in bytes.

ip_tx_count

```
uint32_t sl_wisun_statistics_network_t::ip_tx_count
```

Number of transmitted IPv6 packets.

mpl_forwarded_messages_count

```
uint16_t sl_wisun_statistics_network_t::mpl_forwarded_messages_count
```

Number of forwarded MPL messages.

mpl_freed_messages_count

```
uint16_t sl_wisun_statistics_network_t::mpl_freed_messages_count
```

Number of freed MPL messages.

mpl_new_messages_count

```
uint16_t sl_wisun_statistics_network_t::mpl_new_messages_count
```

Number of new MPL messages.

mpl_not_tx_count

```
uint16_t sl_wisun_statistics_network_t::mpl_not_tx_count
```

Number of deleted MPL messages that were never sent.

Copyright © 2025 Silicon Laboratories. All rights reserved.

mpl_rx_count

```
uint16_t sl_wisun_statistics_network_t::mpl_rx_count
```

Number of received MPL messages.

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.

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.

rpl_malformed_message

```
uint32_t sl_wisun_statistics_network_t::rpl_malformed_message
```

Number of discarded RPL packets due to malformed content.

rpl_memory_overflow

```
uint32_t sl_wisun_statistics_network_t::rpl_memory_overflow
```

Sum of RPL object sizes that have failed allocation in bytes.

rpl_parent_tx_fail

```
uint32_t sl_wisun_statistics_network_t::rpl_parent_tx_fail
```

Number of failed RPL transmissions to the parent.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

rpl_time_no_next_hop

```
uint32_t sl_wisun_statistics_network_t::rpl_time_no_next_hop
```

Number of seconds without an RPL parent.

rpl_total_memory

```
uint32_t sl_wisun_statistics_network_t::rpl_total_memory
```

Amount of memory currently allocated for RPL objects in bytes.

rpl_unknown_instance

```
uint32_t sl_wisun_statistics_network_t::rpl_unknown_instance
```

Number of discarded RPL packets due to an unknown DODAG instance.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_statistics_phy_t

PHY/RF statistics.

Public Attributes

uint32_t	crc_fails Number of CRC failures on reception.
uint32_t	rx_timeouts Number of reception timeouts.
uint32_t	tx_timeouts Number of transmission timeouts.

Public Attribute Documentation

crc_fails

```
uint32_t sl_wisun_statistics_phy_t::crc_fails
```

Number of CRC failures on reception.

rx_timeouts

```
uint32_t sl_wisun_statistics_phy_t::rx_timeouts
```

Number of reception timeouts.

tx_timeouts

```
uint32_t sl_wisun_statistics_phy_t::tx_timeouts
```

Number of transmission timeouts.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_statistics_regulation_t

Regional regulation statistics.

Public Attributes

struct sl_wisun_statistics_regulation_t::@1	@2 Regional regulation statistics.
sl_wisun_statistics_arib_regulation_t	arib ARIB statistics.
uint32_t	duty_cycle_alert_threshold_ms Duty cycle alert threshold in milliseconds.
uint32_t	duty_cycle_alert_threshold_per_channel_ms Duty cycle alert threshold per channel in milliseconds.
uint32_t	duty_cycle_warning_threshold_ms Duty cycle warning threshold in milliseconds.
uint32_t	duty_cycle_warning_threshold_per_channel_ms Duty cycle warning threshold per channel in milliseconds.
uint32_t	tx_duration_ms Sum of transmission durations during the last hour in milliseconds.
uint32_t	tx_duration_per_channel_ms Sum of transmission durations per channel during the last hour in milliseconds.

Public Attribute Documentation

@2

```
struct sl_wisun_statistics_regulation_t::@1 sl_wisun_statistics_regulation_t::@2
```

Regional regulation statistics.

arib

```
sl_wisun_statistics_arib_regulation_t sl_wisun_statistics_regulation_t::arib
```

ARIB statistics.

duty_cycle_alert_threshold_ms

```
uint32_t sl_wisun_statistics_regulation_t::duty_cycle_alert_threshold_ms
```

Duty cycle alert threshold in milliseconds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

duty_cycle_alert_threshold_per_channel_ms

```
uint32_t sl_wisun_statistics_regulation_t::duty_cycle_alert_threshold_per_channel_ms
```

Duty cycle alert threshold per channel in milliseconds.

duty_cycle_warning_threshold_ms

```
uint32_t sl_wisun_statistics_regulation_t::duty_cycle_warning_threshold_ms
```

Duty cycle warning threshold in milliseconds.

duty_cycle_warning_threshold_per_channel_ms

```
uint32_t sl_wisun_statistics_regulation_t::duty_cycle_warning_threshold_per_channel_ms
```

Duty cycle warning threshold per channel in milliseconds.

tx_duration_ms

```
uint32_t sl_wisun_statistics_regulation_t::tx_duration_ms
```

Sum of transmission durations during the last hour in milliseconds.

tx_duration_per_channel_ms

```
uint32_t sl_wisun_statistics_regulation_t::tx_duration_per_channel_ms
```

Sum of transmission durations per channel during the last hour in milliseconds.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_statistics_t

Statistics.

Public Attributes

sl_wisun_statistics_fhss_t	fhss Frequency hopping statistics.
sl_wisun_statistics_heap_t	heap Heap usage statistics.
sl_wisun_statistics_mac_t	mac MAC statistics.
sl_wisun_statistics_network_t	network 6LoWPAN/IP stack statistics
sl_wisun_statistics_phy_t	phy PHY/RF statistics.
sl_wisun_statistics_regulation_t	regulation Regional regulation statistics.
sl_wisun_statistics_wisun_t	wisun Wi-SUN statistics.

Public Attribute Documentation

fhss

```
sl_wisun_statistics_fhss_t sl_wisun_statistics_t::fhss
```

Frequency hopping statistics.

heap

```
sl_wisun_statistics_heap_t sl_wisun_statistics_t::heap
```

Heap usage statistics.

mac

```
sl_wisun_statistics_mac_t sl_wisun_statistics_t::mac
```

MAC statistics.

network

Copyright © 2025 Silicon Laboratories. All rights reserved.

> sl_wisun_statistics_network_t sl_wisun_statistics_t::network

6LoWPAN/IP stack statistics

phy

> sl_wisun_statistics_phy_t sl_wisun_statistics_t::phy

PHY/RF statistics.

regulation

> sl_wisun_statistics_regulation_t sl_wisun_statistics_t::regulation

Regional regulation statistics.

wisun

> sl_wisun_statistics_wisun_t sl_wisun_statistics_t::wisun

Wi-SUN statistics.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

pan_control_tx_count

```
uint32_t sl_wisun_statistics_wisun_t::pan_control_tx_count
```

Number of completed PAN control transmission requests.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_trickle_params_t

Trickle parameter set.

Public Attributes

uint16_t	imax_s Maximum interval size (seconds)
uint16_t	imin_s Minimum interval size (seconds)
uint8_t	k Redundancy constant (0 for infinity)
uint8_t	reserved Reserved, set to zero.

Public Attribute Documentation

imax_s

```
uint16_t sl_wisun_trickle_params_t::imax_s
```

Maximum interval size (seconds)

imin_s

```
uint16_t sl_wisun_trickle_params_t::imin_s
```

Minimum interval size (seconds)

k

```
uint8_t sl_wisun_trickle_params_t::k
```

Redundancy constant (0 for infinity)

reserved

```
uint8_t sl_wisun_trickle_params_t::reserved[3]
```

Reserved, set to zero.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_CERTIF Profile for certification testing.
const sl_wisun_connection_params_t	SL_WISUN_PARAMS_PROFILE_LARGE Profile for a large 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_SMALL Profile for a small network.
const sl_wisun_connection_params_t	SL_WISUN_PARAMS_PROFILE_TEST Profile for development (shorter connection time)

Variable Documentation

SL_WISUN_PARAMS_PROFILE_CERTIF

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_CERTIF
```

Profile for certification testing.

SL_WISUN_PARAMS_PROFILE_LARGE

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_LARGE
```

Profile for a large network.

SL_WISUN_PARAMS_PROFILE_MEDIUM

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_MEDIUM
```

Profile for a medium network.

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_WISUN_PARAMS_PROFILE_SMALL

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_SMALL
```

Profile for a small network.

SL_WISUN_PARAMS_PROFILE_TEST

```
const sl_wisun_connection_params_t SL_WISUN_PARAMS_PROFILE_TEST
```

Profile for development (shorter connection time)

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_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.
const sl_wisun_lfn_params_t	SL_WISUN_PARAMS_LFN_TEST Profile for test usage, best performance but highest power consumption.

Variable Documentation

SL_WISUN_PARAMS_LFN_BALANCED

```
const sl_wisun_lfn_params_t SL_WISUN_PARAMS_LFN_BALANCED
```

Profile providing balance between power consumption and performance.

SL_WISUN_PARAMS_LFN_ECO

```
const sl_wisun_lfn_params_t SL_WISUN_PARAMS_LFN_ECO
```

Profile optimized for low power consumption.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Predefined BR parameter sets

Predefined BR parameter sets

Predefined BR parameter sets for sl_wisun_br_set_connection_parameters().

These parameter sets can be used either as-is to replicate the behavior of sl_wisun_br_set_network_size(), or used as an initialization value for an application-specific parameter set.

Variables

const sl_wisun_br_connection_params_t	SL_WISUN_BR_PARAMS_PROFILE_LARGE Profile for a large network.
const sl_wisun_br_connection_params_t	SL_WISUN_BR_PARAMS_PROFILE_MEDIUM Profile for a medium network.
const sl_wisun_br_connection_params_t	SL_WISUN_BR_PARAMS_PROFILE_SMALL Profile for a small network.
const sl_wisun_br_connection_params_t	SL_WISUN_BR_PARAMS_PROFILE_TEST Profile for development (shorter connection time)

Variable Documentation

SL_WISUN_BR_PARAMS_PROFILE_LARGE

```
const sl_wisun_br_connection_params_t SL_WISUN_BR_PARAMS_PROFILE_LARGE
```

Profile for a large network.

SL_WISUN_BR_PARAMS_PROFILE_MEDIUM

```
const sl_wisun_br_connection_params_t SL_WISUN_BR_PARAMS_PROFILE_MEDIUM
```

Profile for a medium network.

SL_WISUN_BR_PARAMS_PROFILE_SMALL

```
const sl_wisun_br_connection_params_t SL_WISUN_BR_PARAMS_PROFILE_SMALL
```

Profile for a small network.

SL_WISUN_BR_PARAMS_PROFILE_TEST

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
const sl_wisun_br_connection_params_t SL_WISUN_BR_PARAMS_PROFILE_TEST
```

Profile for development (shorter connection time)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Predefined BR LFN-parenting parameter sets

Predefined BR LFN-parenting parameter sets

Predefined LFN parameter sets for sl_wisun_br_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_br_lfn_params_t	SL_WISUN_BR_PARAMS_LFN_BALANCED Profile providing balance between battery life and performance.
const sl_wisun_br_lfn_params_t	SL_WISUN_BR_PARAMS_LFN_ECO Profile with a higher battery life.
const sl_wisun_br_lfn_params_t	SL_WISUN_BR_PARAMS_LFN_TEST Profile for use during tests.

Variable Documentation

SL_WISUN_BR_PARAMS_LFN_BALANCED

```
const sl_wisun_br_lfn_params_t SL_WISUN_BR_PARAMS_LFN_BALANCED
```

Profile providing balance between battery life and performance.

SL_WISUN_BR_PARAMS_LFN_ECO

```
const sl_wisun_br_lfn_params_t SL_WISUN_BR_PARAMS_LFN_ECO
```

Profile with a higher battery life.

SL_WISUN_BR_PARAMS_LFN_TEST

```
const sl_wisun_br_lfn_params_t SL_WISUN_BR_PARAMS_LFN_TEST
```

Profile for use during tests.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN API availability

Wi-SUN API availability

Availability of Wi-SUN API capabilities depends on the library used by the application:

- FFN library contains Full Function Node capability. This library also includes LFN parenting ability for FAN1.1.
- LFN library contains Limited Function Node capability.
- Full library contains both of the above. This larger but more flexible library is provided as a convenience for prototyping. This library allows to choose between FFN and LFN role at runtime, but does not support switching roles on the fly and requires a full board reset to do so.
- BR library contains Border Router functionality.

The libraries are selected using Simplicity SDK components:

- FFN library when only wisun_stack_ffn component is selected.
- LFN library when only wisun_stack_lfn component is selected.
- Full library when both components are selected.
- FFN library by default when no component is selected.
- BR library when wisun_br_stack component is selected. This library cannot be combined with the other libraries.

The API available to each library may vary according to the table below. A status SL_STATUS_NOT_AVAILABLE is returned if the API called with a library flavor that does not support it.

API Name	Full	FFN	LFN	BR
sl_wisun_join	✅	✅	✅	❌
sl_wisun_get_ip_address	✅	✅	✅	❌
sl_wisun_disconnect	✅	✅	✅	❌
sl_wisun_set_trusted_certificate	✅	✅	✅	✅
sl_wisun_set_device_certificate	✅	✅	✅	❌
sl_wisun_set_device_private_key	✅	✅	✅	✅
sl_wisun_get_statistics	✅	✅	✅	✅
sl_wisun_set_tx_power	✅	✅	✅	✅
sl_wisun_set_allowed_channel_mask	✅	✅	✅	✅
sl_wisun_set_channel_mask	✅	✅	✅	✅
sl_wisun_allow_mac_address	✅	✅	✅	✅
sl_wisun_deny_mac_address	✅	✅	✅	✅
sl_wisun_get_join_state	✅	✅	✅	❌
sl_wisun_clear_credential_cache	✅	✅	✅	✅
sl_wisun_get_mac_address	✅	✅	✅	✅
sl_wisun_set_mac_address	✅	✅	✅	✅
sl_wisun_reset_statistics	✅	✅	✅	✅
sl_wisun_get_neighbor_count	✅	✅	✅	✅
sl_wisun_get_neighbors	✅	✅	✅	✅

Copyright © 2025 Silicon Laboratories. All rights reserved.

API Name	Full	FFN	LFN	BR
sl_wisun_get_neighbor_info	✅	✅	✅	✅
sl_wisun_set_unicast_settings	✅	✅	❌	✅
sl_wisun_set_device_private_key_id	✅	✅	✅	✅
sl_wisun_set_regulation	✅	✅	✅	✅
sl_wisun_set_regulation_parameters	✅	✅	✅	✅
sl_wisun_reset_regulation_duty_cycle	✅	✅	✅	✅
sl_wisun_config_neighbor_table	✅	✅	✅	✅
sl_wisun_set_regulation_tx_thresholds	✅	✅	✅	✅
sl_wisun_set_advert_fragment_duration	✅	✅	❌	✅
sl_wisun_set_device_type	✅	✅	✅	✅
sl_wisun_config_mode_switch	✅	✅	✅	✅
sl_wisun_set_mode_switch	✅	✅	✅	✅
sl_wisun_set_connection_parameters	✅	✅	❌	❌
sl_wisun_set_pom_ie	✅	✅	✅	✅
sl_wisun_get_pom_ie	✅	✅	✅	✅
sl_wisun_get_stack_version	✅	✅	✅	✅
sl_wisun_set_lfn_parameters	✅	❌	✅	❌
sl_wisun_set_lfn_support	✅	✅	❌	❌
sl_wisun_set_pti_state	✅	✅	✅	✅
sl_wisun_trigger_frame	✅	✅	✅	✅
sl_wisun_set_security_state	✅	✅	✅	✅
sl_wisun_get_network_info	✅	✅	✅	✅
sl_wisun_get_rpl_info	✅	✅	❌	✅
sl_wisun_get_excluded_channel_mask	✅	✅	✅	✅
sl_wisun_set_tx_power_ddbm	✅	✅	✅	✅
sl_wisun_set_leaf	✅	✅	❌	❌
sl_wisun_set_direct_connect_state	✅	✅	❌	❌
sl_wisun_accept_direct_connect_link	✅	✅	❌	❌
sl_wisun_set_direct_connect_pmk	✅	✅	❌	❌
sl_wisun_set_phy_sensitivity	✅	✅	✅	✅
sl_wisun_set_preferred_pan	✅	✅	❌	❌
sl_wisun_set_eap_identity	✅	✅	✅	❌
sl_wisun_br_start	❌	❌	❌	✅
sl_wisun_br_stop	❌	❌	❌	✅
sl_wisun_br_set_gtk	❌	❌	❌	✅
sl_wisun_br_set_network_size	❌	❌	❌	✅
sl_wisun_br_get_ip_addresses	❌	❌	❌	✅

Copyright © 2025 Silicon Laboratories. All rights reserved.

API Name	Full	FFN	LFN	BR
sl_wisun_br_get_gak	❌	❌	❌	✅
sl_wisun_br_get_state	❌	❌	❌	✅
sl_wisun_br_set_device_certificate	❌	❌	❌	✅
sl_wisun_br_set_broadcast_settings	❌	❌	❌	✅
sl_wisun_br_set_connection_parameters	❌	❌	❌	✅
sl_wisun_br_set_lfn_parameters	❌	❌	❌	✅
sl_wisun_br_set_lfn_support	❌	❌	❌	✅
sl_wisun_br_pan_defect_advertise	❌	❌	❌	✅
sl_wisun_br_set_ipv6_prefix	❌	❌	❌	✅
sl_wisun_br_set_ipv6_up_handler	❌	❌	❌	✅
sl_wisun_br_ipv6_down	❌	❌	❌	✅

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN Regional Regulation API

Wi-SUN Regional Regulation API

Wi-SUN Regional Regulation API.

This API provides the parameters for regional regulation compliance. It allows to set the duty cycle budget, thresholds for warnings and alerts, and other parameters related to the regional regulation. The Wi-SUN stack provides predefined parameter sets for the following regulations:

- ARIB (Japan)
- WPC (India)
- ETSI (Europe)

The regional regulation is configured with sl_wisun_set_regulation_parameters(). Predefined parameter sets can be used for common regulatory requirements, they can be selected with sl_wisun_set_regulation().

Special attention must be taken with frame length when ARIB regulation is used in Japan. The maximum length of a radio frame is indirectly constrained by the channel bandwidth. This is because the regulation imposes a maximum transmission duration per frame, which varies depending on the bandwidth used:

- with a 200 kHz bandwidth, frame duration must be less than 400 ms
- with a 400 kHz bandwidth, frame duration must be less than 200 ms
- with 600 and 800 kHz bandwidths, frame duration must be less than 100 ms

The Wi-SUN stack doesn't currently enforce these constraints, and defers the responsibility to the application. Depending on the selected PHY, the application must ensure that the transmission duration complies with the ARIB regulation by:

- reducing the frame length
- adjusting the 6LoWPAN MTU leading to fragmentation for larger packets

Modules

sl_wisun_regulation_params_t

Predefined regional regulation parameter sets

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_regulation_params_t

Regional regulation parameter set.

Public Attributes

int8_t	duty_cycle_alert_threshold_percent Duty cycle alert threshold in percent, -1 to disable duty cycle alert threshold.
uint8_t	duty_cycle_arib_skip_ack Skip responses from sum of transmissions (ARIB conditions)
int16_t	duty_cycle_budget_per_channel_s Duty cycle budget per channel in seconds per hour, -1 to disable duty cycle budget per channel.
int16_t	duty_cycle_budget_s Duty cycle budget in seconds per hour, -1 to disable duty cycle budget.
uint8_t	duty_cycle_skip_ack Skip ACKs from sum of transmissions.
int8_t	duty_cycle_warning_threshold_percent Duty cycle warning threshold in percent, -1 to disable duty cycle warning threshold.
uint8_t	enable_apc Enable Adaptive Power Control.
uint8_t	enable_arib_channel_access Ensure a min IFS before ACKs (ARIB conditions)
int8_t	fixed_cca_threshold_dbm Fixed CCA threshold in dBm, INT8_MAX to use adaptive CCA threshold.
uint32_t	max_edfe_duration_ms Maximum Extended Directed Frame Exchange (EDFE) duration in milliseconds, 0 to forbid EDFE.
uint8_t	reserved Reserved, set to zero.
uint32_t	version Version of this API.

Public Attribute Documentation

duty_cycle_alert_threshold_percent

```
int8_t sl_wisun_regulation_params_t::duty_cycle_alert_threshold_percent
```

Duty cycle alert threshold in percent, -1 to disable duty cycle alert threshold.

duty_cycle_arib_skip_ack

```
uint8_t sl_wisun_regulation_params_t::duty_cycle_arib_skip_ack
```

Skip responses from sum of transmissions (ARIB conditions)

Copyright © 2025 Silicon Laboratories. All rights reserved.

duty_cycle_budget_per_channel_s

```
int16_t sl_wisun_regulation_params_t::duty_cycle_budget_per_channel_s
```

Duty cycle budget per channel in seconds per hour, -1 to disable duty cycle budget per channel.

duty_cycle_budget_s

```
int16_t sl_wisun_regulation_params_t::duty_cycle_budget_s
```

Duty cycle budget in seconds per hour, -1 to disable duty cycle budget.

duty_cycle_skip_ack

```
uint8_t sl_wisun_regulation_params_t::duty_cycle_skip_ack
```

Skip ACKs from sum of transmissions.

duty_cycle_warning_threshold_percent

```
int8_t sl_wisun_regulation_params_t::duty_cycle_warning_threshold_percent
```

Duty cycle warning threshold in percent, -1 to disable duty cycle warning threshold.

enable_apc

```
uint8_t sl_wisun_regulation_params_t::enable_apc
```

Enable Adaptive Power Control.

enable_arib_channel_access

```
uint8_t sl_wisun_regulation_params_t::enable_arib_channel_access
```

Ensure a min IFS before ACKs (ARIB conditions)

fixed_cca_threshold_dbm

```
int8_t sl_wisun_regulation_params_t::fixed_cca_threshold_dbm
```

Fixed CCA threshold in dBm, INT8_MAX to use adaptive CCA threshold.

Copyright © 2025 Silicon Laboratories. All rights reserved.

max_edfe_duration_ms

> uint32_t sl_wisun_regulation_params_t::max_edfe_duration_ms

Maximum Extended Directed Frame Exchange (EDFE) duration in milliseconds, 0 to forbid EDFE.

reserved

> uint8_t sl_wisun_regulation_params_t::reserved

Reserved, set to zero.

version

> uint32_t sl_wisun_regulation_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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Predefined regional regulation parameter sets

Predefined regional regulation parameter sets for sl_wisun_set_regulation_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_regulation_params_t	SL_WISUN_REGULATION_PARAMS_ARIB Parameter set for ARIB regional regulation.
const sl_wisun_regulation_params_t	SL_WISUN_REGULATION_PARAMS_ETSI Parameter set for ETSI regional regulation.
const sl_wisun_regulation_params_t	SL_WISUN_REGULATION_PARAMS_NONE Default parameter set for no regional regulation.
const sl_wisun_regulation_params_t	SL_WISUN_REGULATION_PARAMS_WPC Parameter set for WPC regional regulation.

Variable Documentation

SL_WISUN_REGULATION_PARAMS_ARIB

```
const sl_wisun_regulation_params_t SL_WISUN_REGULATION_PARAMS_ARIB
```

Parameter set for ARIB regional regulation.

This parameter set is used for Japan and other ARIB compliant regions.

SL_WISUN_REGULATION_PARAMS_ETSI

```
const sl_wisun_regulation_params_t SL_WISUN_REGULATION_PARAMS_ETSI
```

Parameter set for ETSI regional regulation.

This parameter set is used for Europe and other ETSI compliant regions.

SL_WISUN_REGULATION_PARAMS_NONE

```
const sl_wisun_regulation_params_t SL_WISUN_REGULATION_PARAMS_NONE
```

Default parameter set for no regional regulation.

This parameter set is used when no regional regulation is applied.

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_WISUN_REGULATION_PARAMS_WPC

```
const sl_wisun_regulation_params_t SL_WISUN_REGULATION_PARAMS_WPC
```

Parameter set for WPC regional regulation.

This parameter set is used for India and other WPC compliant regions.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Border Router API

Border Router API

Wi-SUN Border Router API contains functions for Border Router specific functionality.

This API is an extension, not a replacement of Wi-SUN Stack API. An application implementing Border Router capability may utilize functions from both APIs. See Wi-SUN API availability for further information.

Functions

sl_status_t sl_wisun_br_get_gak(uint8_t index, uint8_t *gak)
Get Group AES Key.

sl_status_t sl_wisun_br_get_ip_addresses(uint8_t *addr_ll, uint8_t *addr_gua, uint8_t *addr_dodagid)
Get the unicast IPv6 addresses configured in the Border Router.

sl_status_t sl_wisun_br_get_routing_table(uint16_t *entry_count, sl_wisun_br_routing_table_entry_t *table)
Get the Routing Table.

sl_status_t sl_wisun_br_get_routing_table_entry_count(uint16_t *entry_count)
Get the Routing Table Size.

sl_status_t sl_wisun_br_get_state(sl_wisun_br_state_t *state)
Get the Border Router state.

sl_status_t sl_wisun_br_ipv6_down(const uint8_t *data, size_t data_length)
Forward an IPv6 packet to Wi-SUN network.

sl_status_t sl_wisun_br_ipv6_route_exists(const uint8_t *ipv6_address, bool *exists)
Check if the IPv6 destination address is routable in the Wi-SUN network.

sl_status_t sl_wisun_br_pan_defect_advertise(bool defect_advertise, uint32_t min_scan_duration_s, uint32_t max_scan_duration_s)
Start or stop PAN Defect advertisement.

sl_status_t sl_wisun_br_set_broadcast_settings(uint32_t interval_ms, uint8_t dwell_interval_ms)
Set broadcast settings.

sl_status_t sl_wisun_br_set_connection_parameters(const sl_wisun_br_connection_params_t *params)
Configure the Border Router parameter set.

sl_status_t sl_wisun_br_set_device_certificate(uint16_t certificate_options, uint16_t certificate_length, const uint8_t *certificate)
Set the Border Router device certificate.

sl_status_t sl_wisun_br_set_gtk(const uint8_t *gtk, uint8_t index)
Set Group Transient Key.

sl_status_t sl_wisun_br_set_ipv6_prefix(const uint8_t *ipv6_prefix, uint8_t prefix_length)
Set IPv6 prefix for DODAG.

sl_status_t sl_wisun_br_set_ipv6_up_handler(sl_wisun_ipv6_up_handler_t handler)
Register a function to handle IPv6 packets received from Wi-SUN network.

sl_status_t sl_wisun_br_set_lfn_parameters(const sl_wisun_br_lfn_params_t *params)
Configure LFN parenting parameter set.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_status_t sl_wisun_br_set_lfn_support(uint8_t lfn_limit, bool lfn_support_pan)
Configure LFN parenting support.

sl_status_t sl_wisun_br_set_network_size(sl_wisun_network_size_t size)
Set the size of the Wi-SUN network.

sl_status_t sl_wisun_br_start(const uint8_t *name, const sl_wisun_phy_config_t *phy_config)
Start the Border Router.

sl_status_t sl_wisun_br_stop(void)
Stop the Border Router.

sl_status_t sl_wisun_br_trigger_global_repair(void)
Trigger a global repair process from the Border Router.

Macros

#define sl_wisun_set_br_device_certificate sl_wisun_br_set_device_certificate
Define sl_wisun_set_br_device_certificate for legacy.

Function Documentation

sl_wisun_br_get_gak

```
sl_status_t sl_wisun_br_get_gak (uint8_t index, uint8_t * gak)
```

Get Group AES Key.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Index of the key from 0 to 3 for GAKs and 4 to 6 for LFN GAKs
uint8_t *	[out]	gak	Pointer to Group AES Key

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function exports (L)GAK as a plaintext key.

Note

- This functionality is only available when using the debug library.

sl_wisun_br_get_ip_addresses

```
sl_status_t sl_wisun_br_get_ip_addresses (uint8_t * addr_ll, uint8_t * addr_gua, uint8_t * addr_dodagid)
```

Get the unicast IPv6 addresses configured in the Border Router.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	addr_ll	Pointer to Link-Local address
uint8_t *	[out]	addr_gua	Pointer to Unique Local Address or Global Unicast Address
uint8_t *	[out]	addr_dodagid	Pointer to DODAG ID

Copyright © 2025 Silicon Laboratories. All rights reserved.

Returns

- SL_STATUS_OK if successful, an error code otherwise

sl_wisun_br_get_routing_table

```
sl_status_t sl_wisun_br_get_routing_table (uint16_t * entry_count, sl_wisun_br_routing_table_entry_t * table)
```

Get the Routing Table.

Parameters

Type	Direction	Argument Name	Description
uint16_t *	[inout]	entry_count	Maximum number of entries to read on input, number of entries read on output
sl_wisun_br_routing_table_entry_t *	[out]	table	Pointer to the application allocated memory where the routing table is written

Returns

- SL_STATUS_OK if successful, an error code otherwise

The routing table is unordered and only contains the parent-child relationship between the nodes. The table is in the format: | Node GUA/ULA 16 bytes | parent GUA/ULA 16 bytes |

Note

- This function should be called after sl_wisun_br_get_routing_table_entry_count() to get the number of entries in the routing table.

sl_wisun_br_get_routing_table_entry_count

```
sl_status_t sl_wisun_br_get_routing_table_entry_count (uint16_t * entry_count)
```

Get the Routing Table Size.

Parameters

Type	Direction	Argument Name	Description
uint16_t *	[out]	entry_count	Number of entries in the routing table

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function retrieves the number of entries in the routing table. This allows for allocation before calling sl_wisun_br_get_routing_table(). The allocated memory should be at least equal to: sizeof(sl_wisun_br_routing_table_entry_t) * entry_count.

sl_wisun_br_get_state

```
sl_status_t sl_wisun_br_get_state (sl_wisun_br_state_t * state)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Get the Border Router state.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_br_state_t *	[out]	state	Pointer to state • SL_WISUN_BR_STATE_INITIALIZED: ready to start • SL_WISUN_BR_STATE_OPERATIONAL: started

Returns

- SL_STATUS_OK if successful, an error code otherwise

sl_wisun_br_ipv6_down

```
sl_status_t sl_wisun_br_ipv6_down (const uint8_t * data, size_t data_length)
```

Forward an IPv6 packet to Wi-SUN network.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	data	Pointer to IPv6 header
size_t	[out]	data_length	Length of IPv6 packet data in bytes

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function forwards an IPv6 packet to Wi-SUN network. The destination address must belong to the DODAG prefix. This function is used for handling downlink packets from the backhaul interface, see sl_wisun_br_set_ipv6_up_handler() for uplink. It's recommended to use sl_wisun_br_ipv6_route_exists() before forwarding a packet.

sl_wisun_br_ipv6_route_exists

```
sl_status_t sl_wisun_br_ipv6_route_exists (const uint8_t * ipv6_address, bool * exists)
```

Check if the IPv6 destination address is routable in the Wi-SUN network.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	ipv6_address	Pointer to IPv6 destination address
bool *	[out]	exists	Whether the address is routable • **true**: address is routable • **false**: address is not routable

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function checks if the given IPv6 address is a valid destination in the Wi-SUN network. The destination address must belong to the Border Router or one of the connected nodes.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sl_wisun_br_pan_defect_advertise

> sl_status_t sl_wisun_br_pan_defect_advertise (bool defect_advertise, uint32_t min_scan_duration_s, uint32_t max_scan_duration_s)

Start or stop PAN Defect advertisement.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	defect_advertise	True to start PAN Defect procedure, false to stop it.
uint32_t	[in]	min_scan_duration_s	Minimum duration after which devices are allowed to switch their PAN if an alternative one is available.
uint32_t	[in]	max_scan_duration_s	Duration after which devices should systemically switch to the alternative PAN if one is available.

Returns

- SL_STATUS_OK if successful, an error code otherwise.

Note

- The values for `min_scan_duration_s` and `max_scan_duration_s` should be chosen based on the estimated performance of the backup power source. Specifically, `max_scan_duration_s` should be set to a value smaller than the estimated lifetime of the backup power source to ensure a seamless transition to the alternative PAN without long connection interruptions.

sl_wisun_br_set_broadcast_settings

> sl_status_t sl_wisun_br_set_broadcast_settings (uint32_t interval_ms, uint8_t dwell_interval_ms)

Set broadcast settings.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	interval_ms	Broadcast Interval (BDI < BI <= 16777215 ms) (default: 1020 ms)
uint8_t	[in]	dwell_interval_ms	Broadcast Dwell Interval (100-255 ms) (default: 255 ms)

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the broadcast channel hopping schedule. The Broadcast Interval is the duration, between Broadcast Dwell Intervals within the advertising node's broadcast schedule. Its resolution is limited to 24 bits (16777215 ms). The Broadcast Dwell Interval specifies the duration during which the node may transmit a broadcast message. It is recommended a BI value of 4.0 times the maximum of (UDI, BDI). These values affect the whole PAN, since they are propagated from parent to children. This function must be called before starting the Border Router.

sl_wisun_br_set_connection_parameters

> sl_status_t sl_wisun_br_set_connection_parameters (const sl_wisun_br_connection_params_t * params)

Configure the Border Router parameter set.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Parameters

Type	Direction	Argument Name	Description
const sl_wisun_br_connection_params_t *	[in]	params	Parameter set to use

Returns

- SL_STATUS_OK if successful, an error code otherwise

This functions sets the Border Router parameter set. These parameters impact connection time, bandwidth usage, and latency. Use of a predefined parameter set is recommended (Predefined BR parameter sets). Small profile will be used by default for all missing configurations. This function must be called before starting the Border Router.

sl_wisun_br_set_device_certificate

> sl_status_t sl_wisun_br_set_device_certificate (uint16_t certificate_options, uint16_t certificate_length, const uint8_t * certificate)

Set the Border Router device certificate.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	certificate_options	Options for the certificate. **SL_WISUN_CERTIFICATE_OPTION_APPEND**: Append the certificate to the list of server 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
uint16_t	[in]	certificate_length	Size of the certificate data
const uint8_t *	[in]	certificate	Pointer to the certificate data

Returns

- SL_STATUS_OK if successful, an error code otherwise

sl_wisun_br_set_gtk

> sl_status_t sl_wisun_br_set_gtk (const uint8_t * gtk, uint8_t index)

Set Group Transient Key.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	gtk	Group Transient Key
uint8_t	[in]	index	Index of the key from 0 to 3 for GTKs and 4 to 6 for LGTKs

Note

- This API is provided for test and debug purposes: setting a specific key in production is not recommended. The keys set using this API are saved to NVM, and subsequent calls to this API are ignored until the NVM is cleared using sl_wisun_clear_credential_cache.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Returns

- SL_STATUS_OK if successful, SL_STATUS_ALREADY_EXISTS when ignored, an error code otherwise

sl_wisun_br_set_ipv6_prefix

```
sl_status_t sl_wisun_br_set_ipv6_prefix (const uint8_t * ipv6_prefix, uint8_t prefix_length)
```

Set IPv6 prefix for DODAG.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	ipv6_prefix	IPv6 prefix
uint8_t	[in]	prefix_length	Prefix length. Currently, only 64 bit prefixes are supported.

Note

- Must not be called after the border router is started

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the prefix used to generate IP addresses for RPL traffic (DODAGID will derive from it). This function must be called before starting the Border Router.

sl_wisun_br_set_ipv6_up_handler

```
sl_status_t sl_wisun_br_set_ipv6_up_handler (sl_wisun_ipv6_up_handler_t handler)
```

Register a function to handle IPv6 packets received from Wi-SUN network.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_ipv6_up_handler_t	[in]	handler	Pointer to handler function

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function registers a handler function that will be called when the destination address of a received IPv6 packet is outside the DODAG prefix. Setting the handler function to NULL disables the forwarding. This function is used for handling uplink packets towards the backhaul interface, see sl_wisun_br_ipv6_down() for downlink.

sl_wisun_br_set_lfn_parameters

```
sl_status_t sl_wisun_br_set_lfn_parameters (const sl_wisun_br_lfn_params_t * params)
```

Configure LFN parenting parameter set.

Parameters

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
const sl_wisun_br_lfn_params_t *	[in]	params	Parameter set to use

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function sets the LFN parenting parameter set. These parameters impact battery life of all LFNs in the PAN. Use of a predefined parameter set is recommended (Predefined BR LFN-parenting parameter sets). This function must be called before starting the Border Router.

sl_wisun_br_set_lfn_support

```
sl_status_t sl_wisun_br_set_lfn_support (uint8_t lfn_limit, bool lfn_support_pan)
```

Configure LFN parenting support.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	lfn_limit	Maximum number of LFN children • **0**: LFN parenting is disabled in the Border Router • **> 0**: Maximum number of LFN children the Border Router can parent
bool	[in]	lfn_support_pan	LFN parenting support in the PAN • **true**: LFN parenting is enabled in the PAN • **false**: LFN parenting is disabled in the PAN

Returns

- SL_STATUS_OK if successful, an error code otherwise

This function configures LFN parenting support in the Border Router and in the PAN. Set sl_wisun_config_neighbor_table accordingly.

sl_wisun_br_set_network_size

```
sl_status_t sl_wisun_br_set_network_size (sl_wisun_network_size_t size)
```

Set the size of the Wi-SUN network.

Parameters

Type	Direction	Argument Name	Description
sl_wisun_network_size_t	[in]	size	Size of the network **SL_WISUN_NETWORK_SIZE_AUTOMATIC**: network size is managed automatically **SL_WISUN_NETWORK_SIZE_SMALL**: less than 100 nodes **SL_WISUN_NETWORK_SIZE_MEDIUM**: 100 to 800 nodes **SL_WISUN_NETWORK_SIZE_LARGE**: 800 to 1500 nodes

Returns

Copyright © 2025 Silicon Laboratories. All rights reserved.

SL_STATUS_OK if successful, an error code otherwise

This function sets the size of the network. The size is used to set various stack parameters, such as timing parameters to optimize device behavior in regards to node count. The device will function with any setting but may exhibit non-optimal behavior. Setting the size too large may cause slow connection speed and increased latency. Conversely, a value too small may cause increased network traffic.

sl_wisun_br_start

```
sl_status_t sl_wisun_br_start (const uint8_t * name, const sl_wisun_phy_config_t * phy_config)
```

Start the Border Router.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[in]	name	Name of the Wi-SUN network as a zero-terminated string
const sl_wisun_phy_config_t *	[in]	phy_config	Pointer to PHY configuration

Returns

- SL_STATUS_OK if successful, an error code otherwise

sl_wisun_br_stop

```
sl_status_t sl_wisun_br_stop (void )
```

Stop the Border Router.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- SL_STATUS_OK if successful, an error code otherwise

sl_wisun_br_trigger_global_repair

```
sl_status_t sl_wisun_br_trigger_global_repair (void )
```

Trigger a global repair process from the Border Router.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- SL_STATUS_OK if successful SL_STATUS_NOT_READY if the Border Router is not started SL_STATUS_FAIL if the DODAG Version is not incremented

Copyright © 2025 Silicon Laboratories. All rights reserved.

Increments the RPL DODAG Version to allow nodes in the network to choose a new position whose rank is not constrained by their rank within the old DODAG Version. The new DODAG Version is propagated through DIOs in the entire network and will cause all nodes to re-run a parent selection within the new DODAG Version, effectively temporarily disconnecting them from the network.

Note

- This operation will disrupt network communications as the repair process is carried out. It is not recommended to trigger a global repair more often than once every 24 hours.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Socket API

Modules

- cmsghdr
- in6_addr
- in6_pktinfo
- iovec
- ip6_mtuinfo
- ipv6_mreq
- msghdr
- sockaddr
- sockaddr_in6

Ancillary data object information MACROS

#define CMSG_ALIGN (len)
Returns the length of the data with the required alignment.

#define CMSG_SPACE (len)
Returns the number of bytes an ancillary element with payload of the passed data length occupies.

#define CMSG_LEN (len)
Returns the value to store in the cmsg_len member of the cmsghdr structure.

#define CMSG_FIRSTHDR (mhdr)
Returns a pointer to the first control message header in the given msghdr structure.

#define CMSG_NXTHDR (mhdr, cmsg)
Returns a pointer to the next control message header in the ancillary data buffer.

#define CMSG_DATA (cmsg)
Returns a pointer to the data portion of a cmsghdr.

Flags we can use with recv(), recvfrom() and recvmsg()

#define MSG_TRUNC 0×0001
Normal data truncated, to be used as msg_flags field in the msghdr structure.

#define MSG_CTRUNC 0×0002
Control data truncated, to be used as msg_flags field in the msghdr structure.

#define MSG_PEEK 0×0004
Leave received data in queue, to be used as the flags parameter.

Protocol levels used for socket_setsockopt.

Copyright © 2025 Silicon Laboratories. All rights reserved.

#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 uint32_t Set only No Yes SO_NONBLOCK uint32_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 SO_WRITABLE int32_t Get No No SO_READABLE int32_t Get No 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.

#define SO_WRITABLE 5
Non-standard POSIX option SO_WRITABLE.

#define SO_READABLE 6
Non-standard POSIX option SO_READABLE.

Socket DSCP (Differentiated Services Code Point) QoS supported levels.

#define DSCP_DEFAULT 0
Standard priority and it is socket default.

#define DSCP_AF11 10
Application high priority service: Stack priorities these messages over the default priority messages.

#define DSCP_EF 46
Expedited Forwarding (EF) QoS level enable high priority state: low loss, low delay, and low jitter services.

IPv6 socket options

IPv6 socket options summaryopt_name Data type set/getsockopt sendmsg recvmsg IPV6_TCLASS int Set/Get Yes No IPV6_UNICAST_HOPS int Set/Get No No IPV6_MULTICAST_HOPS int Set/Get No No IPV6_USE_MIN_MTU int Set/Get Yes No IPV6_HOPLIMIT int None Yes Yes IPV6_RECVHOPLIMIT int Set/Get No No IPV6_MULTICAST_LOOP int Set/Get Yes No IPV6_JOIN_GROUP ipv6_mreq_t Set only No No IPV6_ADD_MEMBERSHIP ipv6_mreq_t Set only No No IPV6_LEAVE_GROUP ipv6_mreq_t Set only No No IPV6_DROP_MEMBERSHIP ipv6_mreq_t Set only No No IPV6_PKTINFO in6_pktinfo_t None Yes

Copyright © 2025 Silicon Laboratories. All rights reserved.

Yes IPV6_RCVPKTINFO int Set/Get No No IPV6_RECVTCLASS int Set/Get No No IPV6_DONTFRAG int Set/Get Yes No SO_EDFE_MODE int32_t Set only Yes No

#define IPV6_TCLASS 1
IPv6 header has a field traffic class that contains a 6-bit Differentiated Services Code Point (DSCP) field that is used for packet classification.

#define IPV6_UNICAST_HOPS 2
Set the unicast hop limit for the socket, as int.

#define IPV6_MULTICAST_HOPS 3
Set the multicast hop limit for the socket, as int.

#define IPV6_USE_MIN_MTU 5
Specify PMTU preference, as int; Valid values -1 (PMTUD for unicast, default), 0 (PMTUD always), 1 (PMTUD off).

#define IPV6_HOPLIMIT 8
Ancillary data option on sendmsg()/recvmsg(), value to be used for single ongoing packet, as int; Valid values 0-255, -1 for default.

#define IPV6_RECVHOPLIMIT 11
Ancillary data option on recvmsg(), specifies hop-limit for the single received ongoing packet, as int; A nonzero value enables the option; a value of 0 disables the option.

#define IPV6_MULTICAST_LOOP 14
Specify whether outgoing multicast packets are looped back, as int.

#define IPV6_ADD_MEMBERSHIP 15
Join a multicast group.

#define IPV6_JOIN_GROUP IPV6_ADD_MEMBERSHIP
IPv6 header has a field traffic class that contains a 6-bit Differentiated Services Code Point (DSCP) field that is used for packet classification.

#define IPV6_DROP_MEMBERSHIP 16
Leave a multicast group.

#define IPV6_LEAVE_GROUP IPV6_DROP_MEMBERSHIP
IPv6 header has a field traffic class that contains a 6-bit Differentiated Services Code Point (DSCP) field that is used for packet classification.

#define IPV6_PKTINFO 9
Specify control messages packet info for recvmsg() as struct in6_pktinfo.

#define IPV6_RECVPKTINFO 10
Set delivery of the IPV6_PKTINFO control message on incoming datagrams, as int.

#define IPV6_RECVTCLASS 12
Enables the application to receive the value of the traffic class field from the IPv6 header, as int; A nonzero value enables the option; a value of 0 disables the option.

#define IPV6_DONTFRAG 6
Specify not to fragment datagrams, as int; For Wi-SUN, direct neighbors can receive frames up to 1504 bytes, while the other nodes can receive frames up to 1280 bytes.

#define SO_EDFE_MODE 0xfb
Experimental: Enable Extended Directed Frame Exchange mode.

#define SOCKET_EDFE_MODE SO_EDFE_MODE
IPv6 header has a field traffic class that contains a 6-bit Differentiated Services Code Point (DSCP) field that is used for packet classification.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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_protocol {

IPPROTO_IP = 0
IPPROTO_ICMP = 1
IPPROTO_TCP = 2
IPPROTO_UDP = 3

}

IP protocols.

enum socket_type {

SOCK_STREAM = 1
SOCK_DGRAM = 2
SOCK_RAW = 3

}

Socket types.

Typedefs

typedef struct cmsghdr — cmsghdr_t
Control messages.

typedef struct in6_addr — in6_addr_t
IPv6 Internet address.

typedef struct in6_pktinfo — in6_pktinfo_t
Structure to store the IPv6 packet information.

typedef struct iovec — iovec_t
Structure to store scatter/gather array elements.

typedef struct ip6_mtuinfo — ip6_mtuinfo_t
Structure to store the IPv6 MTU (Maximum Transmission Unit) information.

typedef struct ipv6_mreq — ipv6_mreq_t
Structure to specify the multicast group and the interface.

typedef struct msghdr — msghdr_t
The msghdr structure is used by the recvmsg() and sendmsg() functions.

typedef uint32_t — sa_family_t
Address family.

typedef enum socket_domain — sl_socket_domain_t
Supported address families.

Copyright © 2025 Silicon Laboratories. All rights reserved.

typedef enum socket_protocol — sl_socket_protocol_t
IP protocols.

typedef enum socket_type — sl_socket_type_t
Socket types.

typedef int — sl_wisun_socket_id_t
Socket id.

typedef struct sockaddr_in6 — sockaddr_in6_t
IPv6 address format.

typedef uint32_t — socklen_t
Socket address length type definition.

Variables

const in6_addr_t — in6addr_any
IPv6 wildcard address.

Functions

int — accept(int sockid, struct sockaddr *addr, socklen_t *addrlen)
Accept a connection on a socket.

int — bind(int sockid, const struct sockaddr *addr, socklen_t addrlen)
Bind a name to a socket.

int — close(int sockid)
Close a socket.

int — connect(int sockid, const struct sockaddr *addr, socklen_t addrlen)
Initiate a connection on a socket.

int — getpeername(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len)
Get the address of the peer connected to the socket.

int — getsockname(int socket, struct sockaddr *restrict address, socklen_t *restrict address_len)
Retrieves the local address of a socket.

int — getsockopt(int sockid, int level, int optname, void *optval, socklen_t *optlen)
Get socket option.

int — listen(int sockid, int backlog)
Listen for connections on a socket.

ssize_t — recv(int sockid, void *buf, size_t len, int flags)
Receive a message from a socket.

ssize_t — recvfrom(int sockid, void *buf, size_t len, int flags, struct sockaddr *src_addr, socklen_t *addrlen)
Receive messages from a socket.

ssize_t — recvmsg(int socket, struct msghdr *message, int flags)
Receive a message from a socket.

ssize_t — send(int sockid, const void *buff, size_t len, int flags)
Send a message on a socket.

ssize_t — sendmsg(int socket, const struct msghdr *message, int flags)
Send a message through a connection-mode or connectionless-mode socket.

Copyright © 2025 Silicon Laboratories. All rights reserved.

ssize_t sendto(int sockid, const void *buff, size_t len, int flags, const struct sockaddr *dest_addr, socklen_t addr_len)
Send a message to a given address.

int setsockopt(int sockid, int level, int optname, const void *optval, socklen_t optlen)
Set socket option designated by optname at a given protocol level to the value pointed by optval.

int socket(int domain, int type, int protocol)
Create an endpoint for communication and returns an Id that refers to that endpoint.

Macros

#define APP_LEVEL_SOCKET SOL_APPLICATION
For backward compatibility.

#define IPV6_ADDR_SIZE 16
Size of an IPv6 address.

#define IPV6_PREFIX_SIZE 8
Size of 64 bit IPv6 prefix byte array.

#define SOCK_NONBLOCK 0X00010000
When bitwise ored with socket's type, it sets the O_NONBLOCK status flag on the opened socket file description.

Ancillary data object information MACROS Documentation

Flags we can use with recv(), recvfrom() and recvmsg() Documentation

Protocol levels used for socket_setsockopt. Documentation

application level socket options Documentation

socket level options Documentation

Socket DSCP (Differentiated Services Code Point) QoS supported levels. Documentation

IPv6 socket options Documentation

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.

socket_domain

Copyright © 2025 Silicon Laboratories. All rights reserved.

socket_domain

Supported address families.

Enumerator	
AF_INET6	IP version 6.

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.

socket_type

socket_type

Socket types.

Enumerator	
SOCK_STREAM	stream (connection) socket (TCP)
SOCK_DGRAM	datagram (connectionless) socket (UDP)
SOCK_RAW	raw socket

Typedef Documentation

cmsghdr_t

typedef struct cmsghdr cmsghdr_t

Control messages.

in6_addr_t

typedef struct in6_addr in6_addr_t

IPv6 Internet address.

in6_pktinfo_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef struct in6_pktinfo in6_pktinfo_t
```

Structure to store the IPv6 packet information.

iovec_t

```
typedef struct iovec iovec_t
```

Structure to store scatter/gather array elements.

ip6_mtuinfo_t

```
typedef struct ip6_mtuinfo ip6_mtuinfo_t
```

Structure to store the IPv6 MTU (Maximum Transmission Unit) information.

ipv6_mreq_t

```
typedef struct ipv6_mreq ipv6_mreq_t
```

Structure to specify the multicast group and the interface.

msghdr_t

```
typedef struct msghdr msghdr_t
```

The msghdr structure is used by the recvmsg() and sendmsg() functions.

sa_family_t

```
typedef uint32_t sa_family_t
```

Address family.

sl_socket_domain_t

```
typedef enum socket_domain sl_socket_domain_t
```

Supported address families.

sl_socket_protocol_t

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
typedef enum socket_protocol sl_socket_protocol_t
```

IP protocols.

sl_socket_type_t

```
typedef enum socket_type sl_socket_type_t
```

Socket types.

sl_wisun_socket_id_t

```
typedef int sl_wisun_socket_id_t
```

Socket id.

sockaddr_in6_t

```
typedef struct sockaddr_in6 sockaddr_in6_t
```

IPv6 address format.

socklen_t

```
typedef uint32_t socklen_t
```

Socket address length type definition.

Variable Documentation

in6addr_any

```
const in6_addr_t in6addr_any
```

IPv6 wildcard address.

Function Documentation

accept

```
int accept (int sockid, struct sockaddr * addr, socklen_t * addrlen)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Accept a connection on a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket descriptor
struct sockaddr *	[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.
socklen_t *	[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.

bind

```
int bind (int sockid, const struct sockaddr * addr, socklen_t addrlen)
```

Bind a name to a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket id
const struct sockaddr *	[in]	addr	address structure ptr
socklen_t	[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.

close

```
int close (int sockid)
```

Close a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket id

Returns

- 0 on success, -1 on failure.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Close a socket and remove from the socket handler storage.

connect

```
int connect (int sockid, const struct sockaddr * addr, socklen_t addrlen)
```

Initiate a connection on a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket descriptor.
const struct sockaddr *	[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.
socklen_t	[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.

getpeername

```
int getpeername (int socket, struct sockaddr *restrict address, socklen_t *restrict address_len)
```

Get the address of the peer connected to the socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	socket	The socket file descriptor.
struct sockaddr *restrict	[out]	address	A pointer to a sockaddr structure where the peer's address will be stored.
socklen_t *restrict	[inout]	address_len	A pointer to a socklen_t variable that specifies the size of the 'address' structure.

This function retrieves the address of the peer connected to the specified socket. The peer's address is stored in the provided 'address' structure, and the length of the address is stored in 'address_len'.

Returns

- 0 on success, -1 on failure and sets `errno` appropriately.

getsockname

```
int getsockname (int socket, struct sockaddr *restrict address, socklen_t *restrict address_len)
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

Retrieves the local address of a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	socket	The socket descriptor.
struct sockaddr *restrict	[out]	address	A pointer to a `struct sockaddr` where the local address information will be stored.
socklen_t *restrict	[inout]	address_len	A pointer to a `socklen_t` variable that specifies the size of the `address` structure. Upon successful completion, it contains the actual size of the address data.

Returns

- On success, returns zero. On failure, returns -1 and sets `errno` appropriately.

This function retrieves the local address associated with a socket. It populates the provided `address` structure with the local address information.

getsockopt

> int getsockopt (int sockid, int level, int optname, void * optval, socklen_t * optlen)

Get socket option.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket descriptor.
int	[in]	level	socket protocol level.

int	[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

void *	[out]	optval	option value structure pointer.
socklen_t *	[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.

listen

> int listen (int sockid, int backlog)

Copyright © 2025 Silicon Laboratories. All rights reserved.

Listen for connections on a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket id
int	[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.

recv

> ssize_t recv (int sockid, void * buf, size_t len, int flags)

Receive a message from a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	descriptor of socket to receive the message from.
void *	[out]	buf	pointer to destination data buffer.
size_t	[in]	len	length of destination data buffer.
int	[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)

recvfrom

> ssize_t recvfrom (int sockid, void * buf, size_t len, int flags, struct sockaddr * src_addr, socklen_t * addrlen)

Receive messages from a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	descriptor of socket to receive the message from.
void *	[out]	buf	pointer to destination data buffer.
size_t	[in]	len	length of destination data buffer.
int	[in]	flags	flags to select type of message reception. Ignored in our implementation.
struct sockaddr *	[in]	src_addr	pointer to a sockaddr structure in which the sending address is to be stored.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
socklen_t *	[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.

recvmsg

```
ssize_t recvmsg (int socket, struct msghdr * message, int flags)
```

Receive a message from a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	socket	Specifies the socket file descriptor.
struct msghdr *	[inout]	message	Points to a msghdr structure, containing both the buffer to store the source address and the buffers for the incoming message.
int	[in]	flags	Specifies the type of message reception.

Returns

- The number of bytes received on success, -1 if an error occurred.

This function is typically used with connectionless-mode sockets because it permits the application to retrieve the source address of received data.

send

```
ssize_t send (int sockid, const void * buff, size_t len, int flags)
```

Send a message on a socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket descriptor.
const void *	[in]	buff	pointer to data buffer to send.
size_t	[in]	len	length of data buffer to send.
int	[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).

sendmsg

Copyright © 2025 Silicon Laboratories. All rights reserved.

> ssize_t sendmsg (int socket, const struct msghdr * message, int flags)

Send a message through a connection-mode or connectionless-mode socket.

Parameters

Type	Direction	Argument Name	Description
int	[in]	socket	Specifies the socket file descriptor.
const struct msghdr *	[in]	message	Points to a msghdr structure, containing both the destination address and the buffers for the outgoing message.
int	[in]	flags	Specifies the type of message transmission.

Returns

- The number of bytes received on success, -1 if an error occurred.

sendto

> ssize_t sendto (int sockid, const void * buff, size_t len, int flags, const struct sockaddr * dest_addr, socklen_t addr_len)

Send a message to a given address.

Parameters

Type	Direction	Argument Name	Description
int	[in]	sockid	socket descriptor.
const void *	[in]	buff	pointer to data buffer to send.
size_t	[in]	len	length of data buffer to send.
int	[in]	flags	flags to select send options. Ignored in our implementation.
const struct sockaddr *	[in]	dest_addr	pointer to destination address buffer; Required for datagram sockets.
socklen_t	[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).

setsockopt

> int setsockopt (int sockid, int level, int 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

Type	Direction	Argument Name	Description
int	[in]	sockid	socket ID

Copyright © 2025 Silicon Laboratories. All rights reserved.

Type	Direction	Argument Name	Description
int	[in]	level	protocol level at which the option resides. it could be: • SOL_SOCKET • IPPROTO_IPV6 • SOL_APPLICATION

int	[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

const void *	[in]	optval	Pointer to the socket option new value. The type of variable pointed by optval depends level and optname values.
socklen_t	[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.

socket

> int socket (int domain, int type, int protocol)

Create an endpoint for communication and returns an Id that refers to that endpoint.

Parameters

Type	Direction	Argument Name	Description
int	[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)
int	[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)
int	[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

Copyright © 2025 Silicon Laboratories. All rights reserved.

Returns

- The socket's id on success, (-1) on failure.

Copyright © 2025 Silicon Laboratories. All rights reserved.

cmsghdr

Control messages.

Public Attributes

socklen_t	cmsg_len Message length in bytes, including this header.
int	cmsg_level originating protocol
int	cmsg_type protocol-specific type

Public Attribute Documentation

cmsg_len

```
socklen_t cmsghdr::cmsg_len
```

Message length in bytes, including this header.

cmsg_level

```
int cmsghdr::cmsg_level
```

originating protocol

cmsg_type

```
int cmsghdr::cmsg_type
```

protocol-specific type

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

in6_pktinfo

Structure to store the IPv6 packet information.

Public Attributes

in6_addr_t	ipi6_addr src/dst IPv6 address
int	ipi6_ifindex arrival interface index

Public Attribute Documentation

ipi6_addr

```
in6_addr_t in6_pktinfo::ipi6_addr
```

src/dst IPv6 address

ipi6_ifindex

```
int in6_pktinfo::ipi6_ifindex
```

arrival interface index

Copyright © 2025 Silicon Laboratories. All rights reserved.

iovec

Structure to store scatter/gather array elements.

Public Attributes

void *	iov_base Store the base address of a memory region for input or output.
size_t	iov_len Length of the memory region iov_base points to.

Public Attribute Documentation

iov_base

```
void* iovec::iov_base
```

Store the base address of a memory region for input or output.

iov_len

```
size_t iovec::iov_len
```

Length of the memory region iov_base points to.

Copyright © 2025 Silicon Laboratories. All rights reserved.

ip6_mtuinfo

Structure to store the IPv6 MTU (Maximum Transmission Unit) information.

Public Attributes

struct sockaddr_in6	ip6m_addr IPv6 address.
uint32_t	ip6m_mtu MTU size.

Public Attribute Documentation

ip6m_addr

```
struct sockaddr_in6 ip6_mtuinfo::ip6m_addr
```

IPv6 address.

ip6m_mtu

```
uint32_t ip6_mtuinfo::ip6m_mtu
```

MTU size.

Copyright © 2025 Silicon Laboratories. All rights reserved.

ipv6_mreq

Structure to specify the multicast group and the interface.

Public Attributes

int	ipv6mr_ifindex Local IPv6 address of interface.
struct in6_addr	ipv6mr_multiaddr IPv6 multicast address of group.

Public Attribute Documentation

ipv6mr_ifindex

```
int ipv6_mreq::ipv6mr_ifindex
```

Local IPv6 address of interface.

ipv6mr_multiaddr

```
struct in6_addr ipv6_mreq::ipv6mr_multiaddr
```

IPv6 multicast address of group.

Copyright © 2025 Silicon Laboratories. All rights reserved.

msghdr

The msghdr structure is used by the recvmsg() and sendmsg() functions.

Public Attributes

void *	msg_control ancillary data
socklen_t	msg_controllen ancillary data buffer length
int	msg_flags flags on received message
struct iovec *	msg_iov scatter/gather array
int	msg_iovlen
void *	msg_name ptr to socket address structure
socklen_t	msg_namelen size of socket address structure

Public Attribute Documentation

msg_control

```
void* msghdr::msg_control
```

ancillary data

msg_controllen

```
socklen_t msghdr::msg_controllen
```

ancillary data buffer length

msg_flags

```
int msghdr::msg_flags
```

flags on received message

msg_iov

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
struct iovec* msghdr::msg_iov
```

scatter/gather array

msg_iovlen

```
int msghdr::msg_iovlen
```

elements in msg_iov

msg_name

```
void* msghdr::msg_name
```

ptr to socket address structure

msg_namelen

```
socklen_t msghdr::msg_namelen
```

size of socket address structure

Copyright © 2025 Silicon Laboratories. All rights reserved.

sockaddr

Socket address.

Public Attributes

uint8_t	sa_data 26 bytes of protocol address (IPv6)
sa_family_t	sa_family address family, AF_XXXX

Public Attribute Documentation

sa_data

```
uint8_t sockaddr::sa_data[26]
```

26 bytes of protocol address (IPv6)

sa_family

```
sa_family_t sockaddr::sa_family
```

address family, AF_XXXX

Copyright © 2025 Silicon Laboratories. All rights reserved.

sockaddr_in6

IPv6 address format.

Public Attributes

in6_addr_t	sin6_addr IPv6 address.
sa_family_t	sin6_family AF_INET6.
uint32_t	sin6_flowinfo IPv6 flow information.
in_port_t	sin6_port Transport layer port.
uint32_t	sin6_scope_id Scope ID.

Public Attribute Documentation

sin6_addr

```
in6_addr_t sockaddr_in6::sin6_addr
```

IPv6 address.

sin6_family

```
sa_family_t sockaddr_in6::sin6_family
```

AF_INET6.

sin6_flowinfo

```
uint32_t sockaddr_in6::sin6_flowinfo
```

IPv6 flow information.

sin6_port

```
in_port_t sockaddr_in6::sin6_port
```

Transport layer port.

Copyright © 2025 Silicon Laboratories. All rights reserved.

sin6_scope_id

```
uint32_t sockaddr_in6::sin6_scope_id
```

Scope ID.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Select API

Modules

fd_set

Typedefs

typedef long int **fd_mask**
A type representing a set of file descriptors.

Functions

int **select**(int nfds, fd_set *restrict readfds, fd_set *restrict writefds, fd_set *restrict exceptfds, struct timeval *restrict timeout)
Monitor multiple file descriptors for readiness.

void **sl_wisun_check_read_sockfd_set**(void)
Check the read socket file descriptor set.

void **sl_wisun_check_write_sockfd_set**(void)
Check the write socket file descriptor set.

void **sl_wisun_select_init**(void)
Initialize the select system.

Macros

#define **FD_CLR** (d, set)
Clear the bit for the file descriptor d in the fd_set.

#define **FD_ELT** (d)
Calculate the index in the fds_bits array for the file descriptor d.

#define **FD_ISSET** (d, set)
Check whether the bit for the file descriptor d is set in the fd_set.

#define **FD_MASK** (d)
Create a mask for the bit position within an fd_mask for the file descriptor d.

#define **FD_SET** (d, set)
Set the bit for the file descriptor d in the fd_set.

#define **FD_SETSIZE** 64
Maximum number of file descriptors in 'fd_set'.

#define **FD_ZERO** (set)
Clear all bits in the fd_set.

#define **NFDBITS** (8 * (int) sizeof (fd_mask))
The number of bits in an fd_mask.

Typedef Documentation

Copyright © 2025 Silicon Laboratories. All rights reserved.

fd_mask

```
typedef long int fd_mask
```

A type representing a set of file descriptors.

Function Documentation

select

```
int select (int nfds, fd_set *restrict readfds, fd_set *restrict writefds, fd_set *restrict exceptfds, struct timeval *restrict timeout)
```

Monitor multiple file descriptors for readiness.

Parameters

Type	Direction	Argument Name	Description
int	[in]	nfds	The highest-numbered file descriptor in any of the three sets, plus 1.
fd_set *restrict	[inout]	readfds	An optional pointer to a set of file descriptors to be checked for readability.
fd_set *restrict	[inout]	writefds	An optional pointer to a set of file descriptors to be checked for writability.
fd_set *restrict	[inout]	exceptfds	An optional pointer to a set of file descriptors to be checked for exceptions.
struct timeval *restrict	[in]	timeout	An optional timeout. If NULL, select() blocks indefinitely. To specify a non-blocking poll, set the timeout to zero.

Returns

- On success, returns the total number of bits set in readfds, writefds. If the time limit expires, select() returns zero. On error, -1 is returned, and errno is set appropriately.

This function monitors multiple file descriptors, waiting until one or more of the file descriptors are ready for an I/O operation.

Note

- This implementation does not handle exceptfds.

sl_wisun_check_read_sockfd_set

```
void sl_wisun_check_read_sockfd_set (void )
```

Check the read socket file descriptor set.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function notifies select to check the read socket file descriptor set. The function should be called in response to the reception of SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID. It notifies select to check the read socket file descriptor set for any ready socket to read.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Usage:

```
// Application indication handler
void sl_wisun_on_event(void) {
  // Application code here...
  switch (evt->header.id) {
    // When SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID is received
    case SL_WISUN_MSG_SOCKET_DATA_AVAILABLE_IND_ID:
      // Call the function to check the read socket file descriptor set
      sl_wisun_check_read_sockfd_set();
      break;
  }
  // Application code here...
}
```

sl_wisun_check_write_sockfd_set

```
void sl_wisun_check_write_sockfd_set (void )
```

Check the write socket file descriptor set.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function notifies select to check the write socket file descriptor set. The function should be called in response to the reception of SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID. It notifies select to check the write socket file descriptor set for any ready socket to write.

Usage:

```
void sl_wisun_on_event(void) {
  // Your code here...
  switch (evt->header.id) {
    // When SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID is received
    case SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID:
      // Call the function to check the write socket file descriptor set
      sl_wisun_check_write_sockfd_set();
      break;
  }
  // Your code here...
}
```

sl_wisun_select_init

```
void sl_wisun_select_init (void )
```

Initialize the select system.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function initializes the select system by creating a new event flag with the specified attributes. The event flag is used to indicate when a file descriptor set is ready to be checked for I/O operations.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Copyright © 2025 Silicon Laboratories. All rights reserved.

fd_set

A structure representing a set of file descriptors for select operations.

Public Attributes

fd_mask fds_bits
An array of fd_mask, each bit representing a file descriptor.

Public Attribute Documentation

fds_bits

```
fd_mask fd_set::fds_bits[FD_SETSIZE/(8 *(int) sizeof(fd_mask))]
```

An array of fd_mask, each bit representing a file descriptor.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN Stack Changelog

Wi-SUN Stack Changelog

Release 2.7

(release date 2025-07-16)

New Features and Improvements

- improved protection against key reinstallation
- introduced sl_wisun_set_regulation_parameters()
- introduced sl_wisun_reset_regulation_duty_cycle()
- added MAC parameters
- added sl_wisun_set_eap_identity()
- RCP requires FreeRTOS and CPC with Ocelot

RCP API

- 2.11.0
  - SET_RADIO_REGULATION always resets regulation data when called.
  - Add flag in REQ_DATA_TX to enable reporting of TX duration over the last hour in CNF_DATA_TX.
  - Add REQ_RADIO_TX_DURATION_RESET command.

Bug Fixes

- fixed authenticator dropping supplicant retries with high congestion
- fixed timeout management on asynchronous packets
- fixed async fragment could be split with DFE/EDFE frames
- fixed socket TX buffer full in case of routing or memory error
- fixed segmentation fault when unexpected EDFE final frame is received
- fixed a fault when an LFN fails to complete multiple handshakes in join state 5
- fixed node failing to connect after obtaining an inactive GTK first
- fixed not decrementing hop limit when forwarding multicast packets to LFNs
- fixed authenticator vulnerability to installation of predictable keys
- fixed CCA threshold value used for O-QPSK PHYs
- fixed memory corruption when returning to join state 1 while proxying RPL for LFNs
- fixed MAC tx confirmation overwriting previous one in case of consecutive failures

Release 2.6

(release date 2025-06-16)

New Features and Improvements

- removed data available traces on BR SoC CLI for the dhcpv6 socket
- added MAC statistics to track the time spent in idle and TX states
- introduced sl_wisun_connection_params_t v6
- added MPL statistics

Copyright © 2025 Silicon Laboratories. All rights reserved.

- introduced sl_wisun_lfn_params_t v3, added eapol_unicast_interval_ms
- added sl_wisun_set_preferred_pan() API
- wisun_fan_certification component uses stack's default FAN TPS Version (FAN 1.1)
- limit the number of children a router can accept
- deny temporarily a router unable to parent
- updated BZ and SG channel plans according to PHY TPS 2v03
- FFN and LFN now have storage for Frame Counters of up to 15 different PAN
- ETSI Adaptive Power Control available for EU PHYs
- add sl_wisun_config_neighbor_table_size API. Deprecate sl_wisun_set_neighbor_table_size API.
- add new security neighbor table
- ACK timeout now measured to start of response frame. It allows being back to RX channels right after Tack if no response is received.
- added debug MAC traces by default in RCP
- chan_params_table ownership is transfered to application
- FC-IE transmit flow control now corresponds to the estimated frame duration
- WH-IEs from duplicate frames are now processed.
- extended sl_wisun_statistics_heap_t to output highest used, currently used, free and total heap space.
- added concurrent detection support and sl_wisun_config_concurrent_detection() API
- exposed the node's estimated hop count to the BR in sl_wisun_network_info_t. Added the SL_WISUN_NETWORK_UPDATE_FLAGS_HOP_COUNT event
- added GlobalSign and Wisekey device credentials in Wi-SUN SoC CLI
- added Wisekey device credentials in Wi-SUN SoC Border Router CLI
- added multi fragments EDFE support in SOC
- removed automatic global repair that was done every 12 hours
- added sl_wisun_br_trigger_global_repair() to manually trigger a global repair
- Node Role IE includes LBATS Implemented field
- do not send unicast DIS to denied neighbors
- delay initial DIS so it doesn't overlap with PC sequence
- add DMP support
- introduced Wi-SUN NVM versioning
- removed OQPSK SFM configuration in sample applications
- add more meaningful status codes for SL_WISUN_MSG_SOCKET_DATA_SENT_IND_ID indication
- introduced sl_wisun_br_get_routing_table_entry_count() and sl_wisun_br_get_routing_table() for the SoC Border router
- added an event on BR routing table updates (the type of update is differentiated)
- improved LFN strategy after authentication failure
- Added Vendor Payload IE to advertise the node's concurrent detection capabilities
- default eapol_unicast_interval_ms adjusted to mitigate schedule overlaps during LFN authentication

RCP API

- 2.10.0
  - Support concurrent detection in transmission.
  - Add alternate PHYs to REQ_RADIO_LIST command.
- 2.9.0
  - add command to update LFN unicast schedule for TX
- 2.8.0
  - Use Source PAN ID specified in 15.4 header when relevant. Previously, the PAN ID set with SET_FILTER_PANID was always used, which lacked flexibility. An error would occur if the Source PAN ID in the 15.4 header differed from the one set.
- 2.7.0
  - fix tx-abort command
- 2.6.0
  - handle channel masks for mode-switch

Bug Fixes

- fixed mode switch feature from router / border router to LFN
- fixed PSA crypto key leak preventing authentication after multiple attempts over the device's lifetime

Copyright © 2025 Silicon Laboratories. All rights reserved.

- Border Router parameters can only be set while stopped
- drop security handshake frames from a previous connection
- fixed issue preventing LFN from sending LPAS correctly
- LFN unregister from multicast group when the corresponding socket is closed
- fixed RCP reset triggered when MAC would purge LFN frames upon LFN PAN Advertisement transmission
- seeded MPL messages are now sent the same number of times as their trickle's parameter TimerExpirations
- fixed buffer read overflow in sl_wisun_br_set_ipv6_prefix()
- fixed use-after-free in case of DHCPv6 failure
- fixed dhcp preferred lifetime when address lifetime is infinite
- fixed wrong error message in wisun_soc_cli when reconnecting during an ongoing disconnection
- fixed tasks' priority in SoC BR
- fixed unregistering when changing parent (send NS(ARO) with no lifetime)
- fixed RPL GLobal Repair, no DAO was transmitted if the same preferred parent was selected
- fixed a memory corruption causing authentication failures on Secure Vault Mid devices
- fixed Neighbor Cache Refresh
- fixed issue blocking an LFN in reception state for the duration of a broadcast interval
- fixed aborted frames generating no confirmation
- fixed Source Addressing Mode of EDFE Response Frames in wisun_rcp
- fixed LFN PAN Advertisements sometimes stuck until timeout
- fixed sl_wisun_br_dhcpv6_set_vendor_data API
- fixed EDFE retries not done with a CCA backoff
- fixed an FHSS synchronization issue due to incorrect UFSI values
- fixed radio configuration file error with SG phys
- advertise border router address/prefix lifetime as infinite instead of 0 in DIO packets
- fixed nodes roaming back and forth with PAN Defect and Preferred PAN enabled
- fixed missing timeout for LFN broadcast packet
- analyze inner IPv6 headers to refresh neighbor address registrations
- fixed node stuck in Join State 3 when it cannot decrypt a PC from its previous PAN
- fixed handling of SL_WISUN_MSG_REGULATION_TX_LEVEL_IND_ID on BR CLI
- prevent dao path sequence counter from resetting when going to join state 1 and 3
- fixed rescheduling issue with LFN frames that would make a node completely unable to operate
- fixed invalid duplicated frame detection from LFN neighbors as an FFN
- fixed invalid duplicated frame detection from FFN parent as an LFN
- fixed cleaning of LFN neighbors in MAC neighbor table
- fixed return option value of getsockopt() for SO_SNDLOWAT option
- Path Lifetime in DAO is rounded up
- fixed LFN issue when multiple frames advertise an updated LUS-IE
- handle invalid 4WH message 3 and GKH message 1
- fix send/sendto setting wrong errno when sendq is fulland socket is not in blocking mode
- fix send/sendto not returning the correct number of sent bytes in the case of a partial packet send
- fixed synchronization to broadcast schedule after LFN parent selection
- router now deletes earo route when receiving ns aro lifetime 0
- fixed wrong timeout for LFN
- fixed incorrect backoff slot duration with SUN OFDM
- fixed LFN failing in Join State 3 when it fails to decrypt an FFN-specific packet
- enable 15.4 acceleration in RAIL only if 15.4g is supported
- fixed PAN Defect IE relayed twice by FFNs
- fixed ETSI Adaptive Power Control with FAN TPS version 1.0
- fixed setting correct errno values for socket API
- fixed dependencies for set_dhcp_vendor_data command on BR SoC CLI
- fixed wrong default k trickle parameter
- fixed FFN unable to send DAO until PAN timeout
- fixed Preferred PAN and PAN Defect generating too much traffic
- fixed CCA delay over 1000µs due to inaccurate TX timing reported by RAIL
- fixed LFN unicast schedule resync on LBT-IE receptions from parent

Copyright © 2025 Silicon Laboratories. All rights reserved.

Release 2.3

(release date 2024-12-16)

New Features and Improvements

- added sl_wisun_set_leaf() API to make an FFN act as RPL leaf.
- added sl_wisun_set_tx_power_ddbm() API to set the Tx power with a 0.1 dB resolution
- added sl_wisun_br_set_ipv6_up_handler() API to register a callback function to handle IPv6 packets received from Wi-SUN network
- added sl_wisun_br_ipv6_down() API to forward IPv6 packets to Wi-SUN network from backhaul point-of-view
- added sl_wisun_trace_debug(), sl_wisun_trace_info(), sl_wisun_trace_warn() and sl_wisun_trace_error() to trace events in the stack journal.
- added sl_wisun_set_phy_sensitivity() API to set the RX sensitivity for a chosen PHY
- added lfn_na_wait_duration_m in FFN connection params to configure how often an FFN retransmits DAO on behalf of a registered LFN.
- introduced Direct Connect
- introduced PAN Defect
- added Wi-SUN border router libraries. They were previously kept private.
- added Wi-SUN border router public APIs headers. They were previously kept private.
- added support for Wi-Fi connectivity based on Silicon LABS SiWx917 and WiSeConnect v3.4.0
- added support for an external DHCPv6 Server
- added DAO No-Path transmission to de-register a router prior to leaving the network
- updated the RX sensitivities with updated values.
- updated Registration Refresh Request as mandated by FAN TPS 1.1v09.
- updated NA channel plans 4 and 5 as mandated by PHY TPS amendment 1VA12
- improved retransmission mechanism
- improved handling of parent changes/losses
- reduced weight given to the ETX first measure in the ETX EWMA
- reduced weight given to the RSL first measure in the RSL EWMA
- reworked MAC, FFN and LFN bootstrap state machines
- raised TCP maximum segment size (mss)
- dropped deprecated definitions related to socket options

RCP API

- 2.5.0
  - Allow using security key with index 8.
- 2.4.1
  - Fix the start of the LFN broadcast schedule.
  - FATAL when failing to start the LFN broadcast schedule.
  - FATAL when trying to send an LFN broadcast frame without having the LFN broadcast schedule running.
- 2.4.0
  - Add phy sensitivity to REQ_RADIO_LIST command.
- 2.3.0
  - Add timing info, parent address and parent frame counters to SET_FHSS_FFN_BC command.

Bug Fixes

- fixed incorrect EAP packet filtering when the EAP packet ID would rollover
- fixed PAN Cost calculation
- fixed memory leak when a node is disconnected and has neighbors with registered POM-IE info
- fixed IPV6_RECVTCLASS, IPV6_UNICAST_HOPS, IPV6_MULTICAST_HOPS and SO_EDFE_MODE socket options
- fixed SO_NONBLOCK socket option not being handled correctly

Copyright © 2025 Silicon Laboratories. All rights reserved.

- fixed POM-IE when only 1 other PhyModeId is available
- fixed FFN multicast packet transmission timeout when LFN multicast packet also pending for transmission
- fixed MPL Buffer Message set containing messages with a sequence inferior to min sequence
- fixed MPL messages not being deleted on expiration when trickle timer was inhibited
- fixed multicast entry management in LFN parenting
- fixed RPL candidate parent set, reject neighbors with low RSL
- fixed hop limit field in IPv6 tunneling
- changed parent unregistration to include all parents, preventing bad DAO route on reconnection
- fixed missing probe to a new neighbor in join state 5
- RPL parents from the previous connection are not loaded from NVM if discovery.allow_skip is false
- changed BSI / broadcast schedule update to be only when receiving a PC with a new PAN version
- fixed parent selection causing frequent parent switches or invalid infinite rank advertised
- fixed segmentation fault when candidate parent timeout and no preferred parent is selected
- fixed an issue causing the radio to stay in IDLE state until the next TX after an RX timeout

Release 2.0

(release date 2024-06-05)

New Features and Improvements

- dropped the support of our previous socket API.
- added a random delay before starting an LFN join sequence.
- added support for select(), sendmsg(), recvmsg(), getpeername() and getsockname().
- extended the list of supported socket options.
- split the stack into smaller software modules offering more granularity and helping reduce the memory footprint.

Bug Fixes

- fixed an issue causing the radio to be stuck in RX. The cause has been identified as a race in the lower layers of the radio driver.
- fixed an invalid memory access in the timer and event lists.
- fixed multiple invalid counter and timer updates during state transitions and RPL parent updates. They were causing abusive router disconnections.
- fixed an invalid use of a link-local as a source address in the DAO.
- fixed the condition upon which the MAC reset the CCA failures counter.
- fixed an invalid PAN ID filter configuration that was preventing routers from connecting to a new border router after a PAN timeout.

Release 1.10

(release date 2023-04-10)

New Features and Improvements

- added a new API sl_wisun_set_neighbor_table_size()that set the neighbor table size.

Bug Fixes

- fixed an invalid initialization causing the number of retries sent by Wi-SUN RCP to be inconsistent.
- fixed an error that, in some rare occasions, could cause the IPv6 source routing header corruption.
- fixed an error that was causing significant transmissions delays when a multicast packet reception occurred during the transmission of asynchronous packets.
- fixed LFN clock drift compensation. It sometimes skipped and caused the LFN to disconnect.

Copyright © 2025 Silicon Laboratories. All rights reserved.

- improved the resiliance of 4-way handshake to errors. After a missing packet the joining node and the authenticator ended with a different status.

Release 1.9

(release date 2023-02-14)

New Features and Improvements

- major update of the RCP. Must be used in pair with wsbrd v1.9 and more recent. It significantly reduces the memory footprint. The memory required to run the application no longer depends on the number of routers connected to the border router.
- dropped the high-reliability transmission mechanism. Its use was not recommended and it was causing a significant increase of all communications latency.
- added support for ETSI Adaptive Power Control.
- improved LFNs clock drift compensation. It reduces the power consumption and the connection stability.

Bug Fixes

- fixed an error causing UDP sockets to be closed when receiving packets with an empty payload.
- fixed an error causing the frequency hopping to be blocked on rare occasions.
- fixed an error causing LLC messaging allocation to fail. It was caused by an invalid logic in the mac tx queue.
- fixed an error causing LFNs to use an invalid broadcast schedule after a reconnection.
- fixed an error causing LFNs reconnection to a router to fail.
- fixed an error preventing LFNs to send NS(EARO) and thus to reconnect after a given number of non-consequentive unsuccessful connection attempts.
- fixed an error causing an invalid memory access in dense topologies.
- LUS-IE were sent in all LFN data frames, even when the Listen Interval did not change.

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)

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

- 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)

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

Copyright © 2025 Silicon Laboratories. All rights reserved.

(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)

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Wi-SUN Sockets

Wi-SUN Sockets

Wi-SUN sockets are based on the well-known BSD socket API, also known as the POSIX socket API. Each socket represents a handle for the local endpoint of a communication circuit.

> Earlier Silicon Labs socket API was deprecated in release 1.8 and removed in release 2.0.

API Overview

While Wi-SUN sockets follow the BSD socket API conventions, there are some limitations and omissions in the implementation as described below.

- Wi-SUN FAN supports only IPv6, so none of the IPv4-specific socket functionality is available
- Socket handles are specific to Wi-SUN sockets and cannot be used as file descriptors in C library function calls or other APIs
- POSIX signals are not supported, socket signaling uses Wi-SUN API events instead

Functions supported:

- socket()
- close()
- bind()
- send()
- sendto()
- sendmsg()
- recvfrom()
- recv()
- recvmsg()
- accept()
- connect()
- listen()
- setsockopt()
- getsockopt()
- getsockname()
- getpeername()
- select() (requires a separate application component)

Following functions are not supported:

- gethostbyname()
- gethostbyaddr()
- poll()

Migration to BSD Socket API

The following table details the mapping between the BSD socket API and the former Silicon Labs socket API.

Copyright © 2025 Silicon Laboratories. All rights reserved.

Former Silicon Labs Socket API Function	BSD Function
sl_wisun_open_socket()	socket()
sl_wisun_bind_socket()	bind()
sl_wisun_listen_on_socket()	listen()
sl_wisun_connect_socket()	connect()
sl_wisun_accept_on_socket()	accept()
sl_wisun_send_on_socket()	send()
sl_wisun_sendto_on_socket()	sendto()
sl_wisun_receive_on_socket()	recv()
sl_wisun_close_socket()	close()
sl_wisun_get_socket_option()	getsockopt()
sl_wisun_set_socket_option()	setsockopt()

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.

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, depending on the used protocol.

Receive

To manage received socket data, the socket can be either in indication mode or in polling mode. All sockets default to polling mode.

Copyright © 2025 Silicon Laboratories. All rights reserved.

In polling 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(). 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 are blocking, i.e. the calling RTOS task is suspended until the socket operation completes. Sockets can be configured as non-blocking either by or'ing SOCK_NONBLOCK flag to the type parameter of socket(), or using the socket option SO_NONBLOCK. Only a single blocking operation per socket is allowed.

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

1. 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 ...

Copyright © 2025 Silicon Laboratories. All rights reserved.

Deprecated List

Deprecated List

- Global sl_wisun_set_neighbor_table_size (uint8_t size) SL_DEPRECATED_API_SDK_2025_6
  This function will be removed in the future versions of the Wi-SUN stack, use sl_wisun_config_neighbor_table() instead.
- Global sl_wisun_set_tx_power (int8_t tx_power) SL_DEPRECATED_API_SDK_2024_6
  This function will be removed in the future versions of the Wi-SUN stack, use sl_wisun_set_tx_power_ddbm() instead.
- Global sl_wisun_util_connect (const uint8_t *network_name) SL_DEPRECATED_API_SDK_2024_6
  This function will be removed in the future versions of the Wi-SUN stack.
- Global sl_wisun_util_get_phy_config (sl_wisun_phy_config_t *phy_config) SL_DEPRECATED_API_SDK_2024_6
  This function will be removed in the future versions of the Wi-SUN stack.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

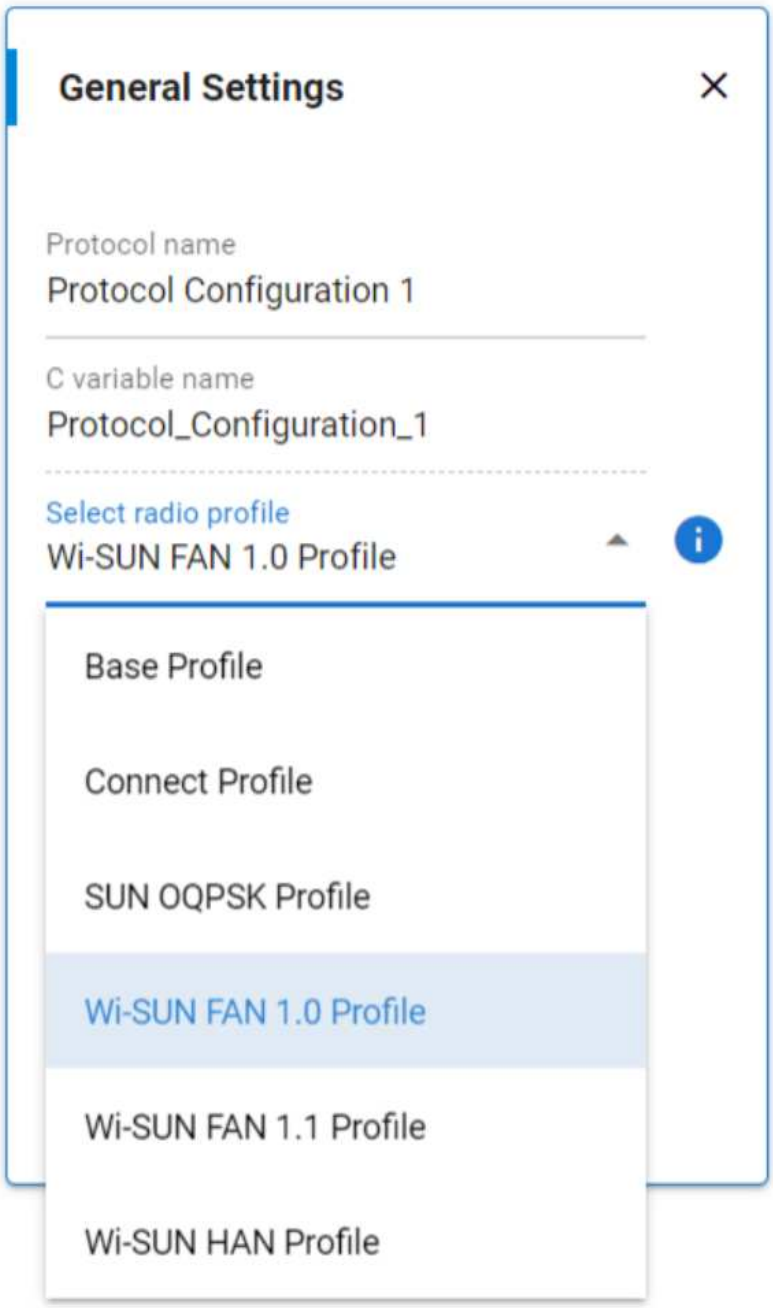
Set Up the Wi-SUN FAN 1.0 Configuration

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.

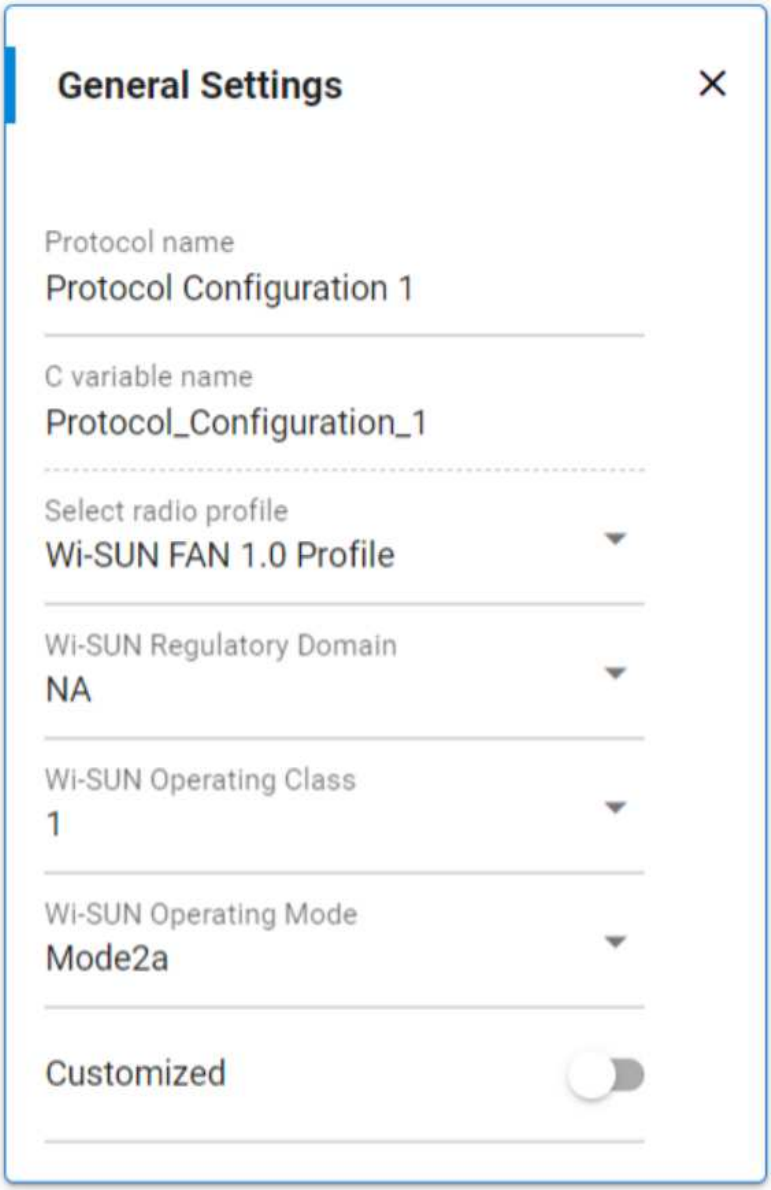
1. Select the **Wi-SUN FAN 1.0 Profile.**

Copyright © 2025 Silicon Laboratories. All rights reserved.



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.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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).

Copyright © 2025 Silicon Laboratories. All rights reserved.

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

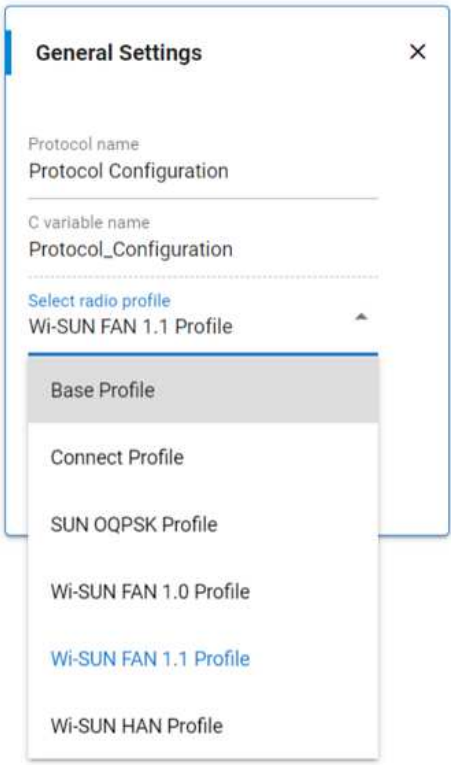
For the best performance with EFR32FG25, use the following radio board:

- BRD4270B revA06 for North America or Japan regions (902-928 MHz)
- BRD4271A revA06 for Europe region (863-876 MHz)
- BRD4272A revA03 for China (470-510 MHz)

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 Profile Specification 2V03.

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.

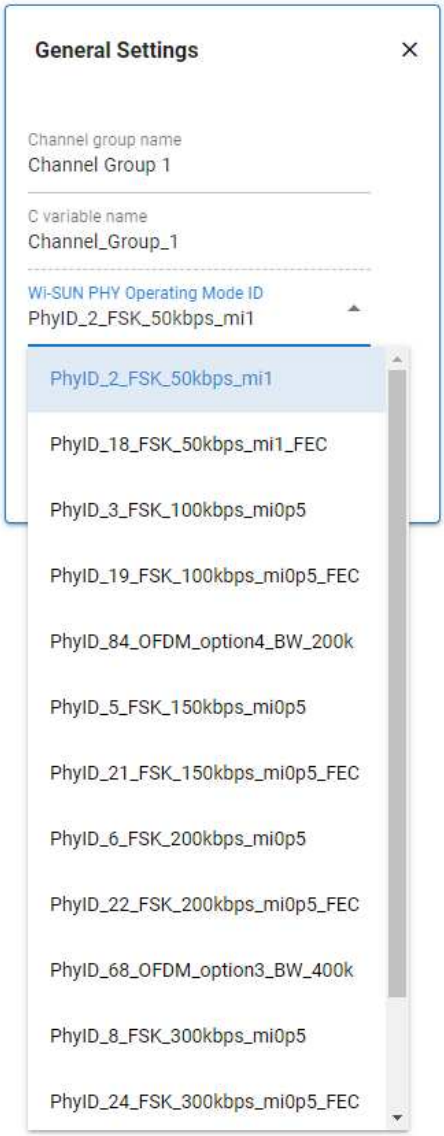


2. Select the Wi-SUN Regulatory Domain (NA, JP, EU, CN, IN, MX, BZ, AU-NZ, KR, PH, MY, HK, SG, TH, VN).

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

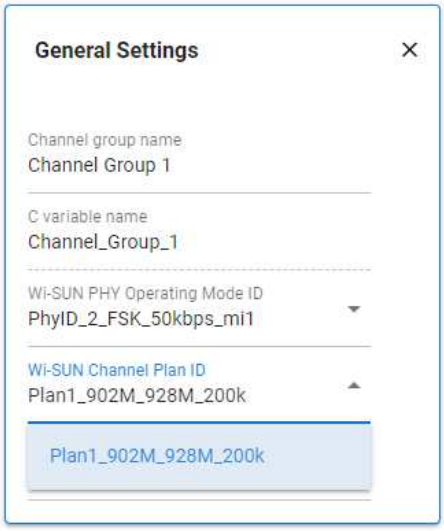
Copyright © 2025 Silicon Laboratories. All rights reserved.



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:

Copyright © 2025 Silicon Laboratories. All rights reserved.



5. Save the *radio_settings.radioconf* file and build the RAILtest project.

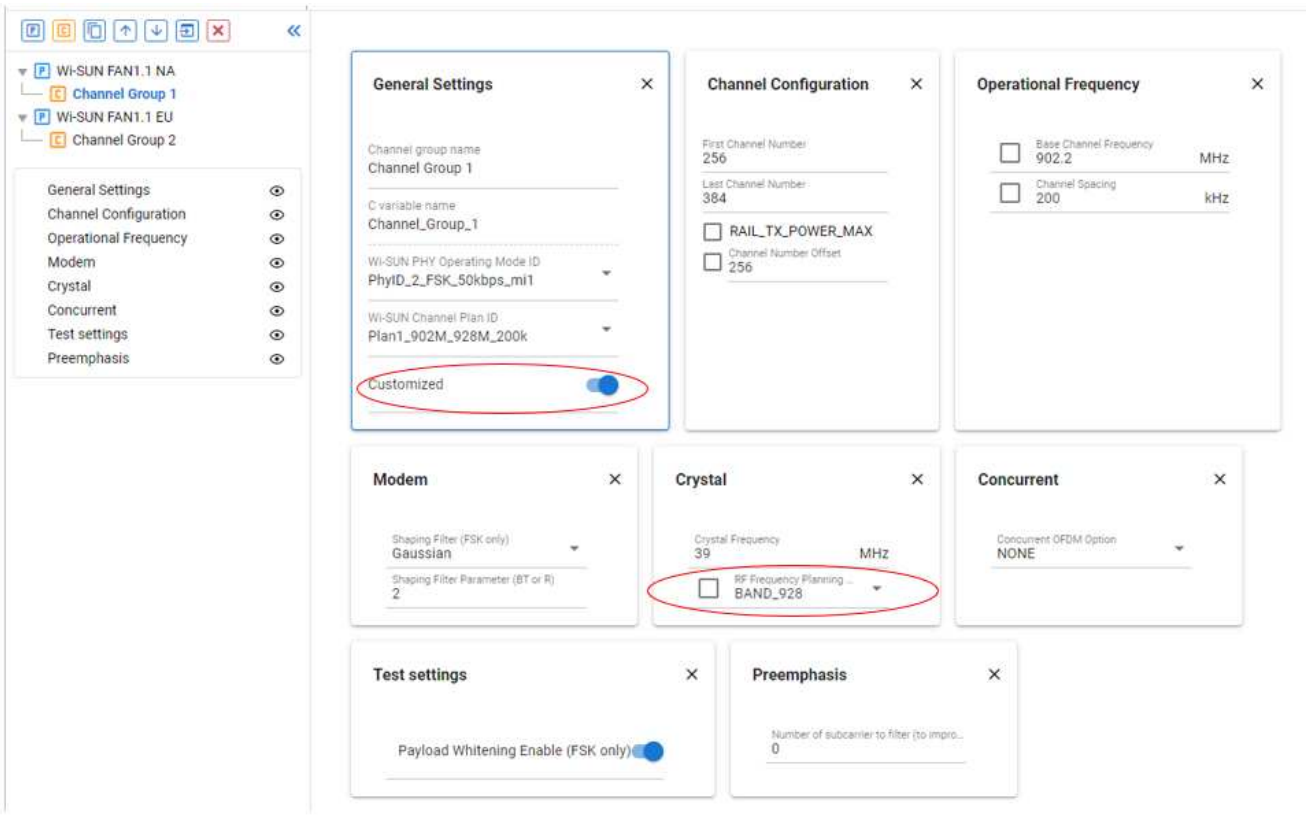
RF Frequency Planning in Multi-PHY Radio Configuration for Different Regions

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.

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.

To select the desired RF Frequency planning setting, turn on **Customized** and select the band, for example BAND_928 as shown.

Copyright © 2025 Silicon Laboratories. All rights reserved.



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

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
> 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
```

Copyright © 2025 Silicon Laboratories. All rights reserved.

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:

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
> 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

```
> 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: 0×00 0×0c 0xde 0×00 0×33 0×44 0×55 0×0f 0×77 0×88 0×99 0xaa 0xbb 0xcc 0xdd 0xee
 0×10 0×11 0×12 0×13 0×14 0×15 0×16 0×17 0×18 0×19 0×1a 0×1b 0×1c 0×1d 0×1e 0×1f 0×20 0×21 0×22 0×23 0×24 0×25 0×26
 0×27 0×28 0×29 0×2a 0×2b 0×2c 0×2d 0×2e 0×2f 0×30 0×31 0×32 0×33 0×34 0×35 0×36 0×37 0×38 0×39 0×3a 0×3b 0×3c 0×3d
 0×3e 0×3f 0×40 0×41 0×42 0×43 0×44 0×45 0×46 0×47 0×48 0×49 0×4a 0×4b 0×4c 0×4d 0×4e 0×4f 0×50 0×51 0×52 0×53 0×54
 0×55 0×56 0×57 0×58 0×59 0×5a 0×5b 0×5c 0×5d 0×5e 0×5f 0×60 0×61 0×62 0×63 0×64 0×65 0×66 0×67 0×68 0×69 0×6a 0×6b
 0×6c 0×6d 0×6e 0×6f 0×70 0×71 0×72 0×73 0×74 0×75 0×76 0×77 0×78 0×79 0×7a 0×7b 0×7c 0×7d 0×7e 0×7f 0×80 0×81 0×82
 0×83 0×84 0×85 0×86 0×87 0×88 0×89 0×8a 0×8b 0×8c 0×8d 0×8e 0×8f 0×90 0×91 0×92 0×93 0×94 0×95 0×96 0×97 0×98 0×99
 0×9a 0×9b 0×9c 0×9d 0×9e 0×9f 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:0×000000000}{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:0×0}{rssi:-97}{lqi:197}{phy:6}{isAck:False}
{syncWordId:0}{antenna:0}{channelHopIdx:254}{payload: 0×03 0×0c 0xde 0×00 0×33 0×44 0×55 0×0f 0×77 0×88 0×99 0xaa 0xbb
0xcc 0xdd 0xee 0×10 0×11 0×12 0×13 0×14 0×15 0×16 0×17 0×18 0×19 0×1a 0×1b 0×1c 0×1d 0×1e 0×1f 0×20 0×21 0×22 0×23 0×24
0×25 0×26 0×27 0×28 0×29 0×2a 0×2b 0×2c 0×2d 0×2e 0×2f 0×30 0×31 0×32 0×33 0×34 0×35 0×36 0×37 0×38 0×39 0×3a 0×3b 0×3c
0×3d 0×3e 0×3f 0×40 0×41 0×42 0×43 0×44 0×45 0×46 0×47 0×48 0×49 0×4a 0×4b 0×4c 0×4d 0×4e 0×4f 0×50 0×51 0×52 0×53 0×54
0×55 0×56 0×57 0×58 0×59 0×5a 0×5b 0×5c 0×5d 0×5e 0×5f 0×60 0×61 0×62 0×63 0×64 0×65 0×66 0×67 0×68 0×69 0×6a 0×6b 0×6c
0×6d 0×6e 0×6f 0×70 0×71 0×72 0×73 0×74 0×75 0×76 0×77 0×78 0×79 0×7a 0×7b 0×7c 0×7d 0×7e 0×7f 0×80 0×81 0×82 0×83 0×84
0×85 0×86 0×87 0×88 0×89 0×8a 0×8b 0×8c 0×8d 0×8e 0×8f 0×90 0×91 0×92 0×93 0×94 0×95 0×96 0×97 0×98 0×99 0×9a 0×9b 0×9c
0×9d 0×9e 0×9f 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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

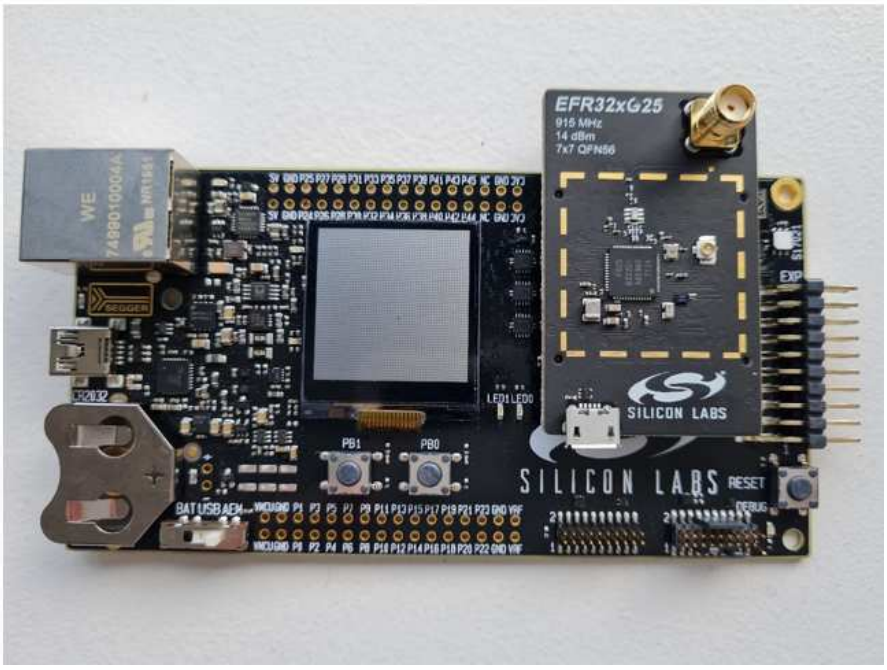
However, the EFR32FG25 radio boards have no calibrated CTune value stored in the EEPROM memory.

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!

Copyright © 2025 Silicon Laboratories. All rights reserved.

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:

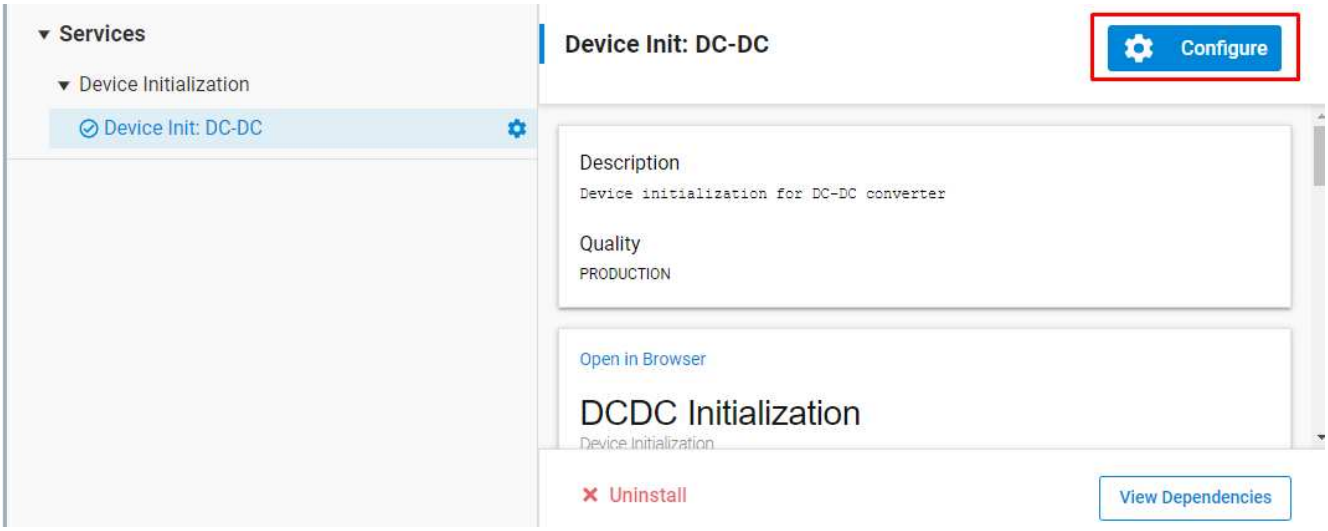
- Set up the PA configuration
- Set the recommended PA level (check the max level depending on the radio board, 16 dBm for EFR32FG25)

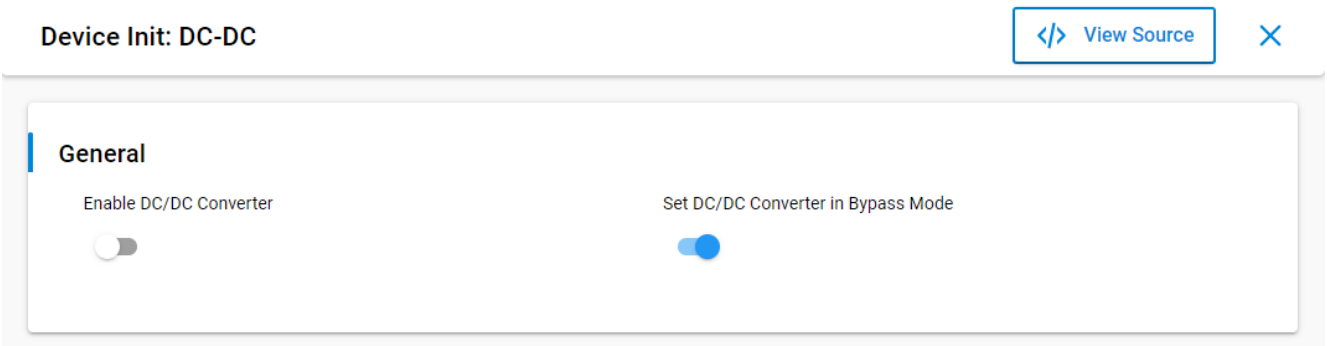
Clocking

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.

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.

Copyright © 2025 Silicon Laboratories. All rights reserved.