

Z-Wave

[Developing with Z-Wave](#)

[Release Notes](#)

[Z-Wave](#)

[Getting Started](#)

[Fundamentals](#)

[Z-Wave Developer's Guide](#)

[Overview](#)

[Evaluation Phase](#)

[Overview](#)

[Z-Wave Getting Started for End Devices \(PDF\)](#)

[Z-Wave 800 Thunderboard Quick Start Guide \(PDF\)](#)

[Z-Wave 800 Dev Kit Quick Start Guide with Unify \(PDF\)](#)

[Z-Wave Networking Basics \(PDF\)](#)

[Z-Wave Control Application Basics \(PDF\)](#)

[Z-Wave Development Basics \(PDF\)](#)

[Z-Wave Association Basics \(PDF\)](#)

[Z-Wave Multi Channel Basic \(PDF\)](#)

[Z-Wave Battery Support Basics \(PDF\)](#)

[Z-Wave Device Configuration Basics \(PDF\)](#)

[Prototype Phase](#)

[Overview](#)

[Z-Wave Plus V2 Application Framework GSDK \(PDF\)](#)

[Z-Wave Zniffer User Guide \(PDF\)](#)

[Z-Wave PC based Controller v5 User Guide \(PDF\)](#)

[Development Phase](#)

[Overview](#)

[Z-Wave 700 Series Integration Guide \(PDF\)](#)

[Z-Wave 800 Series Integration Guide \(PDF\)](#)

[Mandatory Crystal Adjustment for EFR32ZG14/EFR32ZG23 Based Products \(PDF\)](#)

[Bring-Up/Test HW Development User's Guide \(PDF\)](#)

[Manufacture Z-Wave product in volume \(PDF\)](#)

[ZGM230S Dev Kit User's Guide - 800 DevKit \(PDF\)](#)

[Porting Z-Wave Appl. SW from 500 to 700 \(PDF\)](#)

[Porting Z-Wave Appl. SW from 700 to 800 hardware \(PDF\)](#)

[Railtest User'S Guide](#)

[Introduction](#)

[Overview](#)

[Command Line Interface](#)

[Peripherals](#)

[Application Features](#)

[Protocol Specific Features](#)

[Testing Modes](#)

[Miscellaneous](#)

[Serial API Host Appl. Prg. Guide \(PDF\)](#)

[Silicon Labs Z-Wave API Reference Guide](#)

[Configuration](#)

[Command Class Association - Group and Nodes configuration](#)

[Command Class Battery Configuration](#)

[Command Class Binary Switch Configuration](#)

[Command Class Color Switch Configuration](#)

[Command Class Firmware Update configuration](#)

[Command Class Multi Channel Configuration](#)

[Command Class Multilevel Sensor Configuration](#)

[Command Class Multilevel Switch Configuration](#)

[Command Class Notification- notifications configuration](#)

[Command Class Supervision Configuration](#)

[Command Class User Code Configuration](#)

[Command Class User Credential Configuration](#)

[Command Class User Credential I/O Configuration](#)

[Command Class Wake Up Configuration](#)

[ZAF Application Name Configuration](#)

[ZAF Event Distributor Core Configuration](#)

[ZAF Event Distributor Core Configuration](#)

[Application Version Configuration](#)

[Command Class Firmware Update - Bootloader update](#)

[Command Class Firmware Update - Firmware targets](#)

[Command Class ZwavePlusInfo Configuration](#)

[Version Configuration](#)

[Command Class Manufacturer Specific](#)

[Device type](#)

[Multi Channel Configuration](#)

[Security Configuration](#)

[General](#)

[Output channels](#)

[Component filtering](#)

[Z-Wave Radio Configuration](#)

[Max Tx Power Configuration](#)

[Command Classes](#)

[\(Multi Channel\) Association](#)

[ASSOCIATION_GROUP](#)

[ASSOCIATION_GROUP_PACKED](#)

[SAssociationInfo](#)

Association Group Information

Configuration of CC Association Group Information

cc_agi_group_t

cc_agi_config_t

Basic

Battery

SBatteryData

Binary Switch

cc_binary_switch_t

CentralScene

central_scene_configuration_t

s_CC_centralScene_data_t_

_cc_central_scene_t

ColorSwitch

colorComponent_

Common

Configuration

cc_config_parameter_value_t

_cc_config_parameter_attributes_t

cc_config_parameter_metadata_t

cc_config_parameter_buffer_t

cc_configuration_t

cc_configuration_io_interface_t

_cc_config_parameter_attributes_t.flags

DeviceResetLocally

DoorLock

cc_door_lock_capabilities_report_t

cc_door_lock_configuration_t

cc_door_lock_data_t

_door_lock_hw_data_t

DoorLockEvents

Firmware Update Meta Data

Indicator

s_CC_indicator_data_t_

Multichan

cc_multi_channel_config_t

MultilevelSensor

_sensor_type

_sensor_read_result

_sensor_interface

_sensor_administration

MultilevelSwitch

cc_multilevel_switch_t

Notification

- s_CC_notification_data_t_
- _cc_notification_t
- Configuration of CC Notification
 - _cc_notification_t
- Storage for CC Notification
- Supervision
 - SUPERVISION_GET_RECEIVED_HANDLER_ARGS
 - _cc_supervision_report_event
 - Supervision Events
- UserCode
 - SUserCode
 - s_CC_userCode_data_t_
 - UserCodeEvents
 - cc_user_code_event_validate_data_t
- UserCredential
 - u3c_user_t_
 - u3c_credential_metadata_t_
 - u3c_credential_t_
 - u3c_credential_identifier_t_
 - u3c_event_data_validate_t_
 - u3c_event_data_learn_read_done_t_
 - u3c_event_data_learn_start_t_
 - u3c_admin_code_metadata_t_
- WakeUp
 - SWakeupCcData
- ZWavePlusInfo
 - zw_plus_info_config_icons_t
- ZAF
 - ZAF Interface
 - ZAF Actuator Module
 - actuator
 - Application Timer
 - SAppTimer
 - Command Classes List Generator
 - Event Manager
 - MISC
 - NVM
 - True Status Engine
 - _s_zaf_tse_data_input_template_t_
 - ApplicationEvent
 - Association Data Store
 - s_ZAF_ADS_SecurityHighestClass_t
 - CC_Invoker
 - CMD_Publisher

Command Class Utils

FILE_ID

Network

Network Learn Mode

Network Management

Retention Register

ZAF_Transport

ZAF_Transport_Endpoint

_MULTICHAN_NODE_INFO_

_MULTICHAN_NODE_INFO_PACKED_

destination_info_t

MULTICHAN_NODE_ID_PACKED

_TRANSMIT_OPTIONS_TYPE_EX_

_TRANSMIT_OPTIONS_TYPE_SINGLE_EX_

ZAF_Transport_Multicast

ZAF_Transport_Security

TYPES

_MULTICHAN_SOURCE_NODE_ID_

_MULTICHAN_DEST_NODE_ID_

MULTICHAN_DEST_NODE_ID_8bit

_RECEIVE_OPTIONS_TYPE_EX_

ZAF_TRANSPORT_TX_BUFFER

ccc_pair_t

agi_profile_t

transmission_result_t

_cc_handler_input_t

_cc_handler_output_t

CC_handler_map_t

CC_handler_map_v2_t

CC_handler_map_v3_t

CC_handler_map_v4_t

CC_handler_map_v5_t

zaf_cc_config_t

zaf_cc_config_entry_v1_t

zaf_cc_list_t

PRODUCT_ID_ENUM

ZAF_ProtocolConfig

Apps

SerialAPI

cmd_handler_map_t

_transport_t

comm_interface_frame_ptr

Zniffer

_transport_t

comm_interface_frame_ptr
tx_cmd_frame_t
tx_data_frame_t
tx_beam_start_frame_t
tx_beam_stop_frame_t

Z-Wave API

Z-Wave Serial API

Frame Types
GET_INIT_DATA_FLAG
FUNC_ID

ZW Learn Mode defines

Z-Wave Application Transport Interface

SNetworkUpdateRequest
SNodeInfoRequest
SNodeInfo
SCommandClassList_t
SCommandClassSet_t
SCommandClassVersions
SCommandGenerateRandom
SCommandNodeInfo
SCommandClearNetworkStatistics
SCommandSetLearnMode
SCommandSetSmartStartLearnMode
SSetSucNodeId
SCommandSetRfPowerLevel
SSendSucNodeId
SCommandSetPromiscuousMode
SAssignReturnRoute
SCommandSetRfReceiveMode
SDeleteReturnRoute
SCommandGeneric8bParameter
SCommandGeneric16bParameter
SCommandAreNodesNeighbours
SCommandGetRoutingInfo
SSendSlaveNodeInformation
SCommandSetRoutingInfo
SCommandStoreNodeInfo
SCommandStoreHomeID
SCommandSetPriorityRoute
SCommandAesEcb
SCommandGetPriorityRoute
SCommandFailedNodeIDCmd
SCommandPMSetPowerDownCallback
SCommandSetLBTThreshold

SCommandSetMaxInclReqInterval
SCommandNvmBackupRestore
SCommandSetSecurityKeys
SCommandEnableNodeNLS
SCommandGetNodeNLSState
SCommandSendProtocolDataCb
SCommandGetNLSNodes
SProtocolVersion
SProtocolInfo
SNetworkInfo
SLongRangeInfo
t_ExtNodeInfo
SRadioStatus
SApplicationHandles
SRadioConfig_t
SAppNodeInfo_t
SVirtualSlaveNodeInfo_t
SVirtualSlaveNodeInfoTable_t
SProtocolConfig_t
STransmitFrameConfig
STransmitProtocolFrameConfig
SExploreInclusionRequest
SExploreExclusionRequest
SIncludedNodeInfo
STest
SSendData
SSendDataEx
SSendDataBridge
SSendDataMulti
SSendDataMultiEx
SSendDataMultiBridge
SSendProtocolData
SCommandNetworkManagement
SRequestNewRouteDestinations
SCommandNetworkManagementDSK
SZWaveTransmitStatus
SZWaveGenerateRandomStatus
SZWaveNodeInfoStatus
SZWaveLearnModeStatus
SZWaveInvalidTxRequestStatus
SZWaveInvalidCommandStatus
SZWaveGeneric8bStatus
SZWaveGenericBoolStatus
SCommandRequestNodeNeighborUpdate

SCommandRequestNodeTypeNeighborUpdate
SZWaveGetRoutingInfoStatus
SCommandInitiateShutdown
SCommandSetSecurityFlags
SZWaveGetPriorityRouteStatus
SZWaveSetPriorityRouteStatus
SZWaveGetVirtualNodesStatus
SZWaveAesEcbStatus
SZWaveGetBackgroundRssiStatus
SZWaveNetworkManagementStatus
SNvmBackupRestoreStatus
SCommandGetNodeNLSStateStatus
SCommandEnableNodeNLSStatus
SZWaveGetIncludedNodes
SZWaveGetIncludedNodesLR
SZWaveGetIncludedNodesNLS
SZWaveTxPowerMaxSupported
SCommandTransferProtocolCC
SReceiveNodeUpdate
SReceiveSecurityEvent
UTransmitParameters
UCommandStatus
UReceiveCmdPayload
UCommandParameters
SZwaveTransmitPackage
SZwaveCommandStatusPackage
SZwaveCommandPackage
SReceiveSingle
SReceiveMulti
SProtocolTxOptions
SRequestEncryption
UReceiveParameters
SZwaveReceivePackage
Z-Wave Basis API
 _APPL_NODE_TYPE_
 _NODE_TYPE_
 NODEINFO
 RSSI_LEVELS
 Application Node Info
Z-Wave Security API
 _S_SECURITY_S2_INCLUSION_CSA_PUBLIC_DSK_
 _S_APPLICATION_SECURITY_EVENT_DATA_
 Security Keys Bitmask
Z-Wave System Startup API

Z-Wave Transport API

- s_rssi_val
- _S_ROUTE_LINK_T_
- _TX_STATUS_TYPE_
- _RECEIVE_OPTIONS_TYPE
- _TRANSMIT_OPTIONS_TYPE
- _TRANSMIT_MULTI_OPTIONS_TYPE
- _RECEIVE_OPTIONS_TYPE.homeld
- Transmit Options
- Receive Status
- Transmit complete codes

Z-Wave Typedefs

Z-Wave User Task API

- ZW_UserTask_Buffer_t
- ZW_UserTask_t

Components

Events

- EventDistributor
 - event_cc_t
 - zaf_event_distributor_cc_event_handler_map_v1_t
- SEventDistributor

ZAF Event Handling

MfgTokens

NodeMask

Queue

- QueueNotifying
- SQueueNotifying

Timer

- SwTimer
 - SSwTimer
 - SSwTimerPrivate_t
- SwTimerLiasion
 - SSwTimerLiaison

SyncEvent

- USyncEventFunctor
- SSyncEvent
- USyncEventRetFunctor
- SSyncEventRet
- USyncEventArg1Functor
- SSyncEventArg1
- USyncEventArg2Functor
- SSyncEventArg2

Zpal

- Zpal-bootloader

- [zpal_bootloader_info_t](#)
 - [Zpal-bootloader-info-version](#)
- [Zpal-entropy](#)
- [Zpal-init](#)
- [Zpal-misc](#)
 - [zpal_product_id_t](#)
- [Zpal-nvm](#)
- [Zpal-power-manager](#)
 - [pm_domain_status_t](#)
- [Zpal-radio](#)
 - [zpal_radio_transmit_parameter_t](#)
 - [zpal_radio_rx_parameters_t](#)
 - [zpal_radio_receive_frame_t](#)
 - [zpal_radio_network_stats_t](#)
 - [zpal_radio_rf_channel_statistic_t](#)
 - [zpal_radio_profile_t](#)
 - [Number of Channels](#)
- [Zpal-radio-utils](#)
- [Zpal-retention-register](#)
- [Zpal-status](#)
- [Zpal-uart](#)
 - [zpal_uart_config_t](#)
- [Zpal-watchdog](#)
- [Migration Guide](#)
- [Firmware Update Images and Bootloader](#)
- [How to implement a new command class](#)
- [What is a migration code block and how to use them](#)
- [True Status Module](#)
- [When and how to create a Multi Channel application](#)
- [CLI Manipulator Board Application](#)
- [How to use the .hts scripts:](#)
- [Applications](#)
- [Serial API](#)
- [Zniffer](#)
 - [Usage on Linux platform - zwave_soc_door_lock_keypad](#)
- [Door Lock](#)
 - [Usage on Linux platform - zwave_soc_led_bulb](#)
- [LED Bulb](#)
 - [Usage on Linux platform - zwave_soc_multilevel_sensor](#)
- [Multilevel Sensor](#)
 - [Usage on Linux platform - zwave_soc_power_strip](#)
- [Power Strip](#)
 - [Usage on Linux platform - zwave_soc_sensor_pir](#)
- [Sensor PIR](#)

[Usage on Linux platform - zwave_soc_switch_on_off](#)

[Switch On/Off](#)

[Usage on Linux platform - zwave_soc_wall_controller](#)

[Wall Controller](#)

[Z-Wave Command Class Configurator](#)

[Z-Wave Platform Abstraction Layer](#)

[ZPAL declaration](#)

[Silicon Labs ZPAL](#)

[Deprecated List](#)

[Important Changes](#)

[Examples](#)

[Overview](#)

[How to Use Certified Apps \(PDF\)](#)

[How to Use Pre-Certified Apps \(PDF\)](#)

[Additional Topics](#)

Developing with Z-Wave

Developing with Silicon Labs Z-Wave

Z-Wave is a wireless communication standard designed to allow smart home devices of all types and brands to “talk” and work together. It is found inside products such as lights, door locks, security systems, climate controls, window blinds and much more, connecting them together in functional and intuitive ways.



The content on these pages is intended for those who want to experiment with or are already developing a Z-Wave application using the Silicon Labs Z-Wave SDK.

For details about this release: Links to release notes are available on the [silabs.com Gecko SDK page](https://silabs.com/Gecko_SDK_page).

For Silicon Labs' Z-Wave product information: See the [product pages on silabs.com](https://silabs.com/product_pages).

For background about Z-Wave: The [Fundamentals section](#) is a good place to start.

To get started with development: See the [Getting Started section](#) to get started working with example applications.

If you are already in development: See the [Developer's Guide](#) for details or go directly to the [API Reference](#).

If you have a product ready for certification: See the [silabs.com Z-Wave certification page](https://silabs.com/Z-Wave_certification_page) for an overview and useful links.

Z-Wave

Z-Wave SDK Version 7.24.1 (July 21, 2025) - Release Notes

[Simplicity SDK Version 2025.6.1](#)

Z-Wave and Z-Wave Long Range SDK is designed to meet the demands of the future smart home and beyond, where increasing needs for more sensors and battery-operated devices require both long range and low power. Context-aware environments are the next evolution in the smart home market, and they require technologies that have been optimized specifically for these applications.

Click [here](#) for earlier releases.

Release Summary

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

Key Features

Added in 7.24.1

None.

Added in 7.24.0

- The Z-Wave stack and associated binary resources are compiled with the Link Time Optimization (LTO) option enabled.
- S2v2 draft implementation supports the Supervision Command Class and is aligned with the latest proposed specification.
- ZPAL watchdog refactoring.
- Z-Wave Power Manager refactoring.

API Changes

Added in 7.24.1

The duration and delay configuration macros for the Button Press component were updated to include the time unit in their names.

Added in 7.24.0

- The watchdog is not enabled by stack anymore. It is the application's duty to enable it using `zpal_watchdog_init` and `zpal_enable_watchdog`. Application tasks should not block other tasks for more than 4 seconds.
- Introduction of `zw_power_manager_init()`, `zw_power_manager_lock()`, `zw_power_manager_relock()`, `zw_power_manager_lock_cancel()`, to replace and wrap the ZPAL power manager functions. An application must use the `zw_power_manager` APIs rather than their ZPAL counterparts.
- Aligned User Credential Command Class according to the 2025A specification changes by modifying the API of the `CC_UserCredential_move_credential_and_report()` function. The specification for the "User Credential Association Set" and "User Credential Association Report" commands was updated: the option to move a credential to a different slot was removed. Moving a credential to a different slot is still possible via the CLI; however, the lifeline group can no longer be notified about a credential slot change.
- App initialization is changed. `main.c` has been removed from the sample applications and is now generated automatically by the *System Setup (sl_main)* platform component instead of the `sl_system`. For more information, refer to the documentation for [System Initialization and Action Processing](#).

- The Debug Print component was replaced by the ZPAL Log API, which allows filtering log messages by source component and log level.

Bug Fixes

Below are important bugs fixed in the release. For a complete list, refer to [this section](#).

Fixed in 7.24.1

- Fixed an issue preventing existing Z-Wave end devices updated to SDK version 7.24.0 via Over-the-Air firmware update to upgrade beyond this version.
- Fixed an issue where a FLiRS end device would send a corrupted packet if it was not woken up by an EM2/EM3 capable source.

Fixed in 7.24.0

- Fixed an issue where the radio layer would stop receiving packet until the next TX radio event (1409388).
- Fixed a rare occurrence where a null pointer led to a soft reset in the ZAF event queue (1425779).

Chip Enablement

None.

Key Features

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

New Features

Added in 7.24.1

None.

Added in 7.24.0

- The Z-Wave stack and associated binary resources are compiled with the Link Time Optimization (LTO) option enabled. (Quality: GA)
- S2v2 draft implementation supporting the Supervision Command Class and aligned with the latest proposed specification. (Quality: Alpha)
- Z-Wave Applications now support S2v2 commands NLS Node List Get/Report. (Quality: Alpha)

Enhancements

Added in 7.24.1

Improved the Series 800 radio reception capability in environments crowded by wake-up beams.

Added in 7.24.0

- Improved Clear Channel Assessment (CCA) reliability. Previously, only the latest measured RSSI value was used instead of the highest value in the RX window.
- Updated Clock Manager API calls to the new SL clock_manager module API calls.
- Replaced previous logging approach with new logging system via ZPAL Log API, introducing component based enablement, log levels, and routing capability to different output channels.
- Made NVM initialization significantly faster when many user codes are used with User Code Command Class by creation of `ZAF_FILE_ID_CC_USER_CODE_HEADER` to store a new database. The database stores flags about the validity of the corresponding User Code. This removes the limitation of `CC_USER_CODE_MAX_IDS`, increasing its range to 255.
- Added more descriptive error messages for invalid .cc_config file contents.

- Modularized the User Credential Command Class handler functions from `CCUserCredential.c` . Related upstream issue in Z-Wave Alliance Open Source: <https://github.com/Z-Wave-Alliance/z-wave-stack/issues/351>.
- Adapted the User Credential Command Class behavior according to Z-Wave specification rule change, where the `CC:0083.01.0C.11.015` requirement has been removed. Credential Get commands where the UUID does not match the database is answered with a "Credential doesn't exist" type of response.
- Merged Zniffer NCP and Zniffer PTI into a single sample application (zwave_ncp_zniffer).

Removed Features

None.

Deprecated Features

None.

API Changes

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

New APIs

New 7.24.1

None.

New 7.24.0

New API Signature	Deprecated API replaced by this (if any)
REGISTER_CC_V6(cc_api_handler_version, cc, version, handler, basic_set_mapper, basic_get_mapper, lifeline_report_cb, flags, init_cb, reset_cb, migrate_cb)*	None
zpal_status_t zpal_pm_register_domain(zpal_pm_domain_t domain, pm_domain_status_t *radio_domain_status, pm_domain_status_t *deep_sleep_domain_status)	
zpal_status_t zpal_pm_lock(zpal_pm_type_t type, zpal_pm_domain_t domain, uint32_t timeout_ms, uint8_t idx)	zpal_pm_stay_awake**
zpal_status_t zpal_pm_relock(zpal_pm_type_t type, zpal_pm_domain_t domain, uint32_t timeout_ms, uint8_t idx)	zpal_pm_stay_awake**
zpal_status_t zpal_pm_lock_cancel(zpal_pm_type_t type, zpal_pm_domain_t domain, uint8_t idx)	zpal_pm_cancel**
void zpal_pm_lock_cancel_all(void)	zpal_pm_cancel_all**
uint32_t zpal_get_max_timeout(void)	
bool zpal_pm_lock_is_active(zpal_pm_type_t type, zpal_pm_domain_t domain, uint8_t idx, zpal_pm_state_t *state)	zpal_pm_is_active**
bool zpal_pm_lock_type_is_active(zpal_pm_type_t pm_type, zpal_pm_state_t *state)	
zpal_status_t zpal_pm_register_domain(zpal_pm_domain_t domain, pm_domain_status_t *radio_domain_status, pm_domain_status_t *deep_sleep_domain_status)	

* The `REGISTER_CC_V6` macro has been introduced for extending the `REGISTER_CC_V5` in two ways: It introduces `migrate()` callback function similar to `init()` or `reset()` . `migrate()` can be used to execute any migration that is necessary for a command class during firmware upgrade. `migrate()` is called by Firmware Update Command Class, after the new firmware is loaded, but before the Firmware Update Status Report is transmitted. It allows the command class to choose the `cc_api_handler_version` . This allows command classes to update their registration to `REGISTER_CC_V6()` without the need to change the CC handlers.

** API not deprecated but removed

Modified APIs

Modified 7.24.1

Old API	Modified
MIN_VALID_BUTTON_PRESS_DURATION	MIN_VALID_BUTTON_PRESS_DURATION_MS
SHORT_BUTTON_PRESS_DURATION	SHORT_BUTTON_PRESS_DURATION_MS
MEDIUM_BUTTON_PRESS_DURATION	MEDIUM_BUTTON_PRESS_DURATION_MS
LONG_BUTTON_PRESS_DURATION	LONG_BUTTON_PRESS_DURATION_MS
APP_BUTTON_PRESS_WAKEUP_DELAY	APP_BUTTON_PRESS_WAKEUP_DELAY_MS

Modified 7.24.0

Old API	Modified
u3c_db_operation_result CC_UserCredential_move_credential_and_report(u3c_credential_type credential_type, uint16_t source_credential_slot, uint16_t destination_uuid, uint16_t destination_credential_slot RECEIVE_OPTIONS_TYPE_EX * p_rx_options)	u3c_db_operation_result CC_UserCredential_move_credential_and_report(uint16_t credential_slot, uint16_t destination_uuid, RECEIVE_OPTIONS_TYPE_EX * p_rx_options)
u3c_user	u3c_user_t
u3c_credential	u3c_credential_t
u3c_event_data_validate	u3c_event_data_validate_t
u3c_event_data_learn_read_done	u3c_event_data_learn_read_done_t
u3c_credential_metadata	u3c_credential_metadata_t
u3c_credential_learn_event_data	u3c_event_data_learn_start_t
- DPRINT(PSTRING) - DPRINTF(PFORMAT, ...)	- ZPAL_LOG(COMPONENT, LEVEL, PFORMAT, ...) - ZPAL_LOG_DEBUG(COMPONENT, PFORMAT, ...) - ZPAL_LOG_INFO(COMPONENT, PFORMAT, ...) - ZPAL_LOG_WARNING(COMPONENT, PFORMAT, ...) - ZPAL_LOG_ERROR(COMPONENT, PFORMAT, ...)

Removed APIs

Removed 7.24.1

None.

Removed 7.24.0

Removed API Name	Was Deprecated?
zpal_pm_stay_awake	no
zpal_pm_register	no
zpal_pm_unregister	no
zpal_pm_is_active	no
zpal_pm_cancel	no
zpal_pm_cancel_all	no
zpal_pm_unregister_all	no
DPRINT(PSTRING)	no
DPRINTF(PFORMAT, ...)	no

Deprecated APIs

None.

Bug Fixes

Fixed in 7.24.1

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1468819	Fixed an issue preventing existing Z-Wave end devices updated to SDK version 7.24.0 via Over-the-Air firmware update to upgrade beyond this version.	None	ZPAL
1465034	Fixed an issue where a FLiRS end device would send a corrupted packet if it was not woken up by an EM2/EM3 capable source.	None	ZPAL
1476826	Fixed CMake compiler issue.	None	All Z-Wave applications
1473709	Fixed compiler issues	None	zwave_soc_led_bulb, zwave_soc_power_strip apps on boards with RGB LED
1473325	Slow refresh has been enabled by default in Central Scene V3 according to the specification.	None	ZAF / Central Scene Command Class
1471384	Fixed User Credential reports not being sent to all Lifeline destinations. Now the CSR_LifelineMandatoryReports_Rev03 CTT test passes.	None	ZAF / User Credential Command Class

Fixed in 7.24.0

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1409388	Fixed an issue where the radio layer would stop receiving packets until the next TX radio event.	None	ZPAL/RAIL
1390101	Fixed an issue preventing the configuration of a TX output power above +14 dBm in the Serial API controller application.	None	ZPAL
1424148	Better randomization of the delay used to send retransmissions.	None	ZPAL
1257690	Fixed bootloader slot size configuration not being modifiable via the graphical user interface	None	ZPAL
1380945	Fixed an issue where, under heavy traffic, the packets were not properly ordered when sent over the air. Acknowledgment packets could be sent in reverse. Retransmission attempts might occur after new packets.	None	Transport layer
1383828	Sleeping reporting applications now correctly detect the button press duration when woken up from a deep sleep state via a button press.	None	Reporting Sleeping End Node devices
1425779	Fixed a rare occurrence where a null pointer led to a soft reset in the ZAF event queue.	None	ZAF / TSE
1426510	Fixed an issue when notifications were not sent out sometimes during an OTA firmware update.	None	ZAF / Notification Command Class
1383233	Max number of user code limitation has been restored to 255. The unsuccessful inclusion issue in case of higher configuration value has been fixed.	None	ZAF / User Code Command Class
1414398	Fixed an issue when the report to the lifeline was invalid in case of multiple User IDs were used and all user codes were erased.	None	ZAF / User Code Command Class

ID	Issue Description	GitHub / Salesforce Reference (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1402601	Protect against configuring an endpoint for CC AGI with more association groups than allowed by the configuration of Association Command Class.	github.com/Z-Wave-Alliance/z-wave-stack/issues/385	ZAF / Association Command Class
1388187	Fixed not being able to add or modify User Credential Command Class credentials locally (e.g. via the CLI).	None	ZAF / User Credential Command Class
1388186	Fixed an issue when the DUT rejects modification of a password credential. The UserCredentialCmdClassV1_Rev01 CTT test now passes successfully.	None	ZAF / User Credential Command Class
1396326	Fixed an issue when the lifeline group contains only the multi channel endpoint(s) of a node, a User Credential Command Class / User Set request from this node's root device were not answered - instead, only the registered endpoint(s) received the report. With this fix the CCA_U3CReportUserData_Rev01 CTT test now passes successfully.	None	ZAF / User Credential Command Class
1381226	Fixed Supervision status in response to a User Set when the database is already in the requested state and U3C is used - now reports "Success".	None	ZAF / User Credential Command Class
1385661	Changing the state of the door bolt is no longer possible when the device is not included in a network.	None	ZAF / User Code Command Class
1440915	Fixed the number of indicator blinks from 5 to 6 to pass the CDR_ZWPv2IndicatorCCRequirements_Rev01 CTT test.	None	All Z-Wave applications
1377838	The inclusion indicator LED no longer keeps blinking when a KEX frame is missing during the inclusion process.	None	All Z-Wave applications
1390955	Fixed the GPIO configuration of Serial API Controller UART RX and TX pins	None	zwave_ncp_serial_api_controller
1394004	The SAPI command REMOVE_NODE_FROM_NETWORK could falsely fail if removing a device from another network that has the same node ID as a device in the remover's network.	None	zwave_ncp_serial_api_controller
1383938	Z-Wave OTW bootloader project for custom board now compiles without errors, displays warning to pay attention to default UART configuration.	None	bootloader-uart-xmodem-zwave-otw
1384692	Fixed an issue when the `get_rgb_values` CLI command returned with invalid values.	None	zwave_soc_power_strip
1429395	Fixed an issue when the sleeping apps fell back to sleep during long button press.	None	Reporting Sleeping End Node devices
1384690	Fixed an issue when the CLI were not working with some board-application combinations.	None	- BRD2603A - zwave_soc_wall_controller - BRD2603A - zwave_soc_multilevel_sensor - BRD2705A - zwave_soc_wall_controller

Chip Enablement

None.

Application Example Changes

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

New Examples

None.

Modified Examples

Modified 7.24.1

Example Name	Changes	Supported Software Variants if applicable	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
All example apps	The configuration names for the App Button Press component now include the unit for durations (an _MS postfix was added).	All example apps	- ModeName - ModeName	- OPN: All Z-Wave supported 800 series OPNs - All Z-Wave supported 800 series boards	

Modified 7.24.0

Example Name	Changes	Supported Software Variants if applicable	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
Zniffer Readme	The Zniffer application now includes the "Zniffer PTI" sample application capabilities	zwave_ncp_zniffer		- OPN: All Z-Wave supported 800 series OPNs - All Z-Wave supported 800 series boards	
zwave_soc_door_lock_keypad Readme	- The default bolt state of the Door Lock Keypad application is changed to unlocked. - Changing the state of the door bolt is no longer possible when the device is not included in a network.	zwave_soc_door_lock_keypad		- OPN: All Z-Wave supported 800 series OPNs - All Z-Wave supported 800 series boards	

Example Name	Changes	Supported Software Variants if applicable	Supported Modes	Supported OPNs / Boards / OPN Combinations	Supported Host Interfaces
zwave_soc_door_lock_keypad Readme	- CLI is enabled in EM2 by modifying the EUSART baudrate to 9600 baud. - On development kit boards BRD2603A and BRD2705A, the following command should be applied in the Admin console to adjust the baudrate: `serial vcom config speed 9600`	zwave_soc_door_lock_keypad		- OPN: All Z-Wave supported 800 series OPNs - All Z-Wave supported 800 series boards	
zwave_soc_led_bulb Readme	Added `toggle_led_bulb` CLI command.	zwave_soc_led_bulb		- OPN: All Z-Wave supported 800 series OPNs - All Z-Wave supported 800 series boards	

Removed Examples

Removed 7.24.1

None.

Removed 7.24.0

Removed Example Name	Was Deprecated?
Zniffer PTI	No. Features merged in the "Zniffer" sample application.

Deprecated Examples

None.

Known Issues and Limitations

ID	Issue or Limitation Description	GitHub / Salesforce Reference (if any)	Workaround (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
1399709	Reported S2 command class version remains 1 until certification covers S2v2.		Currently not available.	All Z-Wave applications

ID	Issue or Limitation Description	GitHub / Salesforce Reference (if any)	Workaround (if any)	Affected Software Variants, Hardware, Modes, Host Interfaces
369430	All S2 multicast frames are sent using verified delivery S2_TXOPTION_VERIFY_DELIVERY whether or not a response is expected.		Change source code depending on the frame sent.	ZAF
1172849	On Series 800, the sleep state does not take advantage of EM1P current savings.		Currently not available.	800 Series chips
1067228	Zniffer on BRD4204D does not detect LR wakeup beams		Use different board for sniffing LR wakeup beams	zwave_ncp_zniffer
1432659	Too many reports are sent when multiple buttons are pressed at once.		Currently not available.	End Device sample apps
1478820	If multiple lifeline reports are simultaneously queued, some of them are occasionally not sent to all destinations. This can be triggered by the controller sending 2 different User Set frames without waiting for a response to the first one, with a delay smaller than 200 ms between the frames.		Currently not available.	End Device sample apps

Impact of Release Changes

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

Impact Statements

None.

Migration Guide

Click [here](#) for the migration guide for deprecated, removed, and modified items.

Using This Release

[Installation and Use](#) | [Help and Feedback](#)

Installation and Use

To upgrade your existing software with this release, see instructions in the [Migration Guide](#).

To run your first demo, see our [Z-Wave Getting Started for End Devices](#).

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

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](#).

Getting Started

Getting Started with Z-Wave Development

To get started with Z-Wave development, download the Simplicity Studio Development environment as described in the [Simplicity Studio 5 User's Guide](#).

This will also prompt you to install the Gecko SDK (GSDK), which contains the Z-Wave SDK. The GSDK combines Silicon Labs wireless software development kits (SDKs) and Gecko Platform into a single, integrated package. The GSDK is your primary tool for developing in the Silicon Labs IoT Software ecosystem. All of Silicon Labs' stacks are written in-house to provide a seamless experience from silicon to tools, allowing you to unlock powerful features with ease, including:

- Abstraction of complex requirements like multiprotocol and pre-certification
- Industry-leading ability to support a large number of nodes
- Ultra-low power consumption
- Strong network reliability

Silicon Labs also helps future-proof your devices with over-the-air software and security updates, helping to minimize maintenance cost and improve your end user product experience!

Note: The recommended method to get started with the GSDK is to first install Simplicity Studio 5, which will set up your development environment and walk you through the GSDK installation. Alternatively, GSDK and other required tools may be installed manually from the [GitHub GSDK site](#).

Resources

Once you have downloaded Simplicity Studio and the GSDK, resources to help you get started are available in the [Evaluation Phase section](#). These include:

- **Z-Wave Getting Started for End Devices**: Introduction to development of end devices using the Development Kits, Simplicity Studio and other software tools (e.g., PC Controller, Zniffer) provided by Silicon Labs. Provides guidance for setting up the hardware and software environment, running and debugging a sample application, creating a Z-Wave network, capturing the RF communication frames, and measuring the power consumption.
- **Z-Wave 800 Thunderboard Quick Start Guide**: Quick hardware overview of the Z-Wave 800 Development Kit that helps to locate the buttons, LEDs, and sensors. Provides the first steps to start the development using this board.
- **Z-Wave 800 Dev Kit Quick-Start Guide with Unify**: Describes the steps to quickly get up and running with the Z-Wave 800 Series Development Kit with Unify environment.

The Z-Wave SDK comes with a number of examples that you can use as a starting point for your own development. See the [Examples section](#) for more information.

Fundamentals

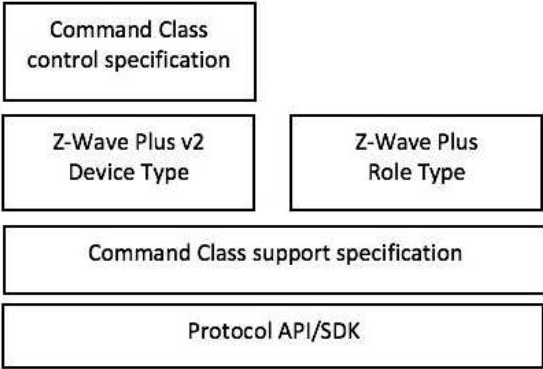
Z-Wave Fundamentals

Z-Wave Specification

The Z-Wave specification defines the details of Z-Wave communication. The documents are structured in several layers, and each document defines the requirements for a given building block or layer of a Z-Wave application. The specifications that are applicable to a product must be followed to pass certification. Certification is required before a product can be sold on the market.

The main Z-Wave specification page is located at: <https://www.silabs.com/wireless/z-wave/specification>.

The specifications for Z-Wave Plus v2 devices are shown in the following figure:



All communication within the Z-Wave network is organized into Command Classes. Command Classes are groups of commands and responses related to a certain function of a device.

Command Class Control Specification

The Command Class Control Specification describes the requirements associated with the control of Z-Wave Command Classes. It contains a list of requirements applying to all Z-Wave compliant nodes controlling a given Command Class.

For more information, see [Z-Wave Specifications](#).

Z-Wave Plus v2 Device Type and Role Type Specifications

The Z-Wave Plus v2 Device Type and Z-Wave Plus Role Type describe the different types of devices that can be part of a Z-Wave network and the different types of network behavior these devices may adopt. It also standardizes the way in which these devices announce their primary purpose and other capabilities to the rest of the network. When combined, the Device and the Role Type specifications help to ensure devices in the network know how, when, and what to communicate, as well as what to expect from other devices in the network.

For more information, see [Z-Wave Specifications](#).

Together with the Command Class specification, this provides the foundation on which Z-Wave interoperability is built.

Z-Wave Command Class Specification

The Z-Wave Command Class Specifications describe standardized messages defined in Z-Wave. This is the “language” that Z-Wave devices speak. The various Command Classes define how commands and information are carried to, from, and

between devices; for example, the command for switching a light bulb on or how to request the energy metering data from the last 10 minutes that has been gathered by an electrical socket.

For more information, see [Z-Wave Specifications](#).

The following are four types of command classes.

- Application command classes define application level functionality supported by any node regardless of device type or role.
- Management command classes define node administration and management. These classes may be mandated by the device type or role.
- Encapsulation command classes define command classes used to encapsulate other command classes.
- Network command classes define functionality related to RF or Z/IP gateway, and controller specific command classes.

For more information, see [Z-Wave Specifications](#).

Protocol API/SDK

The protocol API/SDK is a software implementation running on Silicon Labs chips. It enables the developer to use the Z-Wave protocol with API functions and therefore build applications quickly. The SDK is distributed in [Simplicity Studio](#).

General Information About Z-Wave

The following are recommended Z-Wave resources for your first encounter with Silicon Labs Z-Wave products. They provide an overview of the Z-Wave technology, describe available development tools, and can help you to choose the right development kit.

- [Z-Wave Main page](#): Z-Wave technology introduction from end-user perspective. Provides a simple overview of the benefits of Z-Wave, typical applications, and available products.
- [Silicon Labs' Z-Wave products page](#): Ideal starting point to look for hardware and software resources for Z-Wave product development or to compare different development kits.
- [Introduction to Z-Wave](#): Quick technical overview of Z-Wave
- [Z-Wave 700 Starter Kit Overview](#): Lists the kit contents and features, relevant technical documents, and the steps for getting started with development.
- [Z-Wave 800 Pro Kit Overview](#): Lists the kit contents and features, relevant technical documents, and the steps for getting started with development.
- [Z-Wave 800 Series Development Kit Overview](#): Lists the kit contents and features, relevant technical documents and the steps for getting started with development.

Overview

Z-Wave Developers Guide

The Developer's Guide content is organized in the following topics:

- **Evaluation Phase:** Contains recommended Z-Wave resources that provide basic information about Z-Wave technology and features, getting started with the development environment, and trying out the Z-Wave sample applications.
- **Prototype Phase:** Contains recommended Z-Wave resources for development possibilities and provides guidance on making basic customizations.
- **Development Phase:** Contains recommended Z-Wave resources for more advanced development possibilities to help you create custom Z-Wave products based on Z-Wave hardware.

Overview

Evaluation Phase

The following are recommended Z-Wave resources that provide basic information about Z-Wave technology and features, getting started with the development environment and trying out the Z-Wave sample applications.

Getting Started Guides

Name	Topic	Content
Z-Wave Getting Started for End Devices	End Devices	Introduction to development of end devices using the Development Kits, Simplicity Studio and other software tools (e.g., PC Controller, Zniffer) provided by Silicon Labs. Provides guidance for setting up the hardware and software environment, running and debugging a sample application, creating a Z-Wave network, capturing the RF communication frames, and measuring the power consumption.
Z-Wave 800 Thunderboard Quick Start Guide	800 Series Dev Kit	Quick HW overview of the Z-Wave 800 Dev Kit that helps to locate the buttons, LEDs and sensors. Provides the first steps to start development using this board.
Z-Wave 800 Dev Kit Quick-Start Guide with Unify	800 Series Dev Kit	Describes the steps to quickly get up and running with the Z-Wave 800 Series Development Kit with Unify environment.

'The Basics' Series

Name	Topic	Content
Z-Wave Networking Basics	Networking	Easy-reading introduction to Z-Wave networking functionality. This document introduces network management, routing and service discovery.
Z-Wave Control Application Basics	Control Application	Easy-reading introduction to control application design.
Z-Wave Development Basics	Development basics	Easy-reading introduction to Z-Wave development. This document introduces the Z-Wave Specifications and how they are to be used. Refer to the other application notes in the "Basics" series for more networking and application specific topics.
Z-Wave Association Basics	Association	Introduction to Device Type and Command Class specification on Association and Lifeline functionality. The document also covers association of Multi-Channel devices.
Z-Wave Multi Channel Basics	Multi Channel	Easy-reading introduction to Device Type and Command Class specification on Multi Channel functionality. This document introduces basic Multi Channel functionality. Association and Lifeline is introduced in "Z-Wave Association Basics".

Name	Topic	Content
Z-Wave Battery Support Basics	Battery Support	Easy-reading introduction to Device Type and Command Class specification on battery support. The document covers WakeUp and FLiRS beaming functionality.
Z-Wave Device Configuration Basics	Device Configuration	Best practice guidelines for supporting configuration parameters.

Firmware Update

Name	Topic	Content
700 OTA update an End-Device	OTA	Knowledge Article providing step-by-step guidance for performing an Over-The-Air (OTA) update of the application firmware on a Z-Wave 700 series end-device.
How to OTA a Z-Wave 800 series End-Device	OTA	Knowledge Article providing step-by-step guidance for performing an Over-The-Air (OTA) update of the application firmware on a Z-Wave 800 series end-device.
700/800 OTW update a Controller	OTW	Knowledge Article providing step-by-step guidance for performing an Over-The-Wire (OTW) firmware update for Z-Wave 700/800 series controller.

Overview

Prototype Phase

This content contains recommended Z-Wave resources for development possibilities, and provides guidance on making basic customizations. As well as the content listed below, the [Dr Z-Wave blog](#) contains a lot of useful information and practices about Z-Wave technology and development. |

User Guides

Name	Topic	Content
Z-Wave Plus V2 Application Framework GSDK	ZAF	This document describes the Z-Wave Plus V2 Application Framework (ZAF), which is part of the Z-Wave SDK. The Application Framework delivers Command Class modules that can be reused between applications and also some additional modules that make it simpler to develop firmware for new Z-Wave products and devices.
Z-Wave Zniiffer User Guide	Zniiffer	Describes the Z-Wave Zniiffer application, a development tool used for capturing Z-Wave RF communication and presenting the frames in a graphical user interface on a PC.
Z-Wave PC based Controller v5 User Guide	PC Controller	Describes the Z-Wave PC-based Controller application, a development tool used for communication with Z-Wave nodes through a Static Controller (SC).

Associations

Name	Topic	Content
Z-Wave Associations	Association	Z-Wave Associations explained with the following content (16 min video): Theory, examples using PC Controller and Zniiffer

Long Range

Name	Topic	Content
Silicon Labs Z-Wave Training - LR Introduction (2 video presentations)	Long Range	Z-Wave Long Range Introduction + Sensor applications (30 min video) Z-Wave Long Range Hands-On Lab (26 min video)
Z-Wave LR Intro + Sensor apps	Long Range	Z-Wave Long Range Introduction + Sensor applications (30 min video)
Z-Wave LR Hands On - Lab by Eric Ryherd	Long Range	Z-Wave Long Range Hands-On Lab (26 min video)
How to set up LR	Long Range	Knowledge article explaining how to set up Z-Wave Long Range with the Z-Wave 700 Starter Kit.

700 Series

Name	Topic	Content
Z-Wave 700: How to use Software Timers	700 Series Timers	Knowledge article demonstrating an example of using a Software Timer to perform a temperature and humidity sensor reading once every second. It uses the AppTimer module from the Z-Wave Application Framework.
Z-Wave 700: How to implement I2C into the Z-Wave Embedded Framework	700 Series I2C	Knowledge article showing how to interface to the Si7021 sensor using the I2C API from EMLIB and demonstrating how to get I2C implemented in the Z-Wave framework. The Si7021 Relative Humidity and Temperature Sensor used here is already mounted on the WSTK mainboard that is part of the Z-Wave 700 Development Kit.
Z-Wave 700: How to implement SPI into the Z-Wave Embedded Framework	700 Series SPI	Knowledge article showing an implementation of an SPI Master application. In order to implement SPI into the Z-Wave framework, it uses the SPIDRV API from EMDRV peripheral driver.

Overview

Development Phase Overview

This section contains recommended Z-Wave resources for more advanced development possibilities to help create custom Z-Wave products based on Z-Wave hardware.

Name	Topic	Content
Z-Wave specifications	Spec	Link for the latest Z-Wave specifications
CRT-100: Z-Wave Certification Process	Certification	Training covering the certification requirements for customers using Z-Wave in their products (16 min video). The beginning provides an overview while the remainder walks through the web tool that customers use during the steps of their certification.
Z-Wave 700 Series Integration Guide	700 Series	Provides an implementation guide for integrating Z-Wave 700 series devices into product designs. It is intended for product design engineers who aim for a fast integration of Z-Wave 700 series devices. Gives guidance on programming and debug interfaces, crystal calibration, and matching network.
Z-Wave 800 Series Integration Guide	800 Series	Provides an implementation guide for integrating Z-Wave 800 series devices into product designs. It is intended for product design engineers who aim for a fast integration of Z-Wave 800 series devices. Provides an overview of the Z-Wave 800 series device portfolio. Gives guidance on programming and debug interfaces, crystal calibration, and matching network.
Mandatory Crystal Adjustment for EFR32ZG14/EFR32ZG23 Based Products User's Guide	Crystal adjustment	Describes the mandatory adjustment of the system crystal that must be performed on a product based on the EFR32ZG14 and EFR32ZG23 devices. The procedure is required to ensure that the crystal tolerances of the product are within the specifications of the Z-Wave protocol.
Bring-Up/Test HW Development User's Guide	700/800 Series HW	Describes how to use the RailTest software to perform RF bring-up tests and validate a design based on the Z-Wave 700 series (ZGM130S, EFR32ZG14) and 800 series devices (ZGM230S, EFR32ZG23).
Manufacture Z-Wave product in volume	Manufacturing	Describes the manufacturing test flow for Z-Wave 700/800 SoC-based products. Lists the necessary tests and the describes the method for writing the manufacturing keys
EFR32ZG14 Z-Wave 700 Modem SoC Data Sheet	700 Series SoC	Datasheet for EFR32ZG14 Z-Wave 700 Modem SoC
ZGM130S Z-Wave 700 SiP Module Data Sheet	700 Series SiP	Datasheet for the ZGM130S Z-Wave 700 SiP Module
EFR32ZG23 Wireless SoC Family Data Sheet (800)	800 Series SoC	Datasheet for the EFR32ZG23 Z-Wave 800 Modem SoC family

Name	Topic	Content
ZGM230S Z-Wave 800 SiP Module Data Sheet	800 Series SiP	Datasheet for the ZGM230S Z-Wave 800 SiP Module
ZGM230S Z-Wave 800 SiP Module Radio Board User's Guide	800 Series SiP	User's guide for the ZGM230S Z-Wave 800 SiP Module Radio Board (BRD4205B), which is a plug-in board for the Wireless Pro Kit Mainboard (BRD4002A) and the Wireless Starter Kit Mainboard (BRD4001A).
ZGM230S Dev Kit User's Guide - 800 DevKit	800 Series Dev Kit	User's Guide for the ZGM230S Dev Kit, which is a low-cost, small form-factor development and evaluation platform for the ZGM230S Z-Wave SiP Module (System-in-Package Module).
Porting Z-Wave Appl. SW from 500 to 700	Porting	Provides guidelines to Z-Wave application developers for porting applications based on the Z-Wave Application Framework (ZAF) from 500 series SDKs to 700 series SDKs.
Porting Z-Wave Appl. SW from 700 to 800 hardware	Porting	Provides guidelines to Z-Wave application developers for porting applications based on the Z-Wave Application Framework (ZAF) from 700 series SDKs to 800 series SDKs.
RAILtest User's Guide	RAIL	User's guide for the RAILtest application (RAILtest), a simple tool for testing the radio and the functionality of the RAIL library.
Serial API Host Appl. Prg. Guide	Serial API	Guideline for developing serial API-based host applications.

Railtest User'S Guide

RAILtest User's Guide

NOTE: This section replaces *UG409: RAILtest User's Guide*. Further updates to this user guide note will be provided here.

The RAILtest application (RAILtest) provides you with a simple tool for testing the radio and the functionality of the RAIL library. For more advanced usage, developers must write software against the RAIL library and create a custom radio configuration.

Proprietary is supported on all EFR32FG devices. For others, check the device's data sheet under Ordering Information > Protocol Stack to see if Proprietary is supported. In Proprietary SDK version 2.7.n, Connect is not supported on EFR32xG22.

Key Points

- Command Line Interface description
- Application features
- Protocol-specific features
- Testing modes

Overview

Overview

The following is a summary of functionality provided as part of the RAILtest application:

- Transmit and receive packets
- Schedule transmits at a specific time in the RAIL timebase
- Configure RAIL address filtering to receive only specific packets
- Enable CCA mechanisms (CSMA/LBT) to validate that a channel is clear before transmit
- Set a timer callback in the RAIL timebase to see how the RAIL timer API works
- Change the transmit channel within the current configuration's band
- Change the transmit power level
- Enable RF energy sensing of a specified duration across the 2.4 GHz and/or Sub 1-GHz bands, and sleep to wake on this event.
- Output a continuous unmodulated tone for debugging
- Output a continuous modulated PN9 stream for debugging
- Enter into direct mode where data can be sent and received using asynchronous GPIOs as input and output

About RAILtest Versions

RAILtest version 2.14 was released with Gecko SDK (GSDK) v4.2. Previous versions are as follows:

- GSDK v4.1: RAILtest v2.13
- GSDK v4.0: RAILtest v2.12
- GSDK v3.2: RAILtest v2.11
- GSDK v3.1: RAILtest v2.10
- GSDK v3.0: RAILtest v2.9
- GSDK v2.7: RAILtest v2.8

Modes

RAILtest can be in a number of different modes. Each of these modes enables a subset of the test application's functionality, so that nonsensical commands (such as trying to transmit a tone and a packet at the same time) are ignored. The following are supported modes, with a short description.

- **None:** The app is doing nothing, but the radio may be on. Parameters can be set in this mode based on the current radio state, and the timer can be used.
- **Stream:** Send a stream of pseudo-random bits.
- **Tone:** Send a tone at the carrier frequency.
- **ContinuousTx:** Send an unending stream of packets, separated by a configurable delay.
- **DirectMode:** Send data to and from a GPIO, without any packet handling.
- **PacketTx:** Send a specific number of packets, with a summary print at the end.
- **ScheduledTx:** Send one packet scheduled in the future.
- **TxAfterRx:** Schedule a packet after each RX after a specific delay.
- **RxOverflow:** Overflow on receive by delaying in RAILCb_RxPacketReceived.
- **TxUnderflow:** Underflow by not loading data for the next TX sequence.
- **TxCancel:** Cancel a single packet transmit to force an error callback.
- **RfSense:** Sense RF energy to wake the radio.
- **PER** (Packet Error Rate) test: A GPIO is toggled, and stats are gathered on received packets.
- **BER** (Bit Error Rate) test: Statistics are gathered on received bits.

Command Line Interface

Overview

The following is a summary of functionality provided as part of the RAILtest application:

- Transmit and receive packets
- Schedule transmits at a specific time in the RAIL timebase
- Configure RAIL address filtering to receive only specific packets
- Enable CCA mechanisms (CSMA/LBT) to validate that a channel is clear before transmit
- Set a timer callback in the RAIL timebase to see how the RAIL timer API works
- Change the transmit channel within the current configuration's band
- Change the transmit power level
- Enable RF energy sensing of a specified duration across the 2.4 GHz and/or Sub 1-GHz bands, and sleep to wake on this event.
- Output a continuous unmodulated tone for debugging
- Output a continuous modulated PN9 stream for debugging
- Enter into direct mode where data can be sent and received using asynchronous GPIOs as input and output

About RAILtest Versions

RAILtest version 2.16 was released with Gecko SDK (GSDK) v4.4. Previous versions are as follows:

- GSDK v4.3: RAILtest v2.15
- GSDK v4.2: RAILtest v2.14
- GSDK v4.1: RAILtest v2.13
- GSDK v4.0: RAILtest v2.12
- GSDK v3.2: RAILtest v2.11
- GSDK v3.1: RAILtest v2.10
- GSDK v3.0: RAILtest v2.9
- GSDK v2.7: RAILtest v2.8

Modes

RAILtest can be in a number of different modes. Each of these modes enables a subset of the test application's functionality, so that nonsensical commands (such as trying to transmit a tone and a packet at the same time) are ignored. The following are supported modes, with a short description.

- **None:** The app is doing nothing, but the radio may be on. Parameters can be set in this mode based on the current radio state, and the timer can be used.
- **Stream:** Send a stream of pseudo-random bits.
- **Tone:** Send a tone at the carrier frequency.
- **ContinuousTx:** Send an unending stream of packets, separated by a configurable delay.
- **DirectMode:** Send data to and from a GPIO, without any packet handling.
- **PacketTx:** Send a specific number of packets, with a summary print at the end.
- **ScheduledTx:** Send one packet scheduled in the future.
- **TxAfterRx:** Schedule a packet after each RX after a specific delay.
- **RxOverflow:** Overflow on receive by delaying in `RAILCb_RxPacketReceived`.
- **TxUnderflow:** Underflow by not loading data for the next TX sequence.
- **TxCancel:** Cancel a single packet transmit to force an error callback.
- **RfSense:** Sense RF energy to wake the radio.
- **PER** (Packet Error Rate) test: A GPIO is toggled, and stats are gathered on received packets.
- **BER** (Bit Error Rate) test: Statistics are gathered on received bits.

Command Line Interface

The most powerful way to interact with the sample application is through the provided command line interface.

Command Input

The syntax for this interface is the standard command [arg0, arg1, ...] syntax, where the number and type of arguments depend on the specific command. Numeric values can be prefixed with 0x to indicate hexadecimal values.

In RAILtest 2.8 (GSDK v2.7.x), the maximum number of arguments to any command is set by the value of `MAX_COMMAND_ARGUMENTS`, and the maximum length of each command line is set by the value of `APP_COMMAND_INTERFACE_BUFFER_SIZE`.

Starting in RAILtest 2.9 (GSDK v3.0), the maximum number of arguments to any command is set by the value of `SL_CLI_MAX_INPUT_ARGUMENTS`, and the maximum length of each command line is set by the value of `SL_CLI_INPUT_BUFFER_SIZE`.

For a full listing of the command options see [Full Help Text](#) or use the `help` command.

Command Responses

All responses to commands are formatted in a human readable yet parsable format. This format has two variations: single and multiline. Both follow these rules.

- Start and end with curly braces { }
- List the command name, enclosed in parentheses ()
- Contain any number of tag/value pairs enclosed in curly braces { }
- Carriage returns and line feeds are treated as whitespace by any parser

Single Response

Used when a command has a single response.

- There is a single start/end curly brace wrapper
- Tag/value pairs are wrapped in a single set of curly braces, separated by a colon {tag:value}.

Example:

```
> getchannel
{{{(getchannel)}}{channel:4}}
```

Multi Response

Used when a command may have multiple responses, such as when reading a block of memory or receiving multiple packets.

- Response starts with a header, delimited by a hash # at the start of the line.
- Header includes the command name, followed by any tags individually wrapped with curly braces { }.
- Following the header, any number of responses can be provided.
- Data lines do not contain the command name or tags, only the values that correspond to the tags in the order described in the header.

Example:


```
> getmemw 0x20000000 4
#{{{(getmemw)}{address}{value}}
{{{0x20000000}{0x0000e530}}
{{{0x20000004}{0x000051c6}}
{{{0x20000008}{0x0000c939}}
{{{0x2000000c}{0x0000e090}}}
```

Peripherals

Peripherals

Buttons

When enabled, RAILtest can use up to two buttons. When pressed for a short duration (less than 1 second), the first button causes RAILtest to transmit one packet. When pressed for a long duration (greater than 1 second), the first button causes RAILtest to continuously transmit packets until the button is pushed again for a long duration. Pressing the second button causes RAILtest to increment the channel number used for TX and RX until the channel number wraps around to its beginning value.

Starting in RAILtest 2.9 (GSDK v3.0), button functionality is disabled by default. To enable button functionality, the `Simple Button Driver` software component needs to be enabled with an instance name of `btn0` for the first button and `btn1` for the second button.

LEDs

When enabled, RAILtest can use up to two LEDs. The first LED toggles when a packet is successfully received, and the second LED toggles when a packet is successfully transmitted.

Starting in RAILtest 2.9 (GSDK v3.0), LED functionality is disabled by default. To enable LED functionality, the `Simple LED Driver` software component needs to be enabled with an instance name of `led0` for the first LED and `led1` for the second LED.

LCD

When enabled, RAILtest can use the LCD to show packets received and transmitted as well as the channel selected and radio activity.

Starting in RAILtest 2.9 (GSDK v3.0), LCD functionality is disabled by default. To enable LCD functionality, the `RAILtest Application`, `Graphics` software component needs to be enabled.

Application Features

Application Features

Packet Mode

The application starts in packet mode with the receiver enabled. In this mode the application receives and transmits packets using the radio's frame controller hardware. To disable receive use the `rx 0` command. To transmit use `tx [numPackets]` or press button PB0 if button support is enabled. To toggle the continuous transmit mode, hold PB0 for a couple of seconds if button support is enabled, or run the `tx 0` command. When transmitting multiple packets or infinite packets there is a configurable delay in between each transmit. By default, this is 250 ms, but it can be set with the `setTxDelay` command.

All received packets are printed to the console with information like CRC, RSSI, and timestamp as long as notifications are enabled. Notifications are enabled by default, but they can be turned off with the `setRxNotification 0` command.

The application by default sends a fixed packet, but it is possible to override the values through `setTxPayload`. The command allows you to modify the values of the payload at specific offsets. For instance, to modify the first 4 bytes sent in the packet to be 0x01 0x02 0x03 0x04, use:

```
setTxPayload 0 0x01 0x02 0x03 0x04
```

To view the currently configured TX Packet information, use `printTxPacket`.

Note: The packet format depends on the current PHY configuration. If your PHY has a dynamic frame length byte then that will be used to determine how much data should be sent.

Direct Mode

In direct mode the radio will still attempt to decode received packets, but it will only be able to transmit packets sent over the DIN pin. The GPIOs for direct mode are fixed for now to the following pins.

```
DOUT - EFR32_PC11 -> EXP_HEADER16/WSTK_P13  
DIN  - EFR32_PC10 -> EXP_HEADER15/WSTK_P12
```

The data on these pins is an asynchronous stream of bits at the expected data rate from the radio (DOUT) or to the radio (DIN). To enter direct mode, issue the `directMode 1` command after starting the app. To leave direct mode use `directMode 0`. If you want to transmit, you must enable the transmitter by issuing `directTx 1` and later stop it with `directTx 0`. Receive is controlled using the standard `rx 1/0` command, but is enabled by default when not transmitting.

Note: Direct mode does not work in certain modulations (for example 4FSK). If you require this mode, contact support to verify your configuration.

Channels/Frequencies

The specific channel configuration depends on the PHY configuration you have chosen for your test app. To switch between channels, use the `setChannel [num]` command. If RX is active (`rx 1`), then any TX or RX in progress will be aborted

and the new channel switched to. In addition, if button support is enabled, you can use button PB1 to cycle through channels.

To modify your frequency to a value not defined in the channel list, you will need to set the application into the `FREQUENCY_OVERRIDE` debug mode via `setDebugMode`, which tells the application to ignore the current channel selection. Once in the `FREQUENCY_OVERRIDE` debug mode, you can use the `freqOverride` command to switch to another center frequency.

Note: The `freqOverride` command requires you to be in `FREQUENCY_OVERRIDE` debug mode. The radio state must also be IDLE for the frequency to be modified -- call `rx 0` first.

```
setDebugMode 1
freqOverride 865000000
```

To leave `FREQUENCY_OVERRIDE` debug mode and return to normal channel-based operation, use `setDebugMode 0`.

Caution: The modem is configured to a specific band. The application will not restrict you from changing the frequency out of band but this could cause significant issues, including forcing a chip reboot.

Command Scripting

The command scripting feature built into RAILtest allows for CLI commands to be executed without back-and-forth interaction on the command line. The same commands that can be executed one by one can be queued up within RAILtest and executed sequentially later. Additionally, a command script can be saved to flash, and any script saved to flash will run automatically on device boot.

In RAILtest 2.8 (GSDK v2.7.x) support is integrated into RAILtest to use a command script in RAM, but to use a command script in flash, the `Flash Data` plugin must be enabled.

In RAILtest 2.9 (GSDK v3.0), to use a command script in RAM, the `CLI Storage in RAM` software component needs to be enabled with an instance name of `inst0`. To use a command script in flash, the `CLI Storage in NVM3` software component needs to be enabled with an instance name of `inst0`.

This is an example of how to enter, print, and run a script from RAM. Note that the `wait` command needs to be used to allow time for the first scripted command (that is, `tx 1`) to successfully run to completion before the second scripted command (that is, `tx 2`) starts.

```
enterScript
tx 1
wait 500000
tx 2
endScript
printScript
runScript
```

This is an example of how to enter, print, run, and clear a script from flash.

```
enterScript 1
tx 1
wait 500000
tx 2
endScript
printScript 1
runScript 1
clearScript 1
```

This is an example of how to enter and run a script from flash on boot.

```
enterScript 1
tx 1
wait 500000
tx 2
endScript
reset
```

Note: For rapid command entry (for example, automated testing), you may need to increase the USART RX buffer from its default value, specified by a define similar to `SL_IOSTREAM_USART_VCOM_RX_BUFFER_SIZE` in RAILtest 2.9, to account for long command string entry being interrupted by chip interrupts. For lengthy (for example hours long) automated test programs, a notable decrease in test time will come from disabling CLI command history, setting `SL_CLLNUM_HISTORY_BYTES` to 0 (RAILtest 2.9).

RAIL Timebase

The microsecond RAIL timebase is used for features requiring specific timing. You can read this timebase with `getTime`, and you can also set this microsecond timebase with `setTime`. Avoid changing the timebase when the timebase is actively being used, such as during clear channel assessment, during scheduled TX or RX, and so on.

Scheduled TX

The RAIL API has its own timebase that can be used to schedule the start of transmits. It measures time as a 32-bit integer in microseconds. As part of this change, many APIs will now return the time they were run, all RX packets report the time they were received, and all TX packets return the time that the transmit was completed. There is also the command `getTime` to print the current time in the RAIL timebase.

To test scheduled transmit we provide two commands: `txAt` and `txAfterRx`. These allow you to send a packet at an absolute time in the RAIL timebase or exactly some number of microseconds after each receive, respectively.

Note: These APIs should provide reliable timing but you may have to adjust for overhead that relates to preamble and PA ramp time in order to transmit at the exact time you want.

Clear Channel Assessment

The RAIL API provides the ability to specify a Pre-Transmit Operation for every transmit, one of which is Scheduled Transmit discussed above. Others are also provided to support two common medium access methodologies that delay transmission until the channel is clear:

- CSMA-CA (Carrier Sense Multiple Access with Collision Avoidance) -- based on IEEE 802.15.4 specification
- LBT (Listen Before Talk) -- based on ETSI EN 300 220-1 specification

Both CSMA-CA and LBT are similar: Before transmission, a device waits for a random 'backoff' period of time and then performs a CCA (Clear Channel Assessment) for a fixed period of time. If the channel is free, the transmit proceeds; if busy, the process is repeated with a new random backoff period. The random backoff period is determined by a randomly chosen multiple of a fixed backoff unit of time.

For CSMA-CA, the random multiplier is a power-of-2 exponential whose range increases on each try up to a limit, and the CCA period is typically short -- smaller than a backoff unit and around the time it takes a small packet to be transmitted. For LBT, the random multiplier is linear and typically chosen to allow the random period to range up to the CCA period, which

can be quite long -- much longer than the on-air time of the actual transmission it's gating. LBT also specifies that if the channel is found busy during CCA, the process cannot repeat until the channel is free.

In RAIL and RAILtest, several parameters are exposed through the `setLbtParams` and `getLbtParams` commands to configure either CSMA-CA or LBT operation. Their interpretation depends on which mode has been chosen:

- `minBo` : Minimum backoff. For CSMA-CA the first try's power-of-2 random exponential range, that is the backoff multiplier, will range from 0 to $2^{\text{minBo}} - 1$. Subsequent tries increase minBo by 1 up to maxBo. For LBT this is the minimum backoff multiplier for all tries, typically 0.
- `maxBo` : Maximum backoff. For CSMA-CA this is the maximum power-of-2 random exponential range to which subsequent tries can increase. For LBT this is the maximum backoff multiplier for all tries, typically $\text{maxBo} * \text{backoff} = \text{duration}$. If both minBo and maxBo are 0, a non-random fixed backoff time is configured.
- `tries` : The maximum number of tries ('busy' CCAs), up to 15, that the CSMA-CA or LBT operation will tolerate before declaring the transmission a failure due to channel busy. A value of 0 will perform no CCA assessments, and always transmit immediately.
- `thresh` : The CCA RSSI threshold, in dBm, above which the channel is considered 'busy'.
- `backoff` : The backoff unit period, in RAIL's microsecond time base. This is multiplied by the random backoff multiplier controlled by minBo and maxBo to determine the overall backoff period. For random backoffs, this value must be in the range 100 to 511 microseconds; for fixed backoffs it can go up to 65535 microseconds.
- `duration` : The CCA duration, in RAIL's microsecond time base. The radio determines the maximum RSSI during this period for comparison against the CCA threshold.
- `timeout` : An overall timeout, in RAIL's microsecond time base, for the operation. If transmission does not start before this timeout expires, the transmission will fail. This is more important for limiting LBT due to LBT's unbounded requirement that if the channel is busy, the next try must wait for the channel to clear. A value of 0 indicates that no timeout is imposed.

Several examples:

- RAILtest's default parameters are suitable for the IEEE 802.15.4 PHY for CSMA-CA, equivalent to: `setLbtParams 3 5 4 -75 320 128 0`. This specifies up to 4 CCA attempts. The 1st will choose a random multiplier of $0..7$ (2^{3-1}), the 2nd $0..15$ (2^{4-1}), the 3rd $0..31$ (2^{5-1}), and the 4th $0..31$ since the maxBo limit is 5.
- ETSI EN 300 220-1 LBT's parameters in the 863 MHz band would be: `setLbtParams 0 10 15 -80 500 5000 1000000`. This specifies a 5 millisecond CCA duration with random backoff period of $0..10 * 0.5$ milliseconds and a 1 second timeout limit in case the channel remains busy.
- A single CCA of 160 microseconds after a fixed 1024 microsecond backoff, against a -70 dBm threshold would be: `setLbtParams 0 0 1 -70 1024 160 0`.

In RAILtest, CSMA-CA and LBT are enabled as a mode applied to subsequent transmits through `setLbtMode`, whose choices are 'off', 'csma', and 'lbt'.

Note: On EFR32, LBT is currently implemented using the EFR32's CSMA-based hardware engine, so LBT parameters are mapped to roughly equivalent CSMA parameters. **Note:** Scheduled Transmit and Clear Channel Assessment are currently mutually exclusive, with Scheduled Transmit taking precedence.

Address Filtering

The address filtering code examines the packet as follows.

Bytes: 0 - 255	0 - 8	0 - 255	0 - 8	Variable
Data0	Field0	Data1	Field1	Data2

In the above structure, anything listed as DataN is an optional section of bytes that will not be processed for address filtering. The FieldN segments reference the specific sections in the packet that will be examined during filtering for addresses. There may be up to four addresses per field, and they may each have a size of up to 8 bytes. To set up address filtering, first configure where the addresses are in the packet and how long they are. Next, configure what combinations of matches in Field0 and Field1 should constitute an address match. Lastly, enter addresses into the tables for each field and enable them.

The `configAddressFilter` command can be used to set the offsets and sizes for the address fields as well as how combinations of matches in Field0 and Field1 are combined to determine whether or not an address matches.

Configuring which combinations of Field0 and Field1 constitute a match is the most complex portion of the address filter. The easiest way to think about this is with a truth table. If you consider each of the four possible address entries in a field, you can have a match on any one of those or a match for none of them. This is shown in the 5x5 truth table below where Field0 matches are the rows and Field1 matches are the columns.

Fields	No Match	Address 0	Address 1	Address 2	Address 3
No Match	bit0	bit1	bit2	bit3	bit4
Address 0	bit5	bit6	bit7	bit8	bit9
Address 1	bit10	bit11	bit12	bit13	bit14
Address 2	bit15	bit16	bit17	bit18	bit19
Address 3	bit20	bit21	bit22	bit23	bit24

Since this is only 25 bits it can be represented in one 32bit integer, where a 1 indicates filter pass and a 0 indicates filter fail. This is the `matchTable` parameter in the `configAddressFilter` command. For a simple one field configuration set the `matchTable` to `0x1fffffe` and for a two field configuration pairing indices across the fields use `0x1041040`.

After you have configured address filtering you must also set which addresses are valid and enable them. This can be done with the `setAddress` and `setAddressEnable` commands. You must also turn on the address filtering feature with the `setAddressFiltering` command.

As an example, to configure an address of `0x00 0x01 0x02 0x03` in field0, index0 for a filter that has one field starting at offset 0 with a length of 4 bytes, use the following commands.

```
configAddressFilter 0x1fffffe 0 4 0 0
setAddress 0 0 0x00 0x01 0x02 0x03
setAddressEnable 0 0 1
setAddressFiltering 1
```

In addition to the standard address filtering, if the frame length decoding algorithm is by frame type, then additional filtering functionality is available. Address filtering can be enabled or disabled based on the frame type that is decoded. This can be set by the `addressFilterByFrame` command. This command takes an 8-bit bitmask, in which each bit represents whether addresses are present in that frame. The least significant bit determines whether to apply the address filter to frame 0, and the most significant bit determines whether to apply the address filter to frame 7. `addressFilterByFrame` must be called after `configAddressFilter` for it to take effect.

Automatic State Transitions

RAIL provides an API to configure state transitions to happen automatically on certain radio events:

- a successful receive
- a failed receive
- a successful transmission
- a failed transmission

All of these events can transition to a radio state of idle or receive. In addition, a transmission can be configured to happen after a successful receive.

These states are configured through the `setTxTransitions` and `setRxTransitions` commands. Each of these commands take in two radio states, which are passed in as single letter strings. To transition to receive after a transmission, regardless of its success, the command is `setTxTransitions r r`.

When settings the receive transitions, there is also a third argument, a bitfield to configure which events count as errors. The documentation for this bitfield can be found in the RAIL library documentation. Calling `setRxTransitions i i 0xFF` will ignore all errors that are possible to ignore, and transition the radio to idle after attempting to receive one packet.

In addition to the state transitions, timings can be set for the transitions. The six timings that are currently configurable are:

- idleToRx
- idleToTx
- rxToTx
- txToRx
- rxSearchTimeout
- txToRxSearchTimeout

Each of these timings configures an amount of delay between two states. The search timeouts allow you to set a maximum time that the chip will sit in the RX search state. These can be used to implement short receive timeouts. The rxSearchTimeout is used when transitioning to receive from idle and the txToRxSearchTimeout is used when transitioning to receive from a past transmit. Setting [rx|txToRx]SearchTimeout to zero disables the timeout and radio will stay in RX until an event changes the state. Transitions to idle simply happen as fast as possible. Each timing is configurable in microseconds, up to a maximum of 13 ms, with the exception of txToRxSearchTimeout which could be up to a quarter of the RAIL timebase, which is $2^{32/4}$ microseconds or 18 minutes. For example, to disable the timeouts and set all other transitions to take 200 μ s, use:

```
setStateTimings 200 200 200 200 0 0
```

Auto-Acknowledgment

RAIL contains auto-acknowledgment (ACK) APIs that are exposed through the RAILtest command interface. To initialize and enable this functionality use `autoAckConfig`. For example, `autoAckConfig rx 100 192 1000` will configure the 'defaultState', the state at which the radio returns after an ACK operation, to receive. Transitions from idle will take 100 μ s. Turnaround transitions will take 192 μ s and will wait for an ACK for 1000 μ s. To load a custom ACK payload, call `setAckPayload` and `setAckLength`.

If a packet is received during the ACK window, then the `isAck` flag for that packet will be true. The ACK window begins at the end of the turnaround time and lasts for the timeout length. If auto acknowledgment is not enabled, then the `isAck` flag will always be false.

To customize ACK functionality, users can use the `autoAckPause` command to either skip the wait for an ACK on the transmit side or skip the transmit of an ACK on the receive side. To modify if/how the ACK is transmitted after a receive operation, use the `setTxAckOptions` command.

Conversely, after a transmit operation, waiting for ACK is controlled by the TX options configured by 1.

While auto-ACK is enabled, you should not call `setTxTransitions` and `setRxTransitions`.

You can disable auto-ACK by calling `autoAckDisable`. It will also reset the Rx and Tx transitions to idle.

RF Energy Sensing

The EFR32 has the ability to sense the presence of RF Energy above -20 dBm within either or both the 2.4 GHz and Sub 1-GHz bands and trigger an event if that energy is continuously present for certain durations of time.

This feature is exposed through library `RAIL_RfSense ()` and `RAIL_RfSensed ()` APIs, whose use is exemplified by the RAILtest commands `rfSense` and `sleep`. Both commands allow you to specify the duration in microseconds of continuous RF energy, and which RF band(s) to monitor. The requested duration will be mapped to the nearest duration supported by the hardware; it is not terribly precise and can be off by a factor of 2 or more. Once RF energy of sufficient duration has been sensed, the sensing operation terminates and a new one must be started if additional sensing is desired.

The `rfSense` command activates sensing in the background during normal RAILtest operation, and will report when energy of sufficient duration has been detected through an asynchronous `rfSensedCheck` message. Note that sensing energy within a band on EFR32 precludes normal packet reception in that same band; this is by design.

The `sleep` command allows you to activate RF sensing in combination with entering one of the Energy-Saving Modes EM0..EM4. To wake from these modes, RF energy of the specified duration and band must be sensed. Energy modes 0

through 3 will also wake and terminate RF sensing on any serial input to the CLI. EM4 does not support waking on serial input. It will only wake on RF sense or a pin reset.

Some examples:

RFSENSE Legacy Mode:

```
> rfSense 500 2 // Sense RF energy of ~0.5ms or longer in sub-GHz band
{{{(rfSense)}}{RfSense:MHz}{RfUs:413}} // Closest HW supports is ~0.4ms
...
{{{(rfSensedCheck)}}{RfSensed:MHz}{RfUs:413}}

> sleep 2 500 1 // Sleep in EM2 for RF energy of ~0.5ms in 2.4 GHz band
{{{(sleep)}}{EM:2}{SerialWakeup:On}{RfSense:GHz}}
...
{{{(sleepWoke)}}{EM:2}{SerialWakeup:No}{RfSensed:Yes}{RfUs:413}}

> sleep 4 500 3 // Sleep in EM4 for RF energy of ~0.5ms in any band
{{{(sleep)}}{EM:4s}{SerialWakeup:Off}{RfSense:Any}}
...
{{{(reset)}}{App:RAILtest}{Built:Jul 19 2019 13:36:24}}
{{{(sleepWoke)}}{EM:4s}{SerialWakeup:No}{RfSensed:Yes}}
```

RFSENSE Selective Mode (Transmit Node Setup):

```
> rx 0
{{{(rx)}}{Rx:Disabled}{Idle:Enabled}{Time:27993006}}

> configRfSenseWakeupPhy
{{{(configRfSenseWakeupPhy)}}{RFSense Wakeup PHY:Enabled}}

> fifoModeTestOptions 1 0 // Manually load Tx FIFO to transmit only preamble and syncword
{{{(fifoModeTestOptions)}}{TxFifoManual:Enabled}{RxFifoManual:Disabled}}

> setRfSenseTxPayload 0x2 0xB16F // <Syncword Size(Bytes)> <Syncword>
{{{(setRfSenseTxPayload)}}{RFSense Payload:Set}}

> tx 1
{{{(tx)}}{PacketTx:Enabled}{None:Disabled}{Time:54463886}}
{{{(appMode)}}{None:Enabled}{PacketTx:Disabled}{Time:57481907}}
{{{(txEnd)}}{txStatus:Complete}{transmitted:1}{lastTxTime:57481861}{timePos:6}{lastTxStart:57398486}{ccaSuccess:0}{failed:0}
{lastTxStatus:0x00000000}{isAck:False}}
```

RFSENSE Selective Mode (Receive Node Setup for EFR32xG22):

```
> sleep 2 2 0xB16F 1 // <EM Mode> <Syncword Size(Bytes)> <Syncword> <RFBand>
{{{(sleep)}}{RfSense:Enabled}{None:Disabled}{Time:14176659}}
...
{{{(sleepWoke)}}{EM:2}{SerialWakeup:No}{RfSensed:Yes}{RfUs:0}}

> rfSense 2 0xB16F 1 // <Syncword Size(Bytes)> <Syncword> <RFBand>
{{{(rfSense)}}{RfSense:Enabled}{None:Disabled}{Time:44817696}}

> {{{(rfSensedCheck)}}{RfSensed:GHz}{RfUs:0}}
```

Multi-Timer

If the multi-timer is never enabled (or subsequently disabled with `enableMultiTimer`), then the RAIL timer is a single-instance, hardware timer. If the multi-timer is enabled, the RAIL timer becomes one instance of the software-based multi-timer. The multi-timer can only be enabled or disabled when no timers are running.

Protocol Specific Features

Protocol-Specific Features

IEEE 802.15.4

RAIL provides IEEE 802.15.4-specific hardware acceleration that can be configured through RAILtest commands. To configure the IEEE 802.15.4 2.4 GHz Radio Configuration, use `config2p4GHz802154`. This will configure the IEEE 802.15.4 2.4 GHz modem settings as well as channel scheme, making channels 11-26 available for use. To configure IEEE 802.15.4 hardware acceleration, use the `enable802154` command. The options exposed in `enable802154` allow you to configure similar parameters as `autoAckConfig`.

Indeed, Auto-ACK is enabled when calling `enable802154`. Thus, do not call `setRxTransitions` or `setTxTransitions` after an `enable802154` while Auto-ACK is enabled. The IEEE 802.15.4 ACK payload is already loaded into the ACK buffer; do not call `setAckPayload` or `setAckLength`. However, you will want to enable the Wait-For-ACK transmit option through `configTxOptions` so the hardware looks for the specific ACK corresponding to the transmitted packet. RAILtest is configured to always set frame pending bit in the ACK if it successfully receives a data command to the node. Any ACK modifications should use commands detailed in section [Auto Acknowledgment](#).

To configure the IEEE 802.15.4 SubGHZ SUN FSK GB868 mode (802.15.4e and 802.15.4g), use 'config863MHz802154' or 'config915MHz802154'. The different options can be configured using `set802154e` and `set802154g` commands. These commands map very closely to the RAIL API that it wraps, so refer to the RAIL Library doxygen for further information.

Further IEEE 802.15.4 configuration is done through `acceptFrames`, `setPromiscuousMode`, and `setPanCoordinator`. To configure the node's address, use the `setPanId802154`, `setShortAddr802154`, or `setLongAddr802154` commands. These commands map very closely to the RAIL API that it wraps, so refer to the RAIL Library doxygen for further information.

From this point, use the normal `tx` and `rx` commands to send packets back and forth.

Transmit side:

```
> rx 0
{{(rx){Rx:Disabled}{Idle:Enabled}{Time:1846852}}
> config2p4GHz802154
{{(config2p4GHz802154){802.15.4:Enabled}}
> enable802154 rx 100 192 1000
{{(enable802154){802.15.4:Enabled}{rxDefaultState:Rx}{txDefaultState:Rx}{idleTiming:100}{turnaroundTime:192}{ackTimeout:1000}}
// Turn on transmit wait-for-ACK option
> configTxOptions 1
{{(configTxOptions){waitForAck:True}{removeCrc:False}{syncWordId:0}{txAntenna:Any}{altPreambleLen:False}{ccaPeakRssi:False}}

// Load packet directed towards the receive side
// Packet has the destination address set on the receive side.
> setTxLength 26
{{(setTxLength){TxLength:26}}
> setTxPayload 0 0x1b 0x61 0x98 0x00 0x34 0x12 0x44 0x33 0x55 0x44
{{(setTxPayload){len:26}{payload: 0x1b 0x61 0x98 0x00 0x34 0x12 0x44 0x33 0x55 0x44 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00 0x00}}
> setTxPayload 10 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09
{{(setTxPayload){len:26}{payload: 0x1b 0x61 0x98 0x00 0x34 0x12 0x44 0x33 0x55 0x44 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x00 0x00 0x00 0x00 0x00 0x00}}
> setTxPayload 20 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f
{{(setTxPayload){len:26}{payload: 0x1b 0x61 0x98 0x00 0x34 0x12 0x44 0x33 0x55 0x44 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e 0x0f}}

// Assumes there is another node that will receive and ACK
> tx 1
> {{(tx){PacketTx:Enabled}{None:Disabled}{Time:157683158}}
{{(appMode){None:Enabled}{PacketTx:Disabled}{Time:157689582}}
{{(txEnd){txStatus:Complete}{transmitted:1}{lastTxTime:157689516}{failed:0}{lastTxStatus:0x0}{isAck:False}}
{{(rxPacket){len:4}{timeUs:157689914}{crc:Pass}{rssi:-20}{lqi:96}{phy:0}{isAck:True}{syncWordId:0}{antenna:0}{channelHopIdx:254}{payload:
0x05 0x02 0x00 0x00}}

// Assumes a node does not ACK
> tx 1
> {{(tx){PacketTx:Enabled}{None:Disabled}{Time:165139148}}
{{(rxAckTimeout){ackTimeoutDuration:1021}}
{{(appMode){None:Enabled}{PacketTx:Disabled}{Time:165187375}}
{{(txEnd){txStatus:Complete}{transmitted:1}{lastTxTime:165145515}{failed:0}{lastTxStatus:0x0}{isAck:False}}
```

Receive side:

```
> rx 0
{{(rx){Rx:Disabled}{Idle:Enabled}{Time:1846852}}
> config2p4GHz802154
{{(config2p4GHz802154){802.15.4:Enabled}}
> enable802154 rx 100 192 1000
{{(enable802154){802.15.4:Enabled}{rxDefaultState:Rx}{txDefaultState:Rx}{idleTiming:100}{turnaroundTime:192}{ackTimeout:1000}}
> setPanId802154 0x1234 // PANID: 0x1234, OTA Value: 0x34 0x12
{{(setPanId802154){802.15.4PanId:Success}}
> setShortAddr802154 0x3344 // Short Addr: 0x3344, OTA Value: 0x44 0x33
{{(setShortAddr802154){802.15.4ShortAddress:Success}}
// Long Addr (OTA): 0xDD 0xCC 0xBB 0xAA 0x99 0x88 0x77 0x66
> setLongAddr802154 0xDD 0xCC 0xBB 0xAA 0x99 0x88 0x77 0x66
{{(setLongAddr802154){802.15.4LongAddress:Success}}
> rx 1
{{(rx){Rx:Enabled}{Idle:Disabled}{Time:2070858}}

// Receive a packet and ACK it
{{(rxPacket){len:26}{timeUs:29662246}{crc:Pass}{rssi:-15}{lqi:82}{phy:0}{isAck:False}{syncWordId:0}{antenna:0}{channelHopIdx:254}{payload:
0x1b 0x61 0x98 0x00 0x34 0x12 0x44 0x33 0x55 0x44 0x00 0x01 0x02 0x03 0x04 0x05 0x06 0x07 0x08 0x09 0x0a 0x0b 0x0c 0x0d 0x0e
0x0f}}
```

In 802.15.4, the first byte(s) written in the Tx buffer correspond to the PHR (physical header) and must follow a precise mapping, including at least the frame length. Formatting the PHR can be laborious and is error prone. Silicon Labs recommends using 'set802154PHR'.

For instance, in SUN FSK, to set this PHR:

```
| bit0 bit15|

| mode switch | reserved | FCS type | data whitening | frame length |

| 0 | 00 | 0 | 1 | 20=000 0001 0100|
```

you can use:

```
> setTxLength 18 //must be >= frameLength+PHRsize-FCSsize
{{{setTxLength}}{TxLength:18}{TxLength Written:18}}
> setTxPayload 0 0x10 0x28
{{{setTxPayload}}{len:18}{payload: 0x10 0x28 0x11 0x22 0x33 0x44 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee 0x10 0x11}}
```

Or

```
> setTxLength 18 //must be >= frameLength+PHRsize-FCSsize
{{{setTxLength}}{TxLength:18}{TxLength Written:18}}
> set802154PHR 1 0 1
{{{set802154PHR}}{PhrSize:2}{PHR:0x2810}}
{{{set802154PHR}}{len:18}{payload: 0x10 0x28 0x11 0x22 0x33 0x44 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee 0x10 0x11}}
```

Note that 'set802154PHR':

- Only formats (and writes in the Tx buffer) the PHR. It does not check that the requested PHR format is coherent with the loaded PHY.
- Uses the TxLength value (set with 'setTxLength') to process the PHR frame length field.
- If the FCS size is known, then it set it to TxLength-PHRsize+FCSsize.
- If the FCS size is not known, then it set it to TxLength-PHRsize (SUN OFDM and SUN OQPSK cases).

Note that the PHR value is interpreted by the HW both in Tx and Rx to dynamically adapt the processing.

Typical SUN OFDM usage:

```
> setTxLength 64
{{{setTxLength}}{TxLength:64}{TxLength Written:64}}
> Set802154phr 2 6 0
{{{set802154phr}}{PhrSize:4}{PHR:0x780c00}}
{{{set802154phr}}{len:64}{payload: 0x00 0x0c 0x78 0x00 0x33 0x44 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee 0x10 0x11 0x12
0x13 0x14 0x15 0x16 0x17 0x18 0x19 0x1a 0x1b 0x1c 0x1d 0x1e 0x1f 0x20 0x21 0x22 0x23 0x24 0x25 0x26 0x27 0x28 0x29 0x2a 0x2b 0x2c
0x2d 0x2e 0x2f 0x30 0x31 0x32 0x33 0x34 0x35 0x36 0x37 0x38 0x39 0x3a 0x3b 0x3c 0x3d 0x3e 0x3f}}
> 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:0x00000000}{isAck:False}}
```

To use 802.15.4 filtering with SUN OFDM PHYs on Rx side

```
> rx 0
{{{(rx)}}{Rx:Disabled}{Idle:Enabled}{Time:3470713331}}
> Enable802154 rx 200 1000 2000
{{{(Enable802154)}}{802.15.4:Enabled}{rxDefaultState:Rx}{txDefaultState:Rx}{idleTiming:200}{turnaroundTime:1000}{ackTimeout:2000}
{defaultFramePending:False}}
> set802154e 3
{{{(set802154e)}}{15.4E_GB868:True}{15.4E_EnhAck:True}{15.4E_ImplicitBroadcast:False}}
> setPanId802154 0xCDAB
{{{(setPanId802154)}}{802.15.4PanId:Success}}
> setLongAddr802154 0x12 0x34 0x56 0x78 0x12 0x34 0x56 0x78
{{{(setLongAddr802154)}}{802.15.4LongAddress:Success}}
```

on Tx side

```
> SetTxPayload 4 0x01 0x2D 0xAB 0xCD 0x12 0x34 0x56 0x78 0x12 0x34 0x56 0x78
{{{(SetTxPayload)}}{len:64}{payload: 0x00 0x0c 0x78 0x00 0x01 0x2d 0xab 0xcd 0x12 0x34 0x56 0x78 0x12 0x34 0x56 0x78 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}}
```

To use continuous Tx (setTxStream) with SUN OFDM PHYs:

- All modes listed in [RAIL_StreamMode_t](#) from 0 to 5 are supported except the 1010 stream option.
- the PHR is interpreted to get the rate, scrambler and frameLength parameters. Thus take care to always format it correctly and load it in the TxFIFO.
- to get a packet of infinit length, PHR frameLength must be set to 0. For that txLength must be set to 4


```
> 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}}
```
- to setTxStream 1


```
{{{(setTxStream)}}{Stream:Enabled}{None:Disabled}{StreamMode:PN9}{Time:2504841818}}
```
- If PHR frameLength is set to a value different from 0, it generates an infinite number of packets of size 'frameLength' with an IFS of 1ms.
- for a stream with data filled with a ramp started at a different offset for successive packets:


```
> setTxstream 1 4
```
- To send a tone at the carrier frequency:


```
> setTxTone 1
```

 it is equivalent to


```
> setTxstream 1 0
```
- For an OFDM PHY the tone can also be generated roughly at channel_bandwidth/6


```
> setTxstream 1 5
```

WiSUN Specific

WiSUN is based on IEEE802.15.4 SUN PHYs but with a specific mode switch protocol.

Two commands support the WiSUN mode switch: `trigModeSwitchTx` and `modeSwitchLife`.

`modeSwitchLife` is optional and intended only for test automation or certification use.

You can also activate two specific Rx events MODE_SWITCH_START and MODE_SWITCH_END. >**Note:** that they need to be enabled if `modeSwitchLife` is set to 1.

For Rx events MODE_SWITCH_START and MODE_SWITCH_END activation:

```
> setEventConfig 0x0 0x0 0x6000 0x6000
{{{(setEventConfig)}}{Mask:0x600000000000}{Values:0x600000000000}}
```

Concurrent Listening Specific

802.15.4 concurrent listening allows one radio switching quickly enough between RF channels to detect 802.15.4 preamble on either RF channel. This allows the application to use two 15.4 channels concurrently. Once preamble is detected, the radio stays on channel to receive the packet. It will also provide an ACK, if configured, and schedule any subsequent TX responses before resuming concurrent listening.

To configure concurrent listening the following RAILtest commands must be issued:

```
// Setup 802.15.4 on the RX side.
> rx 0
{{{(rx)}}{Rx:Disabled}{Idle:Enabled}{Time:1010696}}
> config2p4Ghz802154 1 0
{{{(config2p4Ghz802154)}}{802.15.4:Enabled}}

// Note: Channel must be set to one of the concurrent listening channels before enabling receive.
> setChannel 20
{{{(setChannel)}}{channel:20}}

> enable802154 rx 110 192 672 0
{{{(enable802154)}}{802.15.4:Enabled}{rxDefaultState:Rx}{txDefaultState:Rx}{idleTiming:110}{turnaroundTime:192}{ackTimeout:672}
{defaultFramePending:False}}
> setPanId802154 4660 0
{{{(setPanId802154)}}{802.15.4PanId:Success}}
> setShortAddr802154 13124 0
{{{(setShortAddr802154)}}{802.15.4ShortAddress:Success}}
> setLongAddr802154 221 204 187 170 153 136 119 102 0
{{{(setLongAddr802154)}}{802.15.4LongAddress:Success}}

// Configure 802.15.4 concurrent listening on channels 15 and 20.
> configRxChannelSwitching802154 15 20
{{{(configRxChannelSwitching802154)}}{Success:True}}

// Configure receive options to enable concurrent listening.
> setRxOptions 1024
{{{(setRxOptions)}}{storeCrc:False}{ignoreCrcErrors:False}{enableDualSync:False}{trackAborted:False}{removeAppendedInfo:False}{rxAntenna:Any}
{frameDet:On}{skipDCCal:False}{skipSynthCal:False}{rxChannelSwitching:True}{fastRx2Rx:False}}

// Enable the radio receive mode
> rx 1
```

Once configured, should the need arise to update the one or both concurrent listening channels:

```
> rx 0
{{{(rx)}}{Rx:Disabled}{Idle:Enabled}{Time:1010696}}
// Configure 802.15.4 concurrent listening on channels 11 and 17.
> configRxChannelSwitching802154 11 17
{{{(configRxChannelSwitching802154)}}{Success:True}}
> setChannel 17
{{{(setChannel)}}{channel:17}}
// Enable the radio receive mode
> rx 1
```

Concurrent listening can be disabled either by switching to a non-concurrent listening channel e.g.

```
> rx 0
{{{(rx)}}{Rx:Disabled}{Idle:Enabled}{Time:1010696}}
> setChannel 18
{{{(setChannel)}}{channel:18}}
// Enable the radio receive mode
> rx 1
```

Or, by disabling the rxChannelSwitching RX option, i.e., RAIL_RX_OPTION_CHANNEL_SWITCHING

```
> rx 0
{{{(rx)}}{Rx:Disabled}{Idle:Enabled}{Time:1010696}}
> setRxOptions 0
{{{(setRxOptions)}}storeCrc:False}{ignoreCrcErrors:False}{enableDualSync:False}{trackAborted:False}{removeAppendedInfo:False}{rxAntenna:Any}
{frameDet:On}{skipDCCal:False}{skipSynthCal:False}{rxChannelSwitching:False}{fastRx2Rx:False}}
// Enable the radio receive mode
> rx 1
```

Note: Loading the PHY via e.g., config2p4Ghz802154 , after configuring concurrent listening, will de-initialize concurrent listening and configRxChannelSwitching802154 must be issued again to re-configure the feature.

Bluetooth Low Energy

RAIL provides Bluetooth Low Energy (BLE)-specific hardware acceleration. RAILtest provides a few wrappers over these APIs but since BLE is so timing-critical, not much can be done through RAILtest commands. To enable BLE hardware acceleration, use the setBleMode 1 command. You can then configure channel-specific settings with setBleChannelParams. A preset configuration to enable advertising on channel 37 is available using the setBleAdvertising command.

RAILtest can output a BLE advertising packet on channel 37 with the following commands:

```
> rx 0
rx 0
{{{(rx)}}{Rx:Disabled}{Idle:Enabled}{Time:1999306}}
> setBleMode 1
setBleMode 1
{{{(setBleMode)}}{BLE:Enabled}}
> setBleAdvertising
setBleAdvertising 37
{{{(setBleAdvertising)}}{AdvertisingChannel:37}}
{{{(setBleAdvertising)}}{len:28}{payload: 0x02 0x1a 0xff 0xee 0xdd 0xcc 0xbb 0xaa 0x02 0x01 0x06 0x10 0x08 0x53 0x69 0x6c 0x61 0x62 0x73
0x20 0x52 0x41 0x49 0x4c 0x54 0x45 0x53 0x54}}
> tx 0
tx 0
> {{{(tx)}}{ContinuousTx:Enabled}{None:Disabled}{Time:16160500}}
```

After these commands, use your phone to search for available Bluetooth devices. You should see 'Silabs RAILTEST' as an available device. RAILtest is **not** a connectable device.

Z-Wave

RAIL provides Z-Wave-specific hardware acceleration that can be configured through RAILtest commands. Users can use the setZWaveMode command to enter Z-Wave mode and then setZWaveRegion to specify one of the region-specific PHYs, which generally consist of three separate channels. More about these PHYs can be found in [ITU-T G.9959](https://www.itu.int/ITU-T/g/9959) . To send packets between nodes, the following commands must be entered on both nodes, using the same region.

RX Node:

```
// Turn the radio off first in order to configure it
> rx 0
{{(rx)}{Rx:Disabled}{Idle:Enabled}{Time:99647646}}
// Enable Z-Wave Mode (1) – beam detection enabled (0x2), promiscuous mode off (0x1)
> setzwavemode 1 2
{{(setZWaveMode)}{ZWAVE:Enabled}{Promiscuous:Disabled}{BeamDetect:Enabled}}
// Specify the EU Region
> setzwaveregion 0
{{(setZWaveRegion)}{ZWaveRegion:EU-European Union}{ZWaveRegionIndex:0}}
```

On the transmit side, the user must specify the correct Home ID of the target Z-Wave node. This can be skipped, however, by enabling promiscuous mode on the receiver via `setZWaveMode` and/or `setZWaveOptions`. Additionally, the user must specify an accurate length byte, otherwise the transmitter may encounter an underflow. The CRC will be calculated on the fly by the radio hardware, so there is no need to specify it explicitly.

TX Node:

```
// Specify the Home ID in the packet. This is a 4-byte value, starting at byte 0
> setTxPayload 0 0xCA 0xFE 0xC0 0xDE
{{(setTxPayload)}{len:16}{payload: 0xca 0xfe 0xc0 0xde 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee}}
// Specify the length byte (byte 7) for a packet
> setTxPayload 7 15
{{(setTxPayload)}{len:16}{payload: 0xca 0xfe 0xc0 0xde 0x33 0x44 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee}}
```

On the receiver side, for basic functionality, the node must configure its own Home ID. >**Note:** that this will set parameters for the node on which these commands are executed – they do not set the TX packet contents for outgoing packets. To skip Home ID configuration, promiscuous mode can also be enabled.

RX Node:

```
// The Home ID is CAFECODE, using a “don’t care” hash
setzwavehomeid 0xCAFECODE 0x55
{{(setZWaveHomeId)}{Status:Set}}
```

For more advanced receiver functionality, or to be used as a sniffer, channel hopping must be configured. As used in Z-Wave networks, the three channels in each of the Z-Wave regions can be used in rapid succession for different purposes. For a receiving node to be able to receive these packets on any arbitrary channel, the node uses hardware acceleration to hop through the three different channels at specific intervals, using preamble sense mode. The following specification configures the correct hopping scheme, with the on-channel time and preamble sense mode, as well as zero delay between channels. Note, that while this configuration is accurate for most regions, Japan and Korea regions use 270 μ s for all the channels, instead of the 270, 450, and 560 specified below. After this, packets can be transmitted on channel 0, 1, or 2, and will be received on the receiver node, without having explicitly set the channel.

RX Node:

```
// Configure the channel hopping algorithm
> configRxChannelHopping 0 2 270 0 1 2 450 0 2 2 560 0
{{(configRxChannelHopping)}{numberOfChannels:3}{buffer:0x200048b0}{Success:True}}
// Enable channel hopping
> enableRxChannelHopping 1
{{(enableRxChannelHopping)}{Success:True}}
```

TX Node:


```

setChannel 0
{{{setchannel}}}{channel:0}}
> tx 1
> {{{tx}}}{PacketTx:Enabled}{None:Disabled}{Time:1518591782}}
{{{appMode}}}{None:Enabled}{PacketTx:Disabled}{Time:1518601722}}
{{{txEnd}}}{txStatus:Complete}{transmitted:1}{lastTxTime:1518601679}{timePos:6}{lastTxStart:1518597116}{ccaSuccess:0}{failed:0}
{lastTxStatus:0x000000000}{isAck:False}}
setChannel 1
{{{setchannel}}}{channel:1}}
> tx 1
> {{{tx}}}{PacketTx:Enabled}{None:Disabled}{Time:1522247672}}
{{{appMode}}}{None:Enabled}{PacketTx:Disabled}{Time:1522258337}}
{{{txEnd}}}{txStatus:Complete}{transmitted:1}{lastTxTime:1522258277}{timePos:6}{lastTxStart:1522252961}{ccaSuccess:0}{failed:0}
{lastTxStatus:0x000000000}{isAck:False}}
setChannel 2
{{{setchannel}}}{channel:2}}
> tx 1
> {{{tx}}}{PacketTx:Enabled}{None:Disabled}{Time:1526903409}}
{{{appMode}}}{None:Enabled}{PacketTx:Disabled}{Time:1526930874}}
{{{txEnd}}}{txStatus:Complete}{transmitted:1}{lastTxTime:1526930821}{timePos:6}{lastTxStart:1526908743}{ccaSuccess:0}{failed:0}
{lastTxStatus:0x000000000}{isAck:False}}

```

RX Node:

```

{{{rxPacket}}}{len:13}{timeUs:1508290532}{timePos:4}{crc:Pass}{rssi:-10}{lqi:114}{phy:0}{isAck:False}{syncWordId:0}{antenna:0}{channelHopIdx:0}
{payload: 0xca 0xfe 0xc0 0xde 0x33 0x0a 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb}}
{{{rxPacket}}}{len:14}{timeUs:1511945271}{timePos:4}{crc:Pass}{rssi:-8}{lqi:100}{phy:0}{isAck:False}{syncWordId:0}{antenna:0}{channelHopIdx:1}
{payload: 0xca 0xfe 0xc0 0xde 0x33 0x0a 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc}}
{{{rxPacket}}}{len:14}{timeUs:1516608125}{timePos:4}{crc:Pass}{rssi:-10}{lqi:102}{phy:0}{isAck:False}{syncWordId:0}{antenna:0}{channelHopIdx:2}
{payload: 0xca 0xfe 0xc0 0xde 0x33 0x0a 0x55 0x0f 0x77 0x88 0x99 0xaa 0xbb 0xcc}}

```

Cancel Specific Protocol Mode

To cancel any of the specific BLE, 802.15.4 or Z-Wave mode, you can simply call `deinitProtocol`.

Testing Modes

Testing Modes

Packet Error Rate Testing

RAILtest can be used to determine the packet error rate (PER) for a given setup. To set up this test, a piece of test equipment needs to be configured to send a packet on the rising edge of a GPIO. The specific EFR32 pin and port to be used are identified by the defines `PER_PIN` and `PER_PORT` (RAILtest 2.8) or defines `SL_RAIL_TEST_PER_PIN` and `SL_RAIL_TEST_PER_PORT` (RAILtest 2.9). `perRx 100 10000` configures the Packet Error Rate test to send 100 packets, waiting 10000 μ s between each packet. At the end of the test, the app provides an output indicating that Per mode has finished, and the statistics on the test can be recovered with `perStatus` and `status`. Calling `perRx 0 0` cancels an ongoing test, and calling `perRx` has the same effect as calling `resetCounters`.

Bit Error Rate Testing

The EFR32 hardware can enter bit error rate (BER) receive mode for diagnostic purposes. In BER mode, the chip expects to receive a PN9 ($x^9 + x^5 + 1$) transmission. To run the BER test successfully, a radio configuration specific to BER mode must be generated by the radio configurator.

Use the `setBerConfig` command to specify how many bytes to receive in BER receive mode. The maximum number of bytes is 536870911. Specifying 0 or a number greater than the maximum possible value automatically configures the BER test to receive the maximum number of bytes for testing.

Use the `berRx` command to enter or exit BER receive mode. If the test is allowed to run to completion, there is no need to exit BER receive mode.

Use the `berStatus` command to query for test statistics during or after a test. The statistics are reset when `setBerConfig` is run or when `berRx 1` is run. The statistics include:

- BitsToTest (total bits to be tested)
- BitsTested (the number of bits already received and tested)
- PercentDone (percentage of how many configured bytes have been received)
- RSSI (an instantaneous RSSI value corresponding to the last byte received)
- BitErrors (the number of received bits determined to be in error)
- PercentBitError (percentage of bit errors to bits tested)

Some examples:

```
> setberconfig 100000

> berrx 1

> berstatus
{{{berStatus)}}{BitsToTest:800000}{BitsTested:0}{PercentDone:0.00}
{RSSI:0}{BitErrors:0}{PercentBitError:0.00}}

// PN9 transmission enabled here

> berstatus
{{{berStatus)}}{BitsToTest:800000}{BitsTested:121312}{PercentDone:15.16}
{RSSI:-23}{BitErrors:0}{PercentBitError:0.00}}

> berstatus
{{{berStatus)}}{BitsToTest:800000}{BitsTested:800000}{PercentDone:100.00}
{RSSI:-24}{BitErrors:0}{PercentBitError:0.00}}

> berrx 1

> berstatus
{{{berStatus)}}{BitsToTest:800000}{BitsTested:363936}{PercentDone:45.49}
{RSSI:-23}{BitErrors:0}{PercentBitError:0.00}}

> setberconfig 1000000

> berrx 1

> berstatus
{{{berStatus)}}{BitsToTest:8000000}{BitsTested:888672}{PercentDone:11.11}
{RSSI:-23}{BitErrors:0}{PercentBitError:0.00}}

// PN9 transmission disabled here

> berstatus
{{{berStatus)}}{BitsToTest:8000000}{BitsTested:4418528}{PercentDone:55.23}
{RSSI:-96}{BitErrors:960478}{PercentBitError:21.74}}

> berstatus
{{{berStatus)}}{BitsToTest:8000000}{BitsTested:8000000}{PercentDone:100.00}
{RSSI:-97}{BitErrors:2752908}{PercentBitError:34.41}}
```

Miscellaneous

Miscellaneous

RAM Modem Reconfiguration

RAILtest can dynamically change the modem configuration allowing for a quick evaluation of different protocols or settings without the need to compile or flash the chip. This is enabled through the `writeRmrStructure` , `updateConfigurationPointer` and `reconfigureModem` commands. These commands are intended to be used only by Simplicity Studio and not directly from the CLI.

Register/Memory Access

Note: These commands provide direct access to the chip's address space. Writing a value to an address can change RAM and register values, and even reading from an address can change the operation of the chip. Any action taken with these commands will potentially conflict with the running program, possibly leading to unexpected behavior or a system crash. Only use these commands when instructed to do so by your Silicon Labs support contact.

To modify any memory on the system, use the `getmemw` and `setmemw` commands. To use `getmemw` , specify the starting address and the number of 32bit words you want to read. Using `setmemw` is slightly different in that you specify the starting address and the 32bit words to write starting at the specified address. These commands operate on 32bit words so all addresses **must** be 32bit aligned.

Debug

- To output a tone on the current channel (or overridden frequency) use the `setTxTone` command.
- To output a PN9 or 101010 stream on the current channel on a certain antenna, TX_ANTENNA0 or TX_ANTENNA1, use the `setTxStream` command.

Full Help Text

RAILtest 2.14 (GSDK 4.2) Help Text

Note: Each command is displayed (on the left) with its corresponding arguments and associated datatypes (on the right). Required command arguments and their datatypes are listed first (such as `uint32`) with potential optional arguments and their datatypes listed afterwards (such as `uint32opt`). The list of required command argument datatypes includes `uint8` , `uint16` , `uint32` , `int8` , `int16` , `int32` , and `string` . The list of optional command argument datatypes includes `uint8opt` , `uint16opt` , `uint32opt` , `int8opt` , `int16opt` , `int32opt` , and `stringopt` .

```
> help config2p4Ghz802154

____Application_Configuration____
setEventConfig    Control RAIL events.
                  [uint32] eventsMask<31:0>
                  [uint32] eventsValues<31:0>
                  [uint32opt] eventsMask<63:32>
                  [uint32opt] eventsValues<63:32>
resetCounters     Resets the TX and RX counters.
setPeripheralEnable Control LEDs and LCD peripherals.
                  [uint8] [0=Disable] 1=Enable
setNotifications  Control asynchronous status prints (rxPacket,txEnd,txError).
                  [uint8] 0=Disable [1=Enable]
getLogLevels       Show whether notifications or peripherals are enabled.
getVersion         Get version information.
getVersionVerbose  Get verbose version information.
setPtiProtocol     Set PTI protocol for Network Analyzer.
                  [uint8] 0=Custom 2=Thread 3=BLE 4=Connect 5=Zigbee 6=Z-Wave
getPti             Get PTI configuration.
setPrintingEnable  Control all printing in RAILtest, except CLI.
                  [uint8] 0=Disable [1=Enable]

____Receive_and_Transmit____
rx                Control receive mode.
                  [uint8] 0=Disable [1=Enable]
rxAt              Configure scheduled receive.
                  [uint32] startTimeUs
                  [string] 'rel'=Relative 'abs'=Absolute
                  [uint32] endTimeUs
                  [string] 'rel'=Relative 'abs'=Absolute
                  [uint8opt] rxTransEndSched: [0]/1
                  [uint8opt] hardEnd: [0]/1
setRxOptions       Show/Configure receive options (RAIL_RX_OPTIONS).
                  [uint32opt] rxOptionsValues: bitmask of enabled options
tx                Transmit packets with current TX options.
                  [uint32] number of packets, 0=continuous until next 'tx 0'
txWithOptions      Same tx command. This command is deprecated.
                  [uint32] number of packets, 0=continuous until next 'tx 0'
txAfterRx          Schedule a TX with specified delay after each RX.
                  [uint32] delayUs: 0=Disable
configTxOptions     Show/Configure transmit options (RAIL_TX_OPTIONS).
                  [uint32opt] txOptionsValues: bitmask of enabled options
setFixedLength      Configure fixed length packet operation.
                  [uint16] fixedLength: payload bytes
setPower            Set the transmit power. The radio must be IDLE.
                  [int32] power: deci-dBm unless 'raw' is added
                  [stringopt] 'raw'=units are raw power level
getPower            Get the transmit power in deci-dBm.
setPowerConfig      Set transmit PA configuration. Must use setpower afterwards.
                  [uint8] paMode
                  [uint16] millivolts
                  [uint16] rampTimeUs
getPowerConfig       Get the transmit PA configuration.
getPowerLimits       Get min and max powerLevel for a power mode.
                  [uint8opt] powerMode
enablePaAutoMode     Control automatic PA selection based on the TX power level.
                  [uint8] 0=Disable 1=Enable
configPaAutoMode     Configure entries in the default PA Auto Mode plugin.
                  [uint8] start Index
                  [int32opt] min max mode band: PA auto config entries
sweepTxPower         Sweep TX power for the current PA by toning at each level.
offsetLqi            Adjust the hardware's LQI value for received packets.
                  [int32] offset: signed value to add
                  The resulting LQI is capped within 0..255 range.
getRssi             Get RSSI in dBm. It'll be invalid if receiver isn't ready.
                  [uint32opt] [0=don't wait] 1=wait for valid RSSI if possible
sweepPower           Sweep power by toning low/high in a square wave fashion.
```

Silicon Labs Z-Wave API Reference Guide

Important Changes

The changes described in this file will possibly break the build and/or functionality of an existing application. The description serves the purpose of helping to fix the failing build.

7.24.1

7.24.0

Improve Clear Channel Assessment

Change the clear channel assessment configuration to improve performance.

S2V2 covered commands update according to specification

The following Z-Wave protocol command classes are removed from the NLS coverage list:

- 0x22 ZWAVE_CMD_SET_NWL_MODE
- 0x23 ZWAVE_CMD_EXCLUDE_REQUEST

Restore NVM migration script

Fix the issue about missing NVM migration script in the controller.

Restore NVM handler after backup/restore operations

To avoid NVM access by the stack or the application during backup/restore operations, the NVM3 handler is closed by the SAPI command NVMBackupRestoreOperationOpen. However it was never restored. The NVM3 handler of the stack is restored by the SAPI command NVMBackupRestoreOperationClose. It should no longer be required to reset the controller after a backup operation. The controller must always be restarted after a restore operation. The SAPI command NVM backup close now send an answer with the result of the operation (as specified). In case of error when restoring the NVM3 handler, the controller will answer an error. In this case, the controller should be reset. **Breaking change** `zpal_nvm_backup_close` return a `zpal_status_t` instead of void.

Power Manager new implementation

APIs update

Breaking change

- Changes at the application level
 - Power Manager operations are now divided into three domains: STACK, PAL, and APP.
 - New APIs (`zw_power_manager_(lock/relock/is_active/lock_cancel)`) have been introduced to manage power operations within the APP domain, serving as a wrapper to PAL code. **Important:** we strongly recommend any application code to use this wrapper instead of calling ZPAL APIs.
 - Each power lock is now identified by a unique ID. These lock IDs are defined in the [zw_power_manager_ids.h](#) header file through enums. For the APP domain, `ZPAL_PM_APP_RADIO_APPLICATION_ID` is used for radio locks and `ZPAL_PM_APP_DEEP_SLEEP_APPLICATION_ID` is used for DeepSleep locks. If additional locks are needed at the application level, new IDs must be defined in [zw_power_manager_ids.h](#).
- Changes in the ZPAL `zpal_pm_register` API have been deprecated and `zpal_pm_register_domain` introduced. `zpal_pm_stay_awake` has been deprecated for `zpal_pm_lock` (change in behavior, see [zpal_power_manager.h](#) for more).

details). `zpal_pm_cancel` has been moved to `zpal_pm_lock_cancel` `zpal_pm_relock`, `zpal_get_max_timeout` and `zpal_pm_lock_is_active`, `zpal_pm_lock_type_is_active` API has been introduced

Notable changes:

- **locks are now registered per domain: APP, STACK and PAL**
- possible operations on those power locks are defined as follows:
- a power lock can be acquired initially with a lock operation (`zpal_pm_lock`) with `timeout_ms` being of value 0 (permanent) or non-zero (timer).
 - if already acquired (!= INACTIVE state) a lock operation on that given power lock will fail.
- relock operation does the same as lock but enforces the acquisition of a given lock from any state.
- lock_cancel will cancel the last operation on a given lock resulting in a INACTIVE state.
- `zpal_get_max_timeout` has been introduced for one to check maximum allowed value for `timeout_ms`.
- `zpal_pm_lock_is_active` has been introduced for one to check current state of individual power lock. This represents the current state (ACTIVE_PERMANENT, ACTIVE_TIMER, or INACTIVE) of a lock.
- `zpal_pm_lock_type_is_active` has been introduced for one to check global current state of power lock type: RADIO or DEEP_SLEEP.

Introduce firmware update migration callback for Command Classes

New macro `REGISTER_CC_V6()` has been created and can be used to initialize Command Classes. This macro extends `REGISTER_CC_V5()` in two ways:

- Introduced `migrate()` callback function similar to `init()` or `reset()`. `migrate()` can be used to execute any migration that is necessary for a command class during firmware upgrade. `migrate()` is called by Firmware Update Command Class, after the new firmware is loaded, but before the Firmware Update Status Report is transmitted.
- V6 allows the command class to choose the `cc_api_handler_version`. This allows command classes to update its registration to `REGISTER_CC_V6()` without the need to change the CC handlers. For example:
 - `REGISTER_CC_V5(...)` is equivalent to `REGISTER_CC_V6(CC_API_HANDLER_V3, ..., NULL)`
 - `REGISTER_CC_V4(...)` is equivalent to `REGISTER_CC_V6(CC_API_HANDLER_V2, ..., NULL)`

New ZAF File ID for User Code CC

`ZAF_FILE_ID_CC_USER_CODE_HEADER` has been created to store a new database for the User Code Command Class. The database stores flags about the validity of the corresponding User Code, which makes the NVM initialization significantly faster when many user codes are used. This removes the limitation of `CC_USER_CODE_MAX_IDS`, increasing its range to 255.

Controller's NVM application version

In controller's NVM, application's datas have their own version number (instead of the application version). `ZAF_FILE_ID_APP_VERSION` store the app version and an integer with the data version (like `FILE_ID_ZW_VERSION`). From Z-Wave Alliance <https://github.com/Z-Wave-Alliance/z-wave-stack/issues/441>

Door Lock Keypad default bolt state

The default bolt state of the Door Lock Keypad application was changed to unlocked. Unlocking the door bolt is no longer possible when the device is not included into a network.

CLI change in DoorLockKeypad

For the Door Lock Keypad application, the CLI command `sleeping <enable|disable>` was removed, because the CLI can now be invoked even when the device is in the EM2 sleep state. If the CLI is used from Simplicity Commander, no modifications are required. If the CLI is used from a serial terminal, make sure that the chosen baud rate is 9600 bps.

On Development Kits, if the board was flashed with an application whose baud rate is not 9600 bps, the CLI will fail. To fix this, press the reset button on the development kit. This will reset the WSTK/WPK firmware as well, and the automatic baud rate will be recognized automatically. Remotely, the reset can be triggered by the Simplicity Commander command `commander device reset -s <jlink_serial>`. Alternatively, the baud rate can be set through the development kit's admin port.

The admin port can be accessed via Silink, via the Simplicity Commander over ethernet, or via Simplicity Studio (right click on the board and Launch Console). To set the baud rate, use the command `serial vcom config speed <baud_rate>` .

ZPAL Watchdog

New module initialization function, `zpal_watchdog_init()`, added to zpal WDT module.

- The user needs to call this function before enable the watchdog.

Implement new EU_LR frequency

Shifted EU_LR frequency from 864.4/866.4 Mhz to 864.0/866.0 Mhz to comply with ETSI regulation rules.

Exclusion from foreign network

After an exclusion from a foreign network, the controller sent nops. However, nops were sent on the controller's network instead of the foreign network. So according to the network specification (NWK:01C8.1 & LR-NWK:0078.1), the controller no longer sends nops in case of foreign exclusion.

Logging System

Breaking change

The macros `DPRINT` and `DPRINTF` have been replaced by `ZPAL_LOG` which has two additional parameters: the originating component and the log level severity. Logs can now be enabled selectively per component and messages can be routed to different channels based on the log level (debug, info, warning, error). To configure log messages, use the **Configure** button of the **Z-Wave Log** component in Simplicity Studio or edit `zw_log_config.h` directly in the project's `config` directory.

Removal of main.c from Sample Applications

`main.c` was removed from all sample applications and is now generated automatically by the **System Setup (sl_main)** component during project generation. For more information, refer to the documentation for **System Initialization and Action Processing**.

User Credential Command Class

Breaking change

The specification for the **User Credential Association Set** and **User Credential Association Report** commands was updated in the 2025A certification program: the option to move a credential to a different slot was removed.

The end device implementation was aligned to the new specification. As a result, the API for `CC_UserCredential_move_credential_and_report` changed.

Moving a credential to a different slot is still possible via the CLI; however, the lifeline group can no longer be notified about a credential slot change.

7.23.0

S2V2 Alpha

A first implementation of the Security 2 Version 2 specification. Alpha release implements the following NLS commands:

- KEX Report NLS Support Bit
- NLS State Get
- NLS State Set
- NLS State Report and the following SerialAPI commands:
- Transfer Protocol Command Class (0x69)
- Enable Node NLS (0x6A)
- Get Node NLS State (0x6B)
- Request Protocol Command Class Encryption (0x6C)
- Controller Node Send Protocol Data (0xAC) Alpha release will not support:

- virtual nodes
- supervision encapsulation of protocol commands
- NLS commands `NLS Node List Get` and `NLS Node List Report`

Migration guide

See the new [migration guide](#) on how to migrate projects to the latest SDK.

Expansion board removal

The expansion board was removed, so the functionalities are remapped to the **Wireless Starter Kit Mainboard/*Wireless Pro Kit Mainboard***. Because of the reduced number of LEDs and buttons, some functionalities are only available through the CLI. For more information, please refer to the README file of each example application.

Change zwave_soc_sensor_pir notification sending

SensorPIR's behavior has been changed slightly. Now, if the button is pressed for medium time (check readme for exact duration), then the device will send a `NOTIFICATION_EVENT_HOME_SECURITY_MOTION_DETECTION_UNKNOWN_LOCATION` notification and start a timer with a 10 second timeout. If the timeout event occurs or the user sends the `motion_detected deactivate` command through the CLI, then the device will send a `NOTIFICATION_EVENT_HOME_SECURITY_NO_EVENT` notification.

Enable all examples for all boards

All examples are now enabled for all boards. Previously, some example applications required an RGB LED or multiple buttons to access all of the features. Now, the CLI is enabled by default in all example applications, so every feature can be controlled using the CLI. The functionality of each application is described in the README file in each application's respective folder, with the available commands. This section contains a summary of the changes. For more detailed information, please refer to the README file of each example application.

zwave_soc_door_lock_keypad

Previously, this example required four buttons to control all the features. Now, the user code is no longer simulated, the user can enter/modify the user code manually. The door handle is also accessible through the CLI.

The default PIN code for this application was changed from 1234 to 3494 to comply with the rules recommended by the specification for the User Credential Command class.

zwave_soc_led_bulb

Previously, this example required an RGB LED to control the color of the LED. Now, the color can be read through the CLI. If the RGB LED is not available, then a monochrome LED is used to represent the overall brightness of the set color, which will be calculated from the RGB color with the following formula:

```
uint16_t sum_colors = ((rgb_led_attributes.color.red * rgb_led_attributes.intensity) / multilevel_switch_max)
                    + ((rgb_led_attributes.color.green * rgb_led_attributes.intensity) / multilevel_switch_max)
                    + ((rgb_led_attributes.color.blue * rgb_led_attributes.intensity) / multilevel_switch_max);
// The PWM duty cycle is a percentage in the range of 0-100
uint16_t monochrome_value = (100 * sum_colors) / (3 * 255);
```

zwave_soc_power_strip

Previously, this example displayed the Multilevel Switch value on the RGB LED. Now, if an RGB LED is available, the brightness of that LED is controlled by the Multilevel Switch value. Otherwise, a monochrome LED is used to display this value.

Reduce the number of demo applications

The number of the demo applications has been reduced. From now on, two different variants are supported for each SoC and SerialAPI demo application:

- Default: The default firmware with the default region, which is the EU region.
- V255: The default firmware with 255.0.0 application version. This demo is available to test the OTA and OTW firmware update.

For Zniffer applications, only the default firmware is available.

The default region for this firmware is EU. This region is only applied by the Zniffer PTI and the SoC applications if the dedicated MFG_ZWAVE_COUNTRY_FREQ token is empty (0xFF) when the application first starts up. If another region is needed, then the MFG_ZWAVE_COUNTRY_FREQ token must be set with Simplicity Commander using the following command: `commander flash --tokengroup znet --token MFG_ZWAVE_COUNTRY_FREQ:{region_hex}`, where the {region_hex} is the desired region. The available regions can be found in the `zpa_radio.h` file, such as REGION_EU.

The Long-Range capability can be enabled for any of the demo applications by setting the region to US_LR or EU_LR.

The region of the Serial API and NCP Zniffer applications can be set by the host application.

RF Configuration in Apps

The range for APP_MAX_TX_POWER_LR & APP_MAX_TX_POWER values is now set at their highest (200 deci-dBm) by default for all chips.

Removing support of deprecated SAPI commands

Remove handling of SAPI commands `Replication Send Data (0x44)` and `Replication Command Complete (0x45)`

New serial API commands

Add new serial API commands: Z-Wave API Setup Get Supported Regions Sub Command (0x15) & Z-Wave API Setup Regions Info (0x16).

Non-Listening and Frequently Listening Apps behavior using CLI

The behavior is changed on Non-Listening (Sensor PIR, Multilevel Sensor) and Frequently Listening (Door Lock Keypad) applications in case of installed CLI component. After pressing the reset button (or use `commander reset`), the device will wake up for 10 seconds. This allows the user to interact with the device and disable sleeping. The wakeup time can be configurable through the `zw_cli_sleeping` component.

Removal of postbuild.sh

The postbuild.sh script has been removed from the SDK. This script was used to sign the applications with the Z-Wave SDK's sample keys and generate GBL files. The script is no longer needed as the signing is now done by the SLC's postbuild profile (SLPB). The keys can be set in the projects' YAML file or interactively in the Studio's GUI. For key generation check the UG489 User Guide.

CC Configurator moved

The CC Configurator has been moved from the `ZAF/tool/cc_configurator`. The new location is `DevTools/cc_configurator`.

Multilevel sensor

Multilevel sensor can now be assigned to an endpoint, also, multiple multilevel sensors can be used in a single end device. The cc_config file has been changed to follow this format. Endpoint is now part of the "ID" of a given sensor. The sensor structure changed to include information about the assigned endpoint.

Workspace usage

The workspace solutions are now signing the bootloader and application binaries with the same key, which is stored by the application post build configuration. The signing keys can be freely set by the developer in the SLPB files. The solution will

generate three gbl files with the same keys, one for the bootloader, one for the application and one for the combined bootloader and application.

Door Lock Keypad with U3C Beta Application

The Door Lock Keypad with U3C Beta application has been removed and its functionalities has been merged into Door Lock Keypad application. For certain devices, the Door Lock Keypad application now supports and uses the User Credential Command Class. See application README file for details.

7.22.0

RF Configuration in Apps

prior to this release, RF configuration was stored in two different files, depending on whether or not the chip supported the 20dBm TX level. These files have now been merged. The range for APP_MAX_TX_POWER_LR & APP_MAX_TX_POWER values now has a high value of 200 for all chips. It needs to be increased to allow 20dBm transmission on supported chips.

LR Channel configuration management refactored

Prior to the implementation of the Europe Long Range frequency, management of LR channel configuration has been refactored (US_LR products are impacted). There should be no behavior change.

Europe Long Range frequency - Alpha

Implementation of the Europe long range frequency in Alpha state.

Version Management

ZAF_BUILD_NO, SDK_VERSION_[MAJOR|MINOR|PATCH], ZAF_VERSION_[MAJOR|MINOR|PATCH] are not available in Applications anymore, in order to access these values you must include and use the accessor functions defined in "ZAF_version.h".

Region change and Custom LBT Threshold value

Custom values for LBT Threshold level are now cancelled during region switch. It's now necessary to reapply them after the switch.

EResetReason_t removal

The EResetReason_t enum has been removed. You can now use directly zpal_reset_reason_t. Values are aligned with old EResetReason_t.

Z-Wave specific section

Simplicity SDK require that all sections are placed in the linkerscript. If using software components, they are contributed by the Z-Wave software components. For developers not using the software component mechanism, they should contribute the following sections to their linker file.

- Contributed by the zw_core.slcc software component:
 - zw_protocol_cmd_handlers_lr
 - zw_protocol_cmd_handlers
- Contributed by the zw_apputilities.slcc software component:
 - zw_cc_handlers_v3
 - zw_zaf_cc_config
- Contributed by the zaf_event_distributor_soc.slcc software component:
 - zw_zaf_event_distributor_cc_event_handler
- Contributed by the zwave_ncp_serial_api_controller.slcp software component:
 - zw_cmd_handlers

Extended NVM backup and restore

New command is implemented in the serial API: Extended NVM backup and restore. Basically, it is the same command as NVM backup and restore, with the address now stored on 4 bytes instead of the previous 2. For more details on this command, refer to the Serial API specification.

Removal of default debug traces

The default debug traces were removed from stack components. These traces were not enabled by default.

7.21.2

Fix migration process from 7.17 to 7.18+

The migration process of the controller is broken since 7.18.1. Restore this migration process. This process is used when a controller is updated from a version older than 7.18 to 7.18 or newer.

7.21.1

Support

Footprint (heap, stack, ROM) has been optimized to support low-end devices (ZG14).

Fix saving host data in controller

Fixed an issue on 700 series controllers, where the host was no longer able to save data in the controller's NVM. Increased largest NVM object size to match host's limitation (from 204 to 512 bytes currently used by ZPC or ZGW) This is to satisfy the design of the 700 series and allow the host to store data on the controller's NVM.

ZPAL Power Manager

Relocated `zpal_pm_set_device_type` to apps (e.g. sensor is non-listening). Reworked notifications (deep sleep, power down) related to EM4, EM3, EM2 states.

7.21.0

SPAN synchronization

Unsolicited request from end devices will now have `S2_TXOPTION_VERIFY_DELIVERY` enabled by default to allow SPAN synchronization.

Portable Controller Application

The portable controller application has been removed in this version. Last version supporting this application is 7.20.

ZPAL Power Manager

The maximum number of power manager handlers has been reduced from 20 to 17. This number includes both stack-side and application-side handlers.

Application user-specified versioning

A new boolean parameter is introduced into Simplicity Studio. When the user creates a new application, the user can choose whether their application uses the Z-Wave SDK Version, or a specific version he is free to define.

OTW Update of Serial API controller

In version 7.20 of the Z-Wave SDK, The Serial API Controller application's Product ID, used for validation during OTW updates had been changed to include a Generic and Specific Device Type.\ This prevented updating devices running version 7.19 or lower of the Serial API Controller to more recent versions.

To address this incompatibility, the Specific and Generic Device Type bytes of the Product ID, defined in `application_properties_config.h` (used by the struct `slApp_properties` of the bootloader) have been reverted to their previous values of 0.\ This only affects the Product ID; the application still reports the new Device Type values in its Node Information Frame.\ Note that these values can be changed on the fly using the Serial API command `SERIAL_APIAPPL_NODE_INFORMATION` .

The following update paths for this application are not possible using the pre-built binaries:

- 7.19 or lower to 7.20
- 7.20 to 7.21 or higher

However, firmwares that are not affected by this incompatibility can be created manually, as described in the README of the application.

Z-Wave NVM instance configurable from Simplicity Studio

A configurable component (ZPAL Z-Wave NVM instance) has been created to help user to configure the Z-Wave NVM stack instance from Simplicity Studio. This NVM instance is used on series 700 to store ZAF and user application's data. GSDK and Z-Wave stack data are stored in the default NVM instance (which is configurable from the NVM3 Default Config component).\ On other series, all data are stored in the NVM default instance. So this component is available only in projects using series 700.

Improve NVM management on series 700

Add periodic repacks for Z-Wave NVM instance on series 700 in zpal. Avoid to repack this NVM instance on NVM write (which could create latency in wireless communication).\ NVM default instance already has a periodic repack.

Controller's NVM data update

Controller's data in NVM have been updated in order to save the nodeId length (Classic Z-Wave or Long Range Z-Wave). A migration script is required to update data in NVM. This script is enable by installing slc component "Migration from V7.20 (to V7.21)" in the controller project.

7.20.2

zwave_soc_led_bulb Moved to NonCertifiableApps

The `zwave_soc_led_bulb` (LED Bulb) application is no longer certified, so this application is moved from `Apps` to `NonCertifiableApps` folder.

7.20.1

ZAF Retention Register

Added module for retention registers. Application can now use `ZAF_retention_register_read` and `ZAF_retention_register_write` to read and write retention registers not used by ZAF, protocol and PAL. `ZAF_retention_register_count` can be used to get the number of registers available for application use. Register index starts from 0.

7.20.0

ZAF Application Name

Storing the name of applications is now handled by the `zaf_appname` component.\ The name is stored as a string by default. By installing an additional component, `zaf_appname_nvm`, the name can be saved to the non-volatile memory as well.\ When multiple Z-Wave devices are connected, this feature can speed up the process of identifying which application is flashed on each device.

ZAF Job Helper

This module is no longer recommended for use in new applications. The main use case for this module was to transmit different frames for the same event. The new applications should leverage the ZAF Transport Queue instead. For applications that still wants to use this module it has been moved to its own component: `zaf_job_helper.slcc`

ZAF Transport Module

`ZW_TransportEndpoint` was refactored and much local data has been removed. Functions `Transport_SendRequestEP()` and `Transport_SendResponseEP()` were replaced with a common function `ZAF_Transmit()`. The type of `callback_function` in `ZAF_Transmit(buffer, len, txOptions, callback_function)` is now `ZAF_TX_Callback_t`. Note that `ZAF_Transmit()` does not set `S2_TXOPTION_VERIFY_DELIVERY` by default, as it was with `Transport_SendRequestEP()`. This flag can be set in `ZAF_Transmit()`, by enabling it in `TransmitSecurityOptions`. It is not recommended to use this API directly, applications should use the ZAF Transport Queue to ensure that only one frame is sent from the application to the protocol at one time.

ZAF_Transmit migrated to zaf_transport_tx

In order to ensure a single entry point for frames from the application to the protocol, `ZAF_Transmit` is not being called directly by Command Classes.

ZAF Transport Queue

This module implements a transport queue in the application level. This queue ensures that there is only a single entry point for a frame from the application. This adds a new feature to applications, the ability to "send and forget", which means the user only has to send to the queue, and the rest of the process is taken care of. i.e. Send Central Scene Notification and Send Basic Set in sequence. The queue size is configurable and it can bet set by `ZAF_TRANSPORT_CONFIG_QUEUE_SIZE`, the default is 2.

ZAF TX Mutex Removed

This module was used to ensure that a shared buffer frame was only used by one module at the same time. It was very relevant for 500 series to reduce the RAM usage as no stack was available for the tasks, however it became irrelevant for 700 series and forward. This module was only being used by the ZW Transport Multicast module at this point. The buffer was moved to the module itself and the TX Mutex module was removed, therefore the `ZAF_tx_mutex.h` no longer exists.

Merge agi.c/h into CC_AssociationGroupInfo.c/h

The `agi.c/h` module does not exist anymore as the contents of the files have been merged into `CC_AssociationGroupInfo`. This should reduce some complexity since data does not have to be passed between the different modules, and should reduce flash and ram consumption.

ZAF Event Helper module removed

This module was only used by the event distributor soc to enqueue events into the application event queue therefore this functionality was moved into the event distributor soc.

The interface changes are listed below:

Old name		New name
<code>ZAF_EventHelperInit</code>	->	Not needed anymore
<code>ZAF_EventHelperIsInitialized</code>	->	Not needed anymore
<code>ZAF_EventHelperEventEnqueue</code>	->	<code>zaf_event_distributor_enqueue_app_event</code>
<code>ZAF_EventHelperEventEnqueueFromISR</code>	->	<code>zaf_event_distributor_enqueue_app_event_from_isr</code>

The functions that still exist takes the same arguments as before. They are defined in `zaf_event_distributor_soc.h`

Application Name Configurable

Each application's name is configurable. It can be changed by specifying the configuration entry named `ZAF_APP_NAME` in the SLC project description. Printing the application's name, reset reason, and SDK version was moved to `ZAF_PrintAppInfo()`.

Eliminating the config_app.h files

`config_app.h` files were eliminated from sample applications, and the content of the headers were moved to the project source files. Exceptions to this are version numbers, previously stored in these files. Version numbers have been moved to a configurable SLC component. The `APP_VERSION_MAJOR`, `APP_VERSION_MINOR`, and `APP_VERSION_PATCH` numbers are now configurable via the `zw_version` component in the studio GUI. Users still get the original Silicon Labs version numbers by default, but it is also possible to change them as described above.

Common hardware functions

Each application would implement each own `<App>_hw_init` function and have their own `<App>_hw.h` header. The applications that go to deep sleep also had their own `<APP>_hw_deep_sleep_wakeup_handler` function. These two common functions have been consolidated in a `app_hw.h` as `app_hw_init` and `app_hw_deep_sleep_wakeup_handler`. `app_hw_init` MUST be implemented by the application while `app_hw_deep_sleep_wakeup_handler` should only be implemented by applications that can go into deep sleep.

Application's Source Code

Application's source codes were moved from `<application_name>.c` to `app.c` file. The `app.h` headers were also removed with the function `app_init()`. Because of this the function calls were removed from the `main.c` file.

Board Indicator Moved into ZAF

Board indicator LED is initialized in `ZAF_Init()` and the default idle status is set also in it. `Board_IndicateStatus()` is available so it can be used freely in the code.

Command Classes

New Version of Command Classes Handlers

New macro `REGISTER_CC_V5()` has been created and can be used to initialize Command Classes. This macro requires a CC handler that takes only input and output structure as arguments. See `CC_BinarySwitch_handler(cc_handler_input_t * input, cc_handler_output_t * output)` as an example. This makes it possible for CC to exit with status `WORKING`, without a need to pass duration parameter to Supervision CC, or to call Supervision Report directly, as was the case so far.

`REGISTER_CC_V5()` is supported in all Command Classes that support timed change, and their handlers were updated accordingly. List of handlers compatible with `REGISTER_CC_V5()` :

- `CC_BinarySwitch_handler()`
- `CC_ColorSwitch_handler()`
- `CC_DoorLock_handler()`
- `CC_MultilevelSwitch_handler()`
- `CC_MultiChannel_handler()`

Folder structure and private/public headers

Some command classes have more than two source files and header files. This is because the command class contains a lot of logic therefore splitting into more than one source and one header file makes sense for readability. In order to allow interaction between two source files, private header were introduced and to avoid confusion a new folder structure is used. The source files and private header are placed in `src` while the public headers are placed in `inc`.

All Command Classes have been structured to have `inc`, `src` and in some cases, `config` folder. Header files located in `config` folder contain values configurable in Simplicity Studio.

CC Basic

`CC_Basic_Set_handler()` was removed. This function is no longer required by applications as ZAF handles Basic Command Class mapping (assuming all relevant Command Classes implement a Basic Set mapper function).

CC Central Scene

Configuration of Command Class Central Scene was migrated to Z-Wave Command Class Configurator and must be configured in `<app>.cc_config`. More details can be found [here](#).

The configuration of `cc_central_scene_get_supported_key_attributes` was removed from `zw_cc_central_scene` component. Values can either be set using Z-Wave Command Class Configurator, or they are automatically calculated by Command Class.

- Function `cc_central_scene_set_configuration` was removed since this operation is only done internally in the Command Class.
- `cc_central_scene_get_supported_key_attributes` were made static as they are only used internally.
- `cc_central_scene_get_configuration` was removed as obsolete. Slow refresh value is now used only internally in `CC_CentralScene.c` and can be accessed directly.
- Refactored function `CommandClassCentralSceneNotificationTransmit()` :
 - Renamed to `cc_central_scene_notification_tx()`
 - Removed `sourceEndpoint` argument. The function will always use `ROOT` as source endpoint.

CC Door Lock

It is now the responsibility of the Command Class to ensure that the TSE actions are managed. All TSE-related functionalities have been moved from the applications to the Door Lock CC. This includes functions `cc_door_lock_handle_working_state()` and `CC_DoorLock_operation_report_notifyWorking()`

Hardware handling has been moved out of CC Door Lock, leaving the command class with the software logic only.

Supervision Handling Moved to CC

Handling of Supervision Get for Door Lock Operation Set has been moved from application to Command Class.

Function `cc_supervision_get_received_handler` no longer needs to be implemented in the app.

New `REGISTER_CC_V5` made it possible to remove `CmdClassSupervisionReportSend()` with status Working and leave it to CC Supervision to handle the working case.

CC Multi Channel

Configuration of Command Class Multi Channel (endpoints) was migrated to Z-Wave Command Class Configurator and must be configured in `<app>.cc_config`. More details can be found [here](#).

Configuring whether endpoints are identical or not must be done in Studio.

As a result of the migration, `Transport_AddEndpointSupport()` was removed.

To comply with the naming scheme, the following two SLC components were renamed:

- `zw_cc_multichannelcontrol` -> `zw_cc_multi_channel_control`
- `zw_cc_multichannelsupport` -> `zw_cc_multi_channel_support`

CC Notification

The source code in `notification.c` has been moved to `CC_Notification.c` and `notification.c` has been removed.

Notification types have been moved into `cc_notification_types.h` and states/events have been moved into `cc_notification_events.h`. This is done to prepare for automatic generation of these two files based on the types and states/events defined in the Z-Wave specification.

CC_NOTIFICATION_MAX_EVENTS_PER_NOTIFICATION was removed as the number of configured events/states can be derived from the CC configuration.

The CC configuration syntax was simplified to no longer require `state_event:` for each state/event listed under `state_events:`. This is a breaking change and existing `cc_config` files must be updated. More details can be found [here](#).

Existing syntax:

```
state_events:
- state_event: NOTIFICATION_EVENT_POWER_MANAGEMENT_OVERLOADED_DETECTED
- state_event: NOTIFICATION_EVENT_POWER_MANAGEMENT_LOAD_ERROR
```

New syntax:

```
state_events:
- NOTIFICATION_EVENT_POWER_MANAGEMENT_OVERLOADED_DETECTED
- NOTIFICATION_EVENT_POWER_MANAGEMENT_LOAD_ERROR
```

CC Wake Up

Applications that included the Command Class Wake Up had to call three functions in response to the `EVENT_SYSTEM_LEARNMODE_FINISHED` event. These functions were `CC_WakeUp_stayAwake10s`, `CC_WakeUp_AutoStayAwakeAfterInclusion` and `CC_WakeUp_startWakeUpNotificationTimer` if the inclusion state is not excluded. Now this is handled by ZAF. This is done by implementing the weak function `zaf_learn_mode_finished`.

Application Events Abstracted

Some sample applications contained events that contained hardware-specific names. Those names were based on how the Silicon Labs port would generate those events.

Many events had the `BUTTON` keyword and it was removed from event names so `EVENT_APP_BUTTON_<event>` becomes `EVENT_APP_<event>`. Below is the list of renamed events for each application that did **NOT** follow this pattern.

zwave_soc_switch_on_off

Old name		New name
<code>EVENT_APP_BUTTON_TOGGLE_SHORT_PRESS</code>	->	<code>EVENT_APP_TOGGLE_LED</code>
<code>EVENT_APP_BUTTON_TOGGLE_HOLD</code>	->	<code>EVENT_APP_SEND_NIF</code>

zwave_soc_key_fob

Old name		New name
<code>EVENT_APP_BUTTON_UP_ASSOCIATION_GROUP_ADD</code>	->	<code>EVENT_APP_ASSOCIATION_GROUP_ADD_START</code>
<code>EVENT_APP_BUTTON_UP_ASSOCIATION_GROUP_REMOVE</code>	->	<code>EVENT_APP_ASSOCIATION_GROUP_REMOVE_START</code>

New ZAF Responsibilities

In order to make apps smaller and easier to start with, all implementation common to all apps that did not require additional configuration from the user has been moved to ZAF.

Features moved out of apps to ZAF

All sample application would keep a local pointer to the `AppHandles` passed by the `ApplicationTask`. This parameter is already passed to ZAF in the `ZAF_Init` therefore the Applications don't need to keep a local pointer anymore. They should use the functions from `ZAF_Common_interface.h` to retrieve the data needed from the `AppHandles`.

They also had to call `zaf_event_distributor_init` in the `ApplicationTask`. This is not necessary anymore as this functions is called by ZAF in the `ZAF_Init`

The sample applications that support the CC Firmware Update also had to call `cc_firmware_update_send_status_report` in `ApplicationTask` after `ZAF_Init`. This is not needed anymore because the CC Firmware Update handles it internally.

New weak function on EVENT_SYSTEM_LEARNMODE_FINISHED

A new weak function was introduced in ZAF, `zaf_learn_mode_finished`. This function can be used by Command Classes that need to execute an action in response to this event. Currently only the Command Class Wake Up utilizes this functionality.

7.19.1

Command Classes

CC Configuration

The YAML configuration of CC Configuration in `<app>.cc_config` now requires "number" and "file_id" for each parameter. This requirement was added to allow addition/removal of parameters across different application firmware versions. See `MultilevelSensor.cc_config` for an example.

The output array generated in `cc_configuration_config.c` is now sorted by "number". The Z-Wave Command Class Configurator (Z3C) will automatically sort the items, but in case a previously auto-generated file is manually modified, make sure that the array stays sorted by "number". The sorting is not required as of this release, but will be required later as it opens for some flash usage optimizations in `CC_Configuration.c`.

CC Central Scene

`cc_central_scene_properties` was renamed to `cc_central_scene_identical`. If you used this variable in your code, please change any occurrences of it to the new name.

Important Changes in 7.19.0

In general, these changes are designed to allow the user to configure their projects in the Simplicity Studio Component Editor, rather than by updating code or header files. A new Z-Wave Command Class Configurator has been added to Simplicity Studio to accommodate the more advanced Z-Wave command classes that require complex configurations. More details can be found [here](#).

In order to make applications smaller and easier to start with, all implementation that was common to all applications and that did not require additional configuration from the user has been moved to the Z-Wave Application Framework (ZAF).

Configurable Parameters Have Been Moved from config_app.h

A number of parameters, previously located in `config_app.h`, that were mandatory for application configuration now can be configured using the Simplicity Studio GUI and are moved out of this header file. This makes it easier to set up all mandatory configurations at the moment when the new project is created in Studio.

See [Command Classes](#) for more details about specific command class configuration.

Role Type and Device Type

`APP_ROLE_TYPE` was moved to `zaf_config_role_type.h`. `DEVICE_OPTION_MASK` was renamed to `ZAF_CONFIG_DEVICE_OPTION_MASK` and it was also moved to `zaf_config_role_type.h`. Both defines are automatically set by `role_type` components for end devices.

Generic and Specific type can be configured in the GUI:

- `GENERIC_TYPE` was renamed to `ZAF_CONFIG_GENERIC_TYPE`
- `SPECIFIC_TYPE` was renamed to `ZAF_CONFIG_SPECIFIC_TYPE`

They are moved to `zaf_config.h` and can be configured in the "Z-Wave ZAF Component" component.

Requested Security Keys

The configuration of security keys to request from the controller, previously defined as `REQUESTED_SECURITY_KEYS`, was moved from `config_app.h` to `zaf_config.h`. Instead of using `REQUESTED_SECURITY_KEYS`, it is now possible to select any combination of keys in "Z-Wave ZAF Component":

- `ZAF_CONFIG_REQUEST_KEY_S0`
- `ZAF_CONFIG_REQUEST_KEY_S2_UNAUTHENTICATED`
- `ZAF_CONFIG_REQUEST_KEY_S2_AUTHENTICATED`
- `ZAF_CONFIG_REQUEST_KEY_S2_ACCESS`

Number of End Points

The configuration of number of end points, previously defined as `NUMBER_OF_ENDPOINTS`, was renamed to `ZAF_CONFIG_NUMBER_OF_END_POINTS` and moved to `zaf_config.h`.

Application Initialization

`ZAF_setApplicationData()` was removed. The parameters of this function must instead be passed to `ZAF_Init()`.

For Serial API specifically, `SetTaskHandle()` (and `GetTaskHandle()`) was introduced and must be used instead of `ZAF_setApplicationData()`. `ZAF_setAppProtocolConfig()` must be used instead of `ZAF_Init()` to set the protocol configuration.

`ZAF_setAppHandle()` was moved to `ZAF_Init()` and no longer has to be invoked from the application.

ZAF_ApplicationEvents.h

Application events like `EAPPLICATIONEVENT_APP` were moved to `ZAF_ApplicationEvents.h` as they are all the same for most apps. Additional events can be added to the file if desired.

ZAF_Init()

The parameters of `ZAF_Init()` have changed. See the description of `ZAF_Init()` for information about which parameters it takes.

`ZAF_Init()` MUST be invoked from the task function passed to `ZW_ApplicationRegisterTask()`. For the Z-Wave apps included in the SDK, this function is named `ApplicationTask()`.

AssociationInitEndpointSupport()

In a previous release, the need for `AssociationInitEndpointSupport()` was removed as ZAF takes care of the association root group mapping. The function was kept for backwards capability, but is now removed.

Initialization of Command Classes

Command Classes manage their own initialization and so the application no longer needs to do it. Each Command Class uses `REGISTER_CC_V4()` to register its init and reset functions. This ensures that the initialization and reset are performed by ZAF.

Read/Write to NVM

The functions used to read/write from Non-Volatile Memory `zpal_nvm_read(handle, key, object, object_size)` and `zpal_nvm_write(handle, key, object, object_size)` are replaced by `ZAF_nvm_app_read(key, object, object_size)` and `ZAF_nvm_app_write(key, object, object_size)`. The new functions work without needing a pointer to the file system as an input. The application file system must be opened by calling `ZAF_nvm_app_init()` at initialization so it is available to be used by `ZAF_nvm_app_read()` and `ZAF_nvm_app_write()`.

ZAF filesystem can be used with `ZAF_nvm` module. `ZAF_FILE_ID_BASE` was removed. Now ZAF file IDs start from 0.

Removed LED_RGB and Timer Components

Timer components were used to control `RGB_LED`, but these components are replaced by the Silicon Labs component "SL_SIMPLE_RGB_PWM_LED". After this release these components are no longer available.

The following functions were removed from `platform/SiliconLabs/AppsHW/inc/board.h`:

- `Board_RgbLedInitPwmTimer()`
- `Board_RgbLedEnablePwm()`
- `Board_RgbLedDisablePwm()`
- `Board_RgbLedSetPwm()`

New ZAF Responsibilities

In order to make applications smaller and easier to start with, all implementation that was common to all applications and that did not require additional configuration from the user has been moved to ZAF.

Features Moved Out of Apps to ZAF

Learn Mode

ZAF is now responsible for the handling of Learn Mode.

Device Reset Local

ZAF is also responsible for handling the device reset using the "NVM SOC" component.

Removed State Machine from Applications

All sample applications had a state machine implemented into them. It is no longer needed, making the applications simpler and allowing more responsibility to be given to ZAF. Applications can still implement a state machine if the use case requires but our sample applications use cases do not require state machines.

Stay Awake

An application no longer has to pass a stay awake function to ZAF as the stay awake functionality is now handled by ZAF. The callback parameter was removed from `ZAF_Init()` and from `Transport_OnApplicationInitSW()`.

Also, `EZWAVERECEIVETYPE_NODE_UPDATE` and `EZWAVERECEIVETYPE_STAY_AWAKE` need not trigger a call to a stay awake function as this is now handled inside ZAF.

ZAF will determine the stay awake time from the role type and from whether CC Wake Up is linked or not.

`ZAF_FLIrS_StayAwake()` and `CC_WakeUp_stayAwakeIfActive()` must no longer be used and are replaced by `zaf_stay_away()`.

Event Distributor

All sample applications implemented their own event distributor component. However, it was the same between all sample applications, with the only difference being if the application allowed for pooling or full event driven. This logic has been moved into ZAF in the component event distributor and the logic has not changed.

Two types of distributor have been implemented: NCP and SOC

NCP

This is the event distributor used in Serial API and it is the event distributor that should be used in all applications that run the host in a separated chip.

SOC

This is the event distributor used in all other sample applications and it should be used in all applications that run controller and host on the same chip.

NVM SOC

All SOC sample applications use this component as it contains all the NVM logic expected to be common among them. It is responsible for resetting the APP NVM, Load Default Configuration and Setting Default Configuration. It contains handlers that can be used for custom behaviors in the application level.

AppPowerDownCallBack

This function replaces the ZAF Power Module Wrapper for more optimized code. Instead of registering a callback which was called as the last step just before the chip entered deep sleep hibernate, a new weakly-defined function `ZW_AppPowerDownCallBack` is provided.

ZAF Power Module Wrapper - Deprecated

The ZAF Power Module Wrapper is no longer supported. It was responsible for registering functions that would be called as the last step just before the chip enters deep sleep hibernate.

It is important to highlight that the protocol still supports the `EZWAVECOMMANDTYPE_PM_SET_POWERDOWN_CALLBACK` for registering such callbacks at run time. However, we recommend that the weakly-defined function `ZW_AppPowerDownCallBack` is used instead. Since the DeepSleepTimer is no longer using the protocol command, we reduced the `MAX_POWERDOWN_CALLBACKS` from 4 to 3.

Components Renamed

The following component IDs were renamed in order to comply with the snake case naming convention

- `zw_cc_centralscene` renamed to `zw_cc_central_scene`
- `zw_cc_colorswitch` renamed to `zw_cc_color_switch`
- `zw_cc_doorlock` renamed to `zw_cc_door_lock`
- `zw_cc_multilevelsensor` renamed to `zw_cc_multilevel_sensor`
- `zw_cc_multilevelswitchsupport` renamed to `zw_cc_multilevel_switch_support`

Command Classes

Z-Wave Command Class Configurator

An advanced configuration was introduced in order to accommodate the more advanced z-wave command classes that require complex configurations. More details can be found [here](#)

Command Class List Generator

An application no longer needs to pass the Command Classes lists to the protocol and ZAF. ZAF has new components that are capable of generating these lists from the linked command classes. For end points a new macro was introduced:

```
#define ZAF_CC_REGISTER_CONFIG(cc, config, index)
```

This macro needs to be called for each element in a configuration array that contains endpoints. For example:

```
static cc_multilevel_switch_t s_switches[] = {
{
    endpoint = 2,
},
{
    endpoint = 1,
},
};

...

ZAF_CC_REGISTER_CONFIG(COMMAND_CLASS_SWITCH_MULTILEVEL, &s_switches[0], 0);
ZAF_CC_REGISTER_CONFIG(COMMAND_CLASS_SWITCH_MULTILEVEL, &s_switches[1], 1);
```

This is automatically handled in the command classes provided in the SDK but command classes not from the SDK need to follow this standard. This requirement is similar to the `REGISTER_CC`.

I/O Interface for CCs

Some Command Classes have a new level of abstraction, the I/O interface. This interface has the functions used to read, write, and sometimes migrate the data. This level of abstraction allows the user to disable and enable data persistency seamlessly.

CCs that support this interface:

- Battery
- Central Scene
- Color Switch
- Configuration
- Door Lock
- Multilevel Switch
- Notification
- User Code

CC Association

The `AssociationInit()` function was removed and the initialization performed by this function was split into two separate functions: `init()` and `reset()`. These two functions are now passed as callbacks to the `REGISTER_CC_V4` macro. This ensures that the initialization and reset are performed by ZAF, instead of the user code.

`MAX_ASSOCIATION_GROUPS` and `MAX_ASSOCIATION_IN_GROUP` have been moved from `config_app.h` to `cc_association_config.h` and renamed to `CC_ASSOCIATION_MAX_GROUPS_PER_ENDPOINT` and `CC_ASSOCIATION_MAX_NODES_IN_GROUP` respectively. These values can be set in Simplicity Studio in the "Association Command Class" module.

CC AGI

The Association Group Information Command Class has been removed from the `config_app.h` header and implemented in the config generator and the application slcp files. This refactoring enables the user to easily configure the device's group and node information in the Simplicity Studio GUI instead of a source file. All the command classes, utilities, applications, and tests have been refactored to use the automatically-generated headers containing the AGI group and node configuration. The names of the defines have also been refactored to make them easy to understand.

The `AGLResourceGroupSetup` function has been deprecated. In addition, the association group information has been moved from the `config_app.h` header to the `cc_config` yaml file. Redundant duplication of the root device groups is removed. In order to define the device groups and endpoints, developers have to add / modify the `zw_cc_association` section in the respective app `cc_config` file. Developers can look at at WallController, SensorPIR, or PowerStrip applications as a starting point for defining their own groups and endpoints. The configurator will use this information to generate the agi groups and tables in the `cc_association_config.c` source file for the specific application.

CC Basic

Basic Get mapping is now done automatically by ZAF. It does require relevant command classes to implement a certain function. See [How to implement a new command class](#) for more information.

True Status was removed from CC Basic as True Status must be handled by the command class mapped to Basic Set.

As a result of the above changes, the following functions were removed from CC Basic:

1. CC_Basic_report_stx()
2. CC_Basic_prepare_zaf_tse_data()
3. CC_Basic_report_notifyWorking()
4. CC_Basic_GetCurrentValue_handler()
5. CC_Basic_GetTargetValue_handler()
6. CC_Basic_GetDuration_handler()

Additionally, a Supervision flag was added to [CC_Basic_Set_tx\(\)](#) making it possible for the application to decide whether Supervision must be used or not for the given transmission of Basic Set. The use of Supervision was previously hardcoded in CC_BasicController.c.

CC Binary Switch

The Binary Switch command class was rewritten to use ZAF Actuator and take care of reporting the true status so that the application does not have to.

The configuration of this CC must be done in the `*.cc_config` file. Example for root device:

```
zw_cc_binaryswitch:  
  binary_switches:  
    - endpoint: 0
```

See `PowerStrip.cc_config` for an example of endpoints.

CC Central Scene

Three Central Scene config APIs were moved from the application to the Command Class:

- `cc_central_scene_set_configuration` (used to be `setAppCentralSceneConfiguration`)
- `cc_central_scene_get_configuration` (used to be `getAppCentralSceneConfiguration`)
- `cc_central_scene_get_supported_key_attributes` (used to be `getAppCentralSceneReportData`)

The configuration of `cc_central_scene_get_supported_key_attributes` can be done from the Simplicity Studio GUI in the "zw_cc_central_scene" component.

Non-Volatile Memory can be used to store the configuration values by installing the "zw_cc_central_scene_nvm" component.

CC Color Switch

The Command Class Color Switch was refactored to no longer require initialization on the application level. The initialization is now handled internally by ZAF on `ZAF_Init`.

The color configuration is now also handled by the Command Class and not the application. The configuration is available using the Z-Wave Command Class Configurator. The array of all colors supported by the application is no longer needed since it is created by the Command Class based on the configuration.

The functionality that each color component had in its callback function has been removed. Now all color components call the `cc_color_switch_cb` with the color component as parameter. In the `CC_ColorSwitch_Init` one of the parameters was a callback function. This function is now a weak function defined in the command class `cc_color_switch_refresh_cb`.

CC Door Lock

The Door Lock Command Class has been refactored so that all logic related to the Command Class is handled internally by it. The Command Class had five extern functions that had to be implemented by the application.

`CC_DoorLock_OperationSet_handler` , `CC_DoorLock_OperationGet_handler` , `CC_DoorLock_ConfigurationSet_handler` , `CC_DoorLock_ConfigurationGet_handler` and `CC_DoorLock_CapabilitiesGet_handler` . These functions are now implemented inside the command class and six new functions were created:

- `CC_DoorLock_Toggle` to toggle the door lock mode.
- `cc_door_lock_latch_status_handler` to be called when the latch status changes (HW specific function).
- `cc_door_lock_bolt_status_handler` to be called when the bolt status changes (HW specific function).
- `CC_DoorLock_update` to set door conditions (LED) from door handle mode and state.
- `CC_DoorLock_SetOutsideDoorHandleState` to set the Outside Door Handle State for a specific handle.
- `CC_DoorLock_ClearOutsideDoorHandleState` to clear the Outside Door Handle State for a specific handle.

The configuration is now also handled by the Command Class and not the application. The configuration is available using the `Z-Wave Command Class Configurator` .

All type redefinitions and function redefinitions kept for backwards compatibility were removed.

The Door Lock data is now owned by the Command Class and not the application. In our sample application the data was stored in `FILE_ID_APPLICATIONDATA` and code to migrate this information from the application into the command class was added with this assumption. If an application is storing this information in another file ID, the define `DOOR_LOCK_DATA_LEGACY_FILE_ID` in `cc_door_lock_nvm.c` needs to be changed to match this other file ID

CC Firmware Update Metadata

`APP_FIRMWARE_ID` , previously defined in `config_app.h` , has been obsoleted, as well as function `handleFirmWareIdGet()` . Instead, firmware ID (16 bit) is hard-coded in `ota_util.c` to consist of product type ID LSB and product ID LSB.

CC Indicator

`CC_Indicator_Init()` was removed and, as a replacement for the callback parameter, a weakly- defined function was added: `cc_indicator_handler()` . This function should be defined by the platform-specific code to override the weakly-defined function.

If an existing function was previously passed to `CC_Indicator_Init()` , simply rename that function to `cc_indicator_handler()` .

CC Manufacturer Specific

CC Manufacturer Specific was changed so that all applications can be built without this CC. This was done to enable building an even smaller application for updating Secure Element.

As a result, the following files are removed from the SDK:

- `CC_ManufacturerSpecific.h`
- `CC_ManufacturerSpecific_config.c`
- `CC_ManufacturerSpecific_config.h`

This includes following functions:

- `CC_ManufacturerSpecific_ManufacturerSpecificGet_handler()`
- `CC_ManufacturerSpecific_DeviceSpecificGet_handler()`

`APP_MANUFACTURER_ID` , `APP_PRODUCT_ID` and `PRODUCT_TYPE_ID` have been moved to `zaf_config.h` and renamed to `ZAF_CONFIG_MANUFACTURER_ID` , `ZAF_CONFIG_PRODUCT_ID` , and `ZAF_CONFIG_PRODUCT_TYPE_ID` , respectively.

Manufacturer ID, product type ID, and product ID must be set in Simplicity Studio in the Software Components tab.

Device Specific Report contains the chip serial number per default as this is what Silicon Labs recommends. If anything else is desired, `CC_ManufacturerSpecific.c` can be altered.

CC Multilevel Sensor

The sensor configuration is now also handled by the Command Class and not the application. The configuration is available using the Z-Wave Command Class Configurator. The `sensor_interface_t` and `cc_configuration_t` in the application are no longer needed.

In `sensor_interface_t` three callbacks are now weak functions in the Command Class:

`cc_multilevel_sensor_{{sensor_instance}}_interface_init`, `cc_multilevel_sensor_{{sensor_instance}}_interface_deinit` and `cc_multilevel_sensor_{{sensor_instance}}_interface_read_value`. The functions' behavior and parameters were not changed so renaming them should be enough.

In the `cc_configuration_init` function one of the parameters was the `cc_configuration_io_interface_t`, which defined two callback functions for reading and writing. These functions are now two weak functions defined inside the Command Class:

`cc_configuration_io_write` and `cc_configuration_io_read`. The functions' behavior and parameters were not changed so renaming them should be enough.

CC Multilevel Switch

The Command Class Multilevel Switch was refactored to no longer require initialization on the application level. The initialization is now handled internally by ZAF on `ZAF_Init`.

The switch configuration is now also handled by the Command Class and not the application. The configuration is available using the Z-Wave Command Class Configurator. The array of all switches supported by the application is no longer needed since it is created by the Command Class based on the configuration.

In the `cc_multilevel_switch_init` one of the parameters was a callback function. This function is now a weak function defined in the command class `cc_multilevel_switch_support_cb`.

CC Notification

The Command Class Notification was refactored to no longer require initialization on the application level. The initialization is now handled internally by ZAF on `ZAF_Init` and `ZAF_Reset`. `MAX_NOTIFICATION` define was removed from `config_app.h` and moved to `cc_notification_config.h`. This value can be configured in the Simplicity Studio GUI.

- Functions `NotificationEventTrigger()` and `UnsolicitedNotificationAction()` have been merged into a single function named `CC_Notification_TriggerAndTransmit()`.
- Notification CC now handles triggering of TSE by itself.
- Configuration of CC Notification is now handled by the Z-Wave Command Class Configurator and functions `AddNotification()` and `DefaultNotificationStatus()` have been obsoleted.
- Polling mode has been removed.
 - Implementation of deprecated polling mode has been removed from CC Notification.
 - This includes removing enum `NOTIFICATION_STATUS_SET`. Since poll mode is not supported anymore, notification status is the same for all commands.
 - CC Notification now supports push mode only.

CC Supervision

The Command Class Supervision is refactored to no longer require initialization on the application level. The initialization is now handled internally by ZAF on `ZAF_Init`.

The status updates can be enabled/disabled in the Simplicity Studio GUI.

The `cc_supervision_get_received_handler(...)` and `cc_supervision_report_recived_handler(...)` handler callbacks defined in `cc_supervision_handlers.h` have a dummy weakly-defined implementation and can be redefined by the applications.

CC User Code

The User Code Command Class has been refactored so that all logic related to the Command Class is handled internally by it. The Command Class had four extern functions that had to be implemented by the application. `CC_UserCode_Set_handler` , `CC_UserCode_getId_handler` , `CC_UserCode_Report_handler` and `CC_UserCode_UsersNumberReport_handler` . These functions are now implemented inside the command class and two new functions were created:

- `CC_UserCode_reset_data` to reset all the data persisted by the command class.
- `CC_UserCode_Validate` that contains all the validation logic.

Two configuration parameters related to this Command Class that belonged in the application now belong in the Command Class: `CC_USER_CODE_MAX_IDS` (formerly known as `USER_ID_MAX`) and `CC_USER_CODE_DEFAULT` (formerly known as `DEFAULT_USERCODE`).

CC Wake Up

The Command Class Wake Up was refactored to no longer require initialization on the application level. The initialization is now handled internally by ZAF on `ZAF_Init` .

The `CC_WakeUp_setConfiguration()` API was removed from the applications. These parameters can be configured in the Simplicity Studio GUI under the Wake Up Command Class's component.

CC Z-Wave Plus Info

The icon types for the root device and optionally for the endpoints is now configurable. The configuration is handled by the Command Class.

The user icon type and installer icon type for the root device are defined in ZAF config and can be configured in Simplicity Studio.

If an application supports Command Class Multi Channel, the icon types of an endpoint must be configured in the `app.cc_config` file. See the configuration of Power Strip for an example.

7.18.x and Previous Releases

See the corresponding release notes.

Configuration

Configuration

Configuration.

Modules

[Command Class Association - Group and Nodes configuration](#)

[Command Class Battery Configuration](#)

[Command Class Binary Switch Configuration](#)

[Command Class Color Switch Configuration](#)

[Command Class Firmware Update configuration](#)

[Command Class Multi Channel Configuration](#)

[Command Class Multilevel Sensor Configuration](#)

[Command Class Multilevel Switch Configuration](#)

[Command Class Notification- notifications configuration](#)

[Command Class Supervision Configuration](#)

[Command Class User Code Configuration](#)

[Command Class User Credential Configuration](#)

[Command Class User Credential I/O Configuration](#)

[Command Class Wake Up Configuration](#)

[ZAF Application Name Configuration](#)

[ZAF Event Distributor Core Configuration](#)

[ZAF Event Distributor Core Configuration](#)

[Application Version Configuration](#)

[Command Class Firmware Update - Bootloader update](#)

[Command Class Firmware Update - Firmware targets](#)

[Command Class ZwavePlusInfo Configuration](#)

[Version Configuration](#)

[Command Class Manufacturer Specific](#)

[Device type](#)

[Multi Channel Configuration](#)

[Security Configuration](#)

[General](#)

[Output channels](#)

[Component filtering](#)

[Z-Wave Radio Configuration](#)

[Max Tx Power Configuration](#)

Command Class Association - Group and Nodes configuration

Command Class Association - Group and Nodes configuration

Command Class Association - Group and Nodes configuration.

Macros

```
#define CC_ASSOCIATION_MAX_GROUPS_PER_ENDPOINT 1  
Max number of groups <1..18:1>  
  
#define CC_ASSOCIATION_MAX_NODES_IN_GROUP 5  
Max number of nodes <1..18:1>
```

Command Class Battery Configuration

Command Class Battery Configuration

Command Class Battery Configuration.

Macros

```
#define CC_BATTERY_REPORTING_DECREMENTS 10  
Battery reporting decrements <1..255:1>
```

Command Class Binary Switch Configuration

Command Class Binary Switch Configuration

Command Class Binary Switch Configuration.

Macros

```
#define CC_BINARY_SWITCH_CONFIG_DEFAULT_DURATION 0  
Factory default duration in seconds <1..255:1>
```

Command Class Color Switch Configuration

Command Class Color Switch Configuration

Command Class Color Switch Configuration.

Macros

```
#define CC_COLOR_SWITCH_CONFIG_DEFAULT_DURATION_SEC 2
Factory default duration in seconds <1..255:1>
```


Command Class Firmware Update configuration

Command Class Firmware Update configuration

Command Class Firmware Update configuration.

Macros

```
#define CC_FIRMWARE_UPDATE_CONFIG_OTA_CACHE_SIZE 200  
OTA Cache size <1..65535:1>  
  
#define CC_FIRMWARE_UPDATE_CONFIG_SUPPORT_ACTIVATION 1  
Activation supported.  
  
#define CC_FIRMWARE_UPDATE_CONFIG_OTA_MULTI_FRAME 1  
OTA Multi frames enabled.
```

Command Class Multi Channel Configuration

Command Class Multi Channel Configuration

Command Class Multi Channel Configuration.

Macros

```
#define CC_MULTI_CHANNEL_IDENTICAL_ENDPOINTS 1  
Identical endpoints.
```

Command Class Multilevel Sensor Configuration

Command Class Multilevel Sensor Configuration

Command Class Multilevel Sensor Configuration.

Macros

```
#define MULTILEVEL_SENSOR_DEFAULT_AUTOREPORT_PERIOD_MIN 30
Autoreport period in minutes <1..3600:1>

#define MULTILEVEL_SENSOR_REGISTERED_SENSOR_NUMBER_LIMIT 3
Maximum number of sensors <1..20:1>
```

Command Class Multilevel Switch Configuration

Command Class Multilevel Switch Configuration

Command Class Multilevel Switch Configuration.

Macros

```
#define CC_MULTILEVEL_SWITCH_CONFIG_DEFAULT_DURATION_SEC 1
Factory default duration in seconds <1..255:1>
```

Command Class Notification- notifications configuration

Command Class Notification- notifications configuration

Command Class Notification- notifications configuration.

Macros

```
#define CC_NOTIFICATION_MAX_NOTIFICATIONS 1  
Max number of notifications <1..255:1>
```

Command Class Supervision Configuration

Command Class Supervision Configuration

Command Class Supervision Configuration.

Macros

```
#define CC_SUPERVISION_CONFIG_DEFAULT_STATUS_UPDATES_ENABLED 0
Status updates enabled.
```

Command Class User Code Configuration

Command Class User Code Configuration

Command Class User Code Configuration.

Macros

```
#define CC_USER_CODE_MAX_IDS 1
Max number of user Code ID's and USER CODES <1..255:1>

#define CC_USER_CODE_DEFAULT { 0x33, 0x34, 0x39, 0x34 }
The default user code used by the Command Class.
```

Command Class User Credential Configuration

Command Class User Credential Configuration

Command Class User Credential Configuration.

Macros

#define	CC_USER_CREDENTIAL_MAX_USER_UNIQUE_IDENTIFIERS 20 Number of supported User Unique Identifiers <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_LENGTH_USER_NAME 16 Maximum length of User Name <1..255:2>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_PIN_CODE 5 Number of Supported Credential Slots for PIN Code data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_PASSWORD 5 Number of Supported Credential Slots for Password data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_RFID_CODE 0 Number of Supported Credential Slots for RFID Code data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_BLE 0 Number of Supported Credential Slots for BLE data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_NFC 0 Number of Supported Credential Slots for NFC data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_UWB 0 Number of Supported Credential Slots for UWB data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_EYE_BIOMETRIC 0 Number of Supported Credential Slots for Eye Biometric data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_FACE_BIOMETRIC 0 Number of Supported Credential Slots for Face Biometric data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_FINGER_BIOMETRIC 0 Number of Supported Credential Slots for Finger Biometric data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_HAND_BIOMETRIC 0 Number of Supported Credential Slots for Hand Biometric data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_CREDENTIAL_SLOTS_UNSPECIFIED_BIOMETRIC 0 Number of Supported Credential Slots for Unspecified Biometric data <1..65535:1>
#define	CC_USER_CREDENTIAL_MAX_HASH_LENGTH_PIN_CODE 0 Maximum length of PIN Code hash <0..255:1>
#define	CC_USER_CREDENTIAL_MAX_HASH_LENGTH_PASSWORD 0 Maximum length of password hash <0..255:1>
#define	CC_USER_CREDENTIAL_MAX_HASH_LENGTH_RFID_CODE 0 Maximum length of RFID Code hash <0..255:1>


```
#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_BLE 0
Maximum length of BLE data hash <0..255:1>

#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_NFC 0
Maximum length of NFC data hash <0..255:1>

#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_UWB 4
Maximum length of UWB data hash <0..255:1>

#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_EYE_BIOMETRIC 4
Maximum length of Eye Biometric data hash <0..255:1>

#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_FACE_BIOMETRIC 4
Maximum length of Face Biometric data hash <0..255:1>

#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_FINGER_BIOMETRIC 4
Maximum length of Finger Biometric data hash <0..255:1>

#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_HAND_BIOMETRIC 4
Maximum length of Hand Biometric data hash <0..255:1>

#define CC_USER_CREDENTIAL_MAX_HASH_LENGTH_UNSPECIFIED_BIOMETRIC 4
Maximum length of Unspecified Biometric data hash <0..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_PIN_CODE 4
Minimum length of PIN Code data <4..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_PASSWORD 2
Minimum length of Password data <1..255:2>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_RFID_CODE 1
Minimum length of RFID Code data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_BLE 1
Minimum length of BLE data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_NFC 1
Minimum length of NFC data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_UWB 1
Minimum length of UWB data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_EYE_BIOMETRIC 1
Minimum length of Eye Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_FACE_BIOMETRIC 1
Minimum length of Face Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_FINGER_BIOMETRIC 1
Minimum length of Finger Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_HAND_BIOMETRIC 1
Minimum length of Hand Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MIN_DATA_LENGTH_UNSPECIFIED_BIOMETRIC 1
Minimum length of Unspecified Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_PIN_CODE 10
Maximum length of PIN Code data <4..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_PASSWORD 20
Maximum length of Password data <1..255:2>
```

```
#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_RFID_CODE 32
Maximum length of RFID Code data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_BLE 32
Maximum length of BLE data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_NFC 32
Maximum length of NFC data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_UWB 32
Maximum length of UWB data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_EYE_BIOMETRIC 32
Maximum length of Eye Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_FACE_BIOMETRIC 32
Maximum length of Face Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_FINGER_BIOMETRIC 32
Maximum length of Finger Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_HAND_BIOMETRIC 32
Maximum length of Hand Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_MAX_DATA_LENGTH_UNSPECIFIED_BIOMETRIC 32
Maximum length of Unspecified Biometric data <1..255:1>

#define CC_USER_CREDENTIAL_USER_TYPE_SUPPORTED_GENERAL 1
Support User Type 'General'.

#define CC_USER_CREDENTIAL_USER_TYPE_SUPPORTED_PROGRAMMING 1
Support User Type 'Programming'.

#define CC_USER_CREDENTIAL_USER_TYPE_SUPPORTED_NON_ACCESS 1
Support User Type 'Non-Access'.

#define CC_USER_CREDENTIAL_USER_TYPE_SUPPORTED_DURESS 0
Support User Type 'Duress'.

#define CC_USER_CREDENTIAL_USER_TYPE_SUPPORTED_DISPOSABLE 0
Support User Type 'Disposable'.

#define CC_USER_CREDENTIAL_USER_TYPE_SUPPORTED_EXPIRING 0
Support User Type 'Expiring'.

#define CC_USER_CREDENTIAL_USER_TYPE_SUPPORTED_REMOTE_ONLY 0
Support User Type 'Remote Only'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_PIN_CODE 1
Support Credential Learn functionality for Credential Type 'PIN Code'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_PASSWORD 0
Support Credential Learn functionality for Credential Type 'Password'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_RFID_CODE 0
Support Credential Learn functionality for Credential Type 'RFID Code'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_BLE 0
Support Credential Learn functionality for Credential Type 'BLE'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_NFC 0
Support Credential Learn functionality for Credential Type 'NFC'.
```

```
#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_UWB 0
Support Credential Learn functionality for Credential Type 'UWB'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_EYE_BIOMETRIC 0
Support Credential Learn functionality for Credential Type 'Eye Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_FACE_BIOMETRIC 0
Support Credential Learn functionality for Credential Type 'Face Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_FINGER_BIOMETRIC 0
Support Credential Learn functionality for Credential Type 'Finger Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_HAND_BIOMETRIC 0
Support Credential Learn functionality for Credential Type 'Hand Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_SUPPORTED_UNSPECIFIED_BIOMETRIC 0
Support Credential Learn functionality for Credential Type 'Unspecified Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_PIN_CODE 20
Recommended Credential Learn timeout for Credential Type 'PIN Code'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_PASSWORD 20
Recommended Credential Learn timeout for Credential Type 'Password'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_RFID_CODE 20
Recommended Credential Learn timeout for Credential Type 'RFID Code'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_BLE 20
Recommended Credential Learn timeout for Credential Type 'BLE'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_NFC 20
Recommended Credential Learn timeout for Credential Type 'NFC'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_UWB 20
Recommended Credential Learn timeout for Credential Type 'UWB'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_EYE_BIOMETRIC 20
Recommended Credential Learn timeout for Credential Type 'Eye Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_FACE_BIOMETRIC 20
Recommended Credential Learn timeout for Credential Type 'Face Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_FINGER_BIOMETRIC 20
Recommended Credential Learn timeout for Credential Type 'Finger Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_HAND_BIOMETRIC 20
Recommended Credential Learn timeout for Credential Type 'Hand Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_RECOMMENDED_TIMEOUT_UNSPECIFIED_BIOMETRIC 20
Recommended Credential Learn timeout for Credential Type 'Unspecified Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_PIN_CODE 1
Credential Learn Number of Steps for Credential Type 'PIN Code'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_PASSWORD 1
Credential Learn Number of Steps for Credential Type 'Password'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_RFID_CODE 1
Credential Learn Number of Steps for Credential Type 'RFID Code'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_BLE 1
Credential Learn Number of Steps for Credential Type 'BLE'.
```

```
#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_NFC 1
Credential Learn Number of Steps for Credential Type 'NFC'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_UWB 1
Credential Learn Number of Steps for Credential Type 'UWB'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_EYE_BIOMETRIC 1
Credential Learn Number of Steps for Credential Type 'Eye Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_FACE_BIOMETRIC 1
Credential Learn Number of Steps for Credential Type 'Face Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_FINGER_BIOMETRIC 1
Credential Learn Number of Steps for Credential Type 'Finger Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_HAND_BIOMETRIC 1
Credential Learn Number of Steps for Credential Type 'Hand Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_LEARN_NUMBER_OF_STEPS_UNSPECIFIED_BIOMETRIC 1
Credential Learn Number of Steps for Credential Type 'Unspecified Biometric'.

#define CC_USER_CREDENTIAL_CREDENTIAL_RULE_SUPPORTED_SINGLE 1
Support Credential Rule 'Single'.

#define CC_USER_CREDENTIAL_CREDENTIAL_RULE_SUPPORTED_DUAL 0
Support Credential Rule 'Dual'.

#define CC_USER_CREDENTIAL_CREDENTIAL_RULE_SUPPORTED_TRIPLE 0
Support Credential Rule 'Triple'.

#define CC_USER_CREDENTIAL_ALL_USERS_CHECKSUM_SUPPORTED 1
Support All Users Checksum.

#define CC_USER_CREDENTIAL_USER_CHECKSUM_SUPPORTED 1
Support User Checksum.

#define CC_USER_CREDENTIAL_CREDENTIAL_CHECKSUM_SUPPORTED 1
Support Credential Checksum.

#define CC_USER_CREDENTIAL_ADMIN_CODE_SUPPORTED 1
Support for Admin Code.

#define CC_USER_CREDENTIAL_ADMIN_CODE_DEACTIVATE_SUPPORTED 1
Support for Admin Code Deactivation.
```

Command Class User Credential I/O Configuration

Command Class User Credential I/O Configuration

Command Class User Credential I/O Configuration.

Macros

```
#define U3C_BUFFER_SIZE_CREDENTIAL_DATA 100
Size of buffer for holding Credential Data <1..255:1>

#define U3C_BUFFER_SIZE_USER_DESCRIPTOR 20
[SoC NVM driver] User Descriptor table buffer size <1..65535:1>

#define U3C_BUFFER_SIZE_CREDENTIAL_DESCRIPTOR 20
[SoC NVM driver] Credential Descriptor table buffer size <1..65535:1>
```

Command Class Wake Up Configuration

Command Class Wake Up Configuration

Command Class Wake Up Configuration.

Macros

```
#define CC_WAKE_UP_CONFIG_DEFAULT_SLEEP_TIME_SEC 300
Default sleep time between Wake Up Notifications <1..86400:1>

#define CC_WAKE_UP_CONFIG_MINIMUM_SLEEP_TIME_SEC 20
Minimum sleep time between Wake Up Notifications <20..86400:1>

#define CC_WAKE_UP_CONFIG_MAXIMUM_SLEEP_TIME_SEC 86400
Maximum sleep time between Wake Up Notifications <20..86400:1>

#define CC_WAKE_UP_CONFIG_SLEEP_STEP_TIME_SEC 20
Sleep step time <1..255:1>
```

ZAF Application Name Configuration

ZAF Application Name Configuration

ZAF Application Name Configuration.

Macros

```
#define ZAF_APP_NAME "zwave_application"  
Application Name (maximum length: 29)
```

ZAF Event Distributor Core Configuration

ZAF Event Distributor Core Configuration

ZAF Event Distributor Core Configuration.

Macros

```
#define ZAF_EVENT_DISTRIBUTOR_CORE_CONFIG_TIMEOUT 4294967295
Timeout (in ms) when waiting for event <1..4294967295:1>
```


ZAF Event Distributor Core Configuration

ZAF Event Distributor Core Configuration

ZAF Event Distributor Core Configuration.

Macros

```
#define ZAF_EVENT_DISTRIBUTOR_SOC_CONFIG_APP_QUEUE_SIZE 5
Application event queue size <1..255:1>

#define ZAF_EVENT_DISTRIBUTOR_SOC_CONFIG_CC_QUEUE_SIZE 2
Command Class event queue size <1..255:1>
```

Application Version Configuration

Application Version Configuration

Application Version Configuration.

Macros

```
#define APP_VERSION 1
Application Major Version <0..255:1>

#define APP_REVISION 0
Application Minor Version <0..255:1>

#define APP_PATCH 0
Application Patch Version <0..255:1>
```

Command Class Firmware Update - Bootloader update

Command Class Firmware Update - Bootloader update

Command Class Firmware Update - Bootloader update.

Macros

```
#define ZAF_CONFIG_BOOTLOADER_TARGET_ID 1
The bootloader target id <1..255:1>

#define ZAF_CONFIG_BOOTLOADER_UPGRADABLE 0
Indicates if the bootloader can be updated.
```

Command Class Firmware Update - Firmware targets

Command Class Firmware Update - Firmware targets

Command Class Firmware Update - Firmware targets.

Macros

```
#define ZAF_CONFIG_FIRMWARE_TARGET_COUNT 1  
Number of firmware targets <0..10:1>
```

Command Class ZwavePlusInfo Configuration

Command Class ZwavePlusInfo Configuration

Command Class ZwavePlusInfo Configuration.

Macros

```
#define ZAF_CONFIG_USER_ICON_TYPE ICON_TYPE_GENERIC_ON_OFF_POWER_SWITCH
User Icon Type.

#define ZAF_CONFIG_INSTALLER_ICON_TYPE ICON_TYPE_GENERIC_ON_OFF_POWER_SWITCH
Installer Icon Type.
```

Version Configuration

Version Configuration

Version Configuration.

Macros

```
#define ZAF_CONFIG_HARDWARE_VERSION 1
Hardware version <0..255:1>
```

Command Class Manufacturer Specific

Command Class Manufacturer Specific

Command Class Manufacturer Specific.

Macros

```
#define ZAF_CONFIG_MANUFACTURER_ID 0x0000
Manufacturer ID <0x0000..0xFFFF:0x1>

#define ZAF_CONFIG_PRODUCT_TYPE_ID 0x0004
Product Type ID <0x0000..0xFFFF:0x1>

#define ZAF_CONFIG_PRODUCT_ID 0x0000
Product ID <0x0000..0xFFFF:0x1>
```

Device type

Device type

Device type.

Macros

```
#define ZAF_CONFIG_GENERIC_TYPE 0
The generic type of a device.

#define ZAF_CONFIG_SPECIFIC_TYPE SPECIFIC_TYPE_NOT_USED
The specific type of a device.
```


Multi Channel Configuration

Multi Channel Configuration

Multi Channel Configuration.

A multi channel device must specify the number of supported end points. An example of a multi channel device could be a Power Strip for which each outlet is represented by an end point.

Macros

```
#define ZAF_CONFIG_NUMBER_OF_END_POINTS 0
Number of end points <0..255:1>
```

Security Configuration

Security Configuration

Security Configuration.

Macros

```
#define ZAF_CONFIG_REQUEST_KEY_S0 0
Request S0 network key.

#define ZAF_CONFIG_REQUEST_KEY_S2_UNAUTHENTICATED 0
Request S2 Unauthenticated network key.

#define ZAF_CONFIG_REQUEST_KEY_S2_AUTHENTICATED 0
Request S2 Authenticated network key.

#define ZAF_CONFIG_REQUEST_KEY_S2_ACCESS 0
Request S2 Access network key.

#define ZAF_CONFIG_SECURE_ONLY 0
Application can only be controlled when included securely.
```

General

General

General.

Macros

```
#define ZW_LOG_BUFFER_SIZE 96
Size of the log buffer <1..255:1>

#define ZW_LOG_DISPLAY_LEVEL 1
Display log level.

#define ZW_LOG_DISPLAY_COMPONENT 0
Display log source component.
```

Output channels

Output channels

Output channels.

Macros

```
#define ZW_LOG_CHANNEL_DEBUG ""  
Output channel for Debug messages.  
  
#define ZW_LOG_CHANNEL_INFO "stdout"  
Output channel for Info messages.  
  
#define ZW_LOG_CHANNEL_WARNING ""  
Output channel for Warning messages.  
  
#define ZW_LOG_CHANNEL_ERROR "stderr"  
Output channel for Error messages.
```

Component filtering

Component filtering

Component filtering.

Macros

```
#define ZW_LOG_ENABLE_APP 1
Application.

#define ZW_LOG_ENABLE_HW 0
Hardware.

#define ZW_LOG_ENABLE_CC_ASSOCIATION 0
CC Association.

#define ZW_LOG_ENABLE_CC_BATTERY 0
CC Battery.

#define ZW_LOG_ENABLE_CC_BINARY_SWITCH 0
CC Binary Switch.

#define ZW_LOG_ENABLE_CC_CENTRAL_SCENE 0
CC Central Scene.

#define ZW_LOG_ENABLE_CC_COLOR_SWITCH 0
CC Color Switch.

#define ZW_LOG_ENABLE_CC_DOOR_LOCK 0
CC Door Lock.

#define ZW_LOG_ENABLE_CC_FIRMWARE_UPDATE 0
CC Firmware Update.

#define ZW_LOG_ENABLE_CC_INDICATOR 0
CC Indicator.

#define ZW_LOG_ENABLE_CC_MULTI_CHANNEL 0
CC Multi Channel.

#define ZW_LOG_ENABLE_CC_MULTILEVEL_SWITCH 0
CC Multilevel Switch.

#define ZW_LOG_ENABLE_CC_NOTIFICATION 0
CC Notification.

#define ZW_LOG_ENABLE_CC_USER_CODE 0
CC User Code.

#define ZW_LOG_ENABLE_CC_USER_CREDENTIAL 0
CC User Credential.

#define ZW_LOG_ENABLE_CC_VERSION 0
CC Version.

#define ZW_LOG_ENABLE_CC_WAKE_UP 0
CC Wake Up.
```

```
#define ZW_LOG_ENABLE_CC_ZWAVE_PLUS_INFO 0
CC Z-Wave Plus Info.

#define ZW_LOG_ENABLE_ZAF_ACTUATOR 0
ZAF Actuator.

#define ZW_LOG_ENABLE_ZAF_APP_TIMER 0
ZAF App Timer.

#define ZW_LOG_ENABLE_ZAF_CMD_PUBLISHER 0
ZAF Command Publisher.

#define ZW_LOG_ENABLE_ZAF_COMMON 0
ZAF Common.

#define ZW_LOG_ENABLE_ZAF_EVENT_DISTRIBUTOR 0
ZAF Event Distributor.

#define ZW_LOG_ENABLE_ZAF_NVM 0
ZAF NVM.

#define ZW_LOG_ENABLE_ZAF_TRANSPORT 0
ZAF Transport.

#define ZW_LOG_ENABLE_ZAF_TSE 0
ZAF TSE.
```

Z-Wave Radio Configuration

Z-Wave Radio Configuration

Z-Wave Radio Configuration.

Macros

```
#define ZW_REGION REGION_DEFAULT
The supported region of the device.
```

Max Tx Power Configuration

Max Tx Power Configuration

Max Tx Power Configuration.

Macros

```
#define APP_MAX_TX_POWER_LR 200
The maximum allowed Tx power in deci dBm for Z-Wave Long Range network <-200..200:1>

#define APP_MAX_TX_POWER 200
The maximum allowed Tx power in deci dBm for Z-Wave network <-200..200:1>

#define APP_MEASURED_ODBM_TX_POWER 0
The deci dBm output measured at a PA setting of 0dBm <-50..50:1>

#define ENABLE_RADIO_DEBUG 0
Enable Radio Debug.
```


Command Classes

Command Classes

Indicator command class version 3.

- 2018 Silicon Laboratories Inc.
- 2019 Silicon Laboratories Inc.
- 2022 Silicon Laboratories Inc.

Modules

[\(Multi Channel\) Association](#)

[Association Group Information](#)

[Basic](#)

[Battery](#)

[Binary Switch](#)

[CentralScene](#)

[ColorSwitch](#)

[Common](#)

[Configuration](#)

[DeviceResetLocally](#)

[DoorLock](#)

[Firmware Update Meta Data](#)

[Indicator](#)

[Multichan](#)

[MultilevelSensor](#)

[MultilevelSwitch](#)

[Notification](#)

[Supervision](#)

[UserCode](#)

[UserCredential](#)

[WakeUp](#)

[ZWavePlusInfo](#)

(Multi Channel) Association

(Multi Channel) Association

CC Association and CC Multi Channel Association enable application level control of other devices.

For instance, a wall switch can control an LED bulb if an association to the LED bulb is created in the wall switch. Another example could be a motion sensor that turns on an LED bulb when motion is detected. Such associations can be created using either Association Set or Multi Channel Association Set.

Both CCs share a common module that stores the actual associations.

The specifications of CC Association and CC Multi Channel Association can be found in https://github.com/Z-Wave-Alliance/AWG/tree/main/source/management_command_classes/command_class_definitions

Modules

[ASSOCIATION_GROUP](#)

[ASSOCIATION_GROUP_PACKED](#)

[SAssociationInfo](#)

Enumerations

```
enum    NODE\_LIST\_STATUS {
        NODE_LIST_STATUS_SUCCESS = 0
        NODE_LIST_STATUS_NO_MORE_NODES
        NODE_LIST_STATUS_ASSOCIATION_LIST_EMPTY
        NODE_LIST_STATUS_ERROR_LIST
        NODE_LIST_STATUS_ERR_NO_TABLE_ENDPOINT
        NODE_LIST_STATUS_ERR_UNKNOWN_PROFILE
        NODE_LIST_STATUS_ERR_ENDPOINT_OUT_OF_RANGE
        NODE_LIST_STATUS_ERR_GROUP_NBR_NOT_LEGAL
        NODE_LIST_STATUS_ERR_LIFELINE_PROFILE_NOT_SUPPORTED
        NODE_LIST_STATUS_ERR_LIFELINE_SUPPORT_NOT_CC_BASIC
        NODE_LIST_STATUS_ERR_PROFILE_LIFELINE_ONLY_SUPPORT_IN_GRP_1
    }
Enum type NODE\_LIST\_STATUS is used for return status on API call AGLNodeIdListGetNext.
```

Typedefs

```
typedef struct    SAssociationInfo
SAssociationInfo
```

Functions

```
void    CC\_Association\_Init(void)
```

```
void    CC\_Association\_Reset(void)
```

```
NODE\_LIST\_STATUS    handleAssociationGetnodeList(uint8_t groupId, uint8_t ep, destination_info_t **ppListOfNodes, uint8_t *pListLen)
US
Delivers a list of nodes in a given association group corresponding to a given endpoint.
```

e_cmd_handler_re turn_code_t	AssociationRemove (uint8_t groupId, uint8_t ep, ZW_MULTI_CHANNEL_ASSOCIATION_REMOVE_1BYTE_V2_FRAME *pCmd, uint8_t cmdLength) Removes all nodes or given node(s) from all groups or a given group.
void	AssociationGet (uint8_t endpoint, uint8_t *incomingFrame, uint8_t *outgoingFrame, uint8_t *outgoingFrameLength) Handles an incoming (Multi Channel) Association Get command and composes a (Multi Channel) Association Report.
bool	AssociationAddNode (uint8_t groupId, uint8_t endpoint, MULTICHAN_DEST_NODE_ID *pNode, bool multiChannelAssociation) Associates a given node in the given group for a given endpoint.
void	AssociationGetDestinationInit (destination_info_t *pFirstDestination) Initializes the fetching of nodes before e.g.
bool	AssociationGetBitAddressingDestination (destination_info_t **ppNodeList, uint8_t *pListLength, destination_info_t *pNode) Outputs a node where associations to multiple endpoints of that node exist.
uint32_t	AssociationGetSinglecastNodeCount (void) Returns the number of associations after the bit addressing destinations are removed.
destination_info_t *	AssociationGetNextSinglecastDestination (void) Returns the address of the next association like an iterator.
uint8_t	AssociationGetSinglecastEndpointDestinationCount (void) Returns the number of endpoint destinations in the active node association list.
uint8_t	ApplicationGetLastActiveGroupId (void) Returns the latest used association group.
TRANSMIT_OPTIO NS_TYPE_EX *	ReqNodeList (AGI_PROFILE const *const pProfile, cc_group_t const *const pCurrentCmdGrp, const uint8_t sourceEndpoint) Returns TX options containing a list of nodes found in the association group that matches the given AGI profile, command class / command pair and source endpoint.
e_cmd_handler_re turn_code_t	handleAssociationSet (uint8_t ep, ZW_MULTI_CHANNEL_ASSOCIATION_SET_1BYTE_V2_FRAME *pCmd, uint8_t cmdLength, uint8_t commandClass) Handler for Association Set command.

Macros

#define	FREE_VALUE 0x00FF
#define	NOT_VALID_GROUP_ID 0 Invalid group ID.
#define	LIFELINE_GROUP_ID 1 LIFELINE group ID.
#define	LIFELINE_ENDPOINT_ALLOWED 0 The only allowed endpoint of the Lifeline group!
#define	ZAF_FILE_SIZE_ASSOCIATIONINFO (sizeof(SAssociationInfo))

Enumeration Documentation

NODE_LIST_STATUS

NODE_LIST_STATUS

Enum type NODE_LIST_STATUS is used for return status on API call AGI_NodeIdListGetNext.

Enum types from NODE_LIST_STATUS_SUCCESS to NODE_LIST_STATUS_ERROR_LIST deliver status on the call and after NODE_LIST_STATUS_ERROR_LIST deliver an error identifiers pointing to a problem in application AGI/association configuration.

Enumerator	
NODE_LIST_STATUS_SUCCESS	
NODE_LIST_STATUS_NO_MORE_NODES	
NODE_LIST_STATUS_ASSOCIATION_LIST_EMPTY	
NODE_LIST_STATUS_ERROR_LIST	enum values higher than this is error identifiers
NODE_LIST_STATUS_ERR_NO_TABLE_ENDPOINT	
NODE_LIST_STATUS_ERR_UNKNOWN_PROFILE	
NODE_LIST_STATUS_ERR_ENDPOINT_OUT_OF_RANGE	
NODE_LIST_STATUS_ERR_GROUP_NBR_NOT_LEGAL	
NODE_LIST_STATUS_ERR_LIFELINE_PROFILE_NOT_SUPPORTED	
NODE_LIST_STATUS_ERR_LIFELINE_SUPPORT_NOT_CC_BASIC	
NODE_LIST_STATUS_ERR_PROFILE_LIFELINE_ONLY_SUPPORT_IN_GRP_1	

Typedef Documentation

SAssociationInfo

typedef struct SAssociationInfo SAssociationInfo

Function Documentation

CC_Association_Init

void CC_Association_Init (void)

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Remarks

- Must be invoked by CC_Association.c only

CC_Association_Reset

void CC_Association_Reset (void)

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Remarks

- Must be invoked by CC_Association.c only

handleAssociationGetnodeList

```
NODE_LIST_STATUS handleAssociationGetnodeList (uint8_t groupId, uint8_t ep, destination_info_t ** ppListOfNodes,
uint8_t * pListLen)
```

Delivers a list of nodes in a given association group corresponding to a given endpoint.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	groupId	Association group ID.
uint8_t	[in]	ep	Endpoint.
destination_info_t **	[out]	ppListOfNodes	is out double-pointer of type MULTICHAN_NODE_ID deliver node list
uint8_t *	[out]	pListLen	length of list

Returns

- enum type NODE_LIST_STATUS

AssociationRemove

```
e_cmd_handler_return_code_t AssociationRemove (uint8_t groupId, uint8_t ep,
ZW_MULTI_CHANNEL_ASSOCIATION_REMOVE_1BYTE_V2_FRAME * pCmd, uint8_t cmdLength)
```

Removes all nodes or given node(s) from all groups or a given group.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	groupId	Association group ID.
uint8_t	[in]	ep	Multi Channel Endpoint.
ZW_MULTI_CHANNEL_ASSOCIATION_REMOVE_1BYTE_V2_FRAME *	[in]	pCmd	Pointer to the command containing the node IDs to remove.
uint8_t	[in]	cmdLength	Length of the command.

See Association CC and Multi Channel Association CC for details. Returns

- command handler return code

AssociationGet

```
void AssociationGet (uint8_t endpoint, uint8_t * incomingFrame, uint8_t * outgoingFrame, uint8_t * outgoingFrameLength)
```

Handles an incoming (Multi Channel) Association Get command and composes a (Multi Channel) Association Report.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	The endpoint from which the associated nodes must be read.
uint8_t *	[in]	incomingFrame	The incoming frame including CC and command.
uint8_t *	[out]	outgoingFrame	The composed frame ready for transmission.
uint8_t *	[out]	outgoingFrameLength	The total length of the outgoing frame.

AssociationAddNode

```
bool AssociationAddNode (uint8_t groupID, uint8_t endpoint, MULTICHAN_DEST_NODE_ID * pNode, bool multiChannelAssociation)
```

Associates a given node in the given group for a given endpoint.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	groupID	ID of the group in which the association must be made.
uint8_t	N/A	endpoint	Endpoint for which the association must be made.
MULTICHAN_DEST_NODE_ID *	N/A	pNode	Pointer to a node with info about node ID, endpoint, etc.
bool	N/A	multiChannelAssociation	Specifies whether the associated node ID includes an endpoint or not.

The endpoint argument specifies the local endpoint for which the association is made. E.g. if this device would be a Wall Controller/Switch with 4 endpoints, one for each switch, and an association was to be made for endpoint 1. The associated node would receive something when button 1 is pressed (if of course button one represents endpoint 1). **Returns**

- true if association is added, false otherwise.

AssociationGetDestinationInit

```
void AssociationGetDestinationInit (destination_info_t * pFirstDestination)
```

Initializes the fetching of nodes before e.g.

Parameters

Type	Direction	Argument Name	Description
destination_info_t *	[in]	pFirstDestination	Address of the first association in a group. Retrieved by invoking <code>handleAssociationGetnodeList()</code> .

a transmission. It places a destination_node_list into the system.

MUST be invoked before `AssociationGetBitAdressingDestination()`, `AssociationGetSinglecastNodeCount()` and `AssociationGetNextSinglecastDestination()`.

AssociationGetBitAdressingDestination

```
bool AssociationGetBitAdressingDestination (destination_info_t ** ppNodeList, uint8_t * pListLength, destination_info_t * pNode)
```

Outputs a node where associations to multiple endpoints of that node exist.

Parameters

Type	Direction	Argument Name	Description
destination_info_t **	[inout]	ppNodeList	- Address of pointer to an element in the node list. The pointer will be updated as this function travels through the node list.
uint8_t *	[inout]	pListLength	- Pointer to length of the node list. The value will be updated by this function.
destination_info_t *	[out]	pNode	- Address of the memory to where the node is written.

The pNode [OUT] (list) will contain all the endpoints using bit addressing, for endpoints equal or less than 7.

Operation: For the first encounter of a nodeID and an endpoint, this function will search for the same nodeID and add their endpoints to the endpoint-field as bit-flags. First time this function encounters a node without a valid endpoint for bit-addressing, it will terminate and return false because the list is sorted.

If there are zero nodes with multiple endpoints associated, the function will return false and write NULL to the pointer returned, pNode.

If further nodes with multiple endpoints associated exist after the initial list is created, the function will return true and indicate that this function is not finished.

If the function returns true, the invoker must invoke the function again to get the next node.

If there are no more nodes with multiple endpoints, the function will return false, as this function is only designed to return node-destinations with endpoints.

Example with 6 associations in total: (This is a pre-sorted list in accordance to requirements of this function!) 3.1 3.2 4.1 4.2 5.1 6.0

```
destination_info_t * pNodeList // Must point to the first node in an association group.
uint8_t listLength; // Must be set to the length of the node list.
destination_info_t node;
bool moreNodes;

moreNodes = AssociationGetBitAdressingDestination(&pNodeList, &listLength, &node);
// moreNodes is TRUE. (4.1 is still showing endpoint destination)
// pNodeList now points to node 4 endpoint 1.
// listLength now equals 4. (the two from the top with endpoints are removed from list)
// Bit-addressing is set to TRUE. (because 3.1 and 3.2 are bit-addressed)
// → The prepared node contains the first nodeID (3) including the two endpoints as bit addressing.

moreNodes = AssociationGetBitAdressingDestination(&pNodeList, &listLength, &node);
// moreNodes is TRUE. (5.1 is still showing endpoint destination)
// pNodeList now points to node 5 endpoint 1.
// listLength now equals 2. (the four destinations from the top with endpoints are removed from list)
// Bit-addressing is set to TRUE. (because 4.1 and 4.2 are bit-addressed)
// → The prepared node contains the second nodeID (4) including the two endpoints as bit addressing.

moreNodes = AssociationGetBitAdressingDestination(&pNodeList, &listLength, &node);
// moreNodes is FALSE. (6.0 is not having an endpoint destination)
// pNodeList now points to node 6 with no endpoints.
// listLength now equals 2. (the four destinations from the top remain the only removed from list)
// Bit-addressing is set to FALSE. (5.1 was the only endpoint destination of nodeID 5)
```

```
// The nodeID 6 destination is not processed for multichannel transmissions due to how the list is sorted!
```

Returns

- Returns true if this function needs to be called again for a different nodeID with endpoints, otherwise false.

AssociationGetSinglecastNodeCount

```
uint32_t AssociationGetSinglecastNodeCount (void )
```

Returns the number of associations after the bit addressing destinations are removed.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function MUST be invoked AFTER [AssociationGetBitAdressingDestination\(\)](#) has returned false.

Returns

- Returns the number of remaining nodes when there are no more bit addressing nodes.

AssociationGetNextSinglecastDestination

```
destination_info_t * AssociationGetNextSinglecastDestination (void )
```

Returns the address of the next association like an iterator.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function needs to know beforehand how many items it needs to iterate through.

If the number of iterations are more than totalNodeCount, there will be repeated return of the same item. If the number of iterations are less than totalNodeCount, it will impact the next transmission using this iterator.

[AssociationGetDestinationInit\(\)](#) MUST have been invoked before this function.

This function must be invoked only AFTER [AssociationGetBitAdressingDestination\(\)](#) has returned false.

Continuously invoking this function will return the next association. If there are no more associations in the group, the first one in the same group will be returned and so forth.

If the group is empty, the function will ASSERT.

Returns

- Address of an association.

AssociationGetSinglecastEndpointDestinationCount

```
uint8_t AssociationGetSinglecastEndpointDestinationCount (void )
```


Returns the number of endpoint destinations in the active node association list.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

[AssociationGetDestinationInit\(\)](#) MUST have been invoked before this function.

Returns

- Number of destination endpoints found in list.

ApplicationGetLastActiveGroupId

```
uint8_t ApplicationGetLastActiveGroupId (void )
```

Returns the latest used association group.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Latest used association group.

ReqNodeList

```
TRANSMIT_OPTIONS_TYPE_EX * ReqNodeList (AGI\_PROFILE const *const pProfile, cc\_group\_t const *const pCurrentCmdGrp, const uint8_t sourceEndpoint)
```

Returns TX options containing a list of nodes found in the association group that matches the given AGI profile, command class / command pair and source endpoint.

Parameters

Type	Direction	Argument Name	Description
AGI_PROFILE const *const	[in]	pProfile	Pointer to AGI profile.
cc_group_t const *const	[in]	pCurrentCmdGrp	Pointer to command class / command pair.
const uint8_t	[in]	sourceEndpoint	The endpoint for which the nodes are associated.

This function must be called to construct the TX options for [ZW_TransportMulticast_SendRequest\(\)](#). Returns

- transmit option pointer of type TRANSMIT_OPTIONS_TYPE_EX. Return NULL if something went wrong.

handleAssociationSet

```
e_cmd_handler_return_code_t handleAssociationSet (uint8_t ep, ZW_MULTI_CHANNEL_ASSOCIATION_SET_1BYTE_V2_FRAME * pCmd, uint8_t cmdLength, uint8_t commandClass)
```

Handler for Association Set command.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	ep	A given endpoint.
ZW_MULTI_CHANNEL_ASSOCIATION_SET_1BYTE_V2_FRAME*	[in]	pCmd	A command containing the nodes to save in the association database.
uint8_t	[in]	cmdLength	Length of the command.
uint8_t	[in]	commandClass	Caller command class. This function may be called from multiple command classes. This parameter tells which one was it.

Returns

- command handler return code

ASSOCIATION_GROUP

Public Attributes

MULTICHAN_NO subGrp
DE_ID

Public Attribute Documentation

subGrp

MULTICHAN_NODE_ID ASSOCIATION_GROUP::subGrp[CC_ASSOCIATION_MAX_NODES_IN_GROUP]

ASSOCIATION_GROUP_PACKED

Public Attributes

MULTICHAN_NO subGrp
DE_ID_PACKED

Public Attribute Documentation

subGrp

MULTICHAN_NODE_ID_PACKED ASSOCIATION_GROUP_PACKED::subGrp[CC_ASSOCIATION_MAX_NODES_IN_GROUP]

SAssociationInfo

Public Attributes

ASSOCIATION_G
ROUP_PACKED Groups

Public Attribute Documentation

Groups

ASSOCIATION_GROUP_PACKED SAssociationInfo::Groups[ZAF_CONFIG_NUMBER_OF_END_POINTS+1]
[CC_ASSOCIATION_MAX_GROUPS_PER_ENDPOINT]

Association Group Information

Association Group Information

General information

CC Association Group Information (CC AGI) reports which commands a device can control and which commands are reported to the Lifeline.

CC AGI configuration

The Lifeline group is automatically configured, but other groups may be manually configured if desired. For instance, a wall switch might control a bulb using CC Binary Switch. In such case, the wall switch would need to have an association group defined that lists Binary Switch Set. Simplicity Studio / SLC generates this configuration automatically based on user configuration in *.cc_config files.

Example of configuration:

```
zw_cc_agi:
  endpoints:
    - endpoint: 0
      groups:
        - name: BTNO
          profile_msb: ASSOCIATION_GROUP_INFO_REPORT_AGLPROFILE_CONTROL
          profile_lsb: ASSOCIATION_GROUP_INFO_REPORT_AGLCONTROL_KEY01
          commands:
            - command_class: COMMAND_CLASS_BASIC_V2
              command: BASIC_SET_V2
```

The specification of CC AGI can be found in https://github.com/Z-Wave-Alliance/AWG/tree/main/source/management_command_classes/command_class_definitions

Modules

[Configuration of CC Association Group Information](#)

Functions

uint8_t	CC_AGI_groupCount_handler (uint8_t endpoint) Returns the number of configured association groups for a given endpoint.
uint8_t	cc_agi_get_group_id (agi_profile_t const *const profile, ccc_pair_t const *const cmdGrp, uint8_t const endpoint) Returns the association group ID that matches the given AGI profile, command class/command-pair and endpoint.

Function Documentation

CC_AGI_groupCount_handler

```
uint8_t CC_AGI_groupCount_handler (uint8_t endpoint)
```

Returns the number of configured association groups for a given endpoint.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	A given endpoint where 0 is the root device.

This function will return 1 for any endpoint if the node is added to a Z-Wave LR network.

Returns

- Number of configured association groups.

cc_agi_get_group_id

```
uint8_t cc_agi_get_group_id (agi_profile_t const *const profile, ccc_pair_t const *const cmdGrp, uint8_t const endpoint)
```

Returns the association group ID that matches the given AGI profile, command class/command-pair and endpoint.

Parameters

Type	Direction	Argument Name	Description
agi_profile_t const *const	[in]	profile	AGI profile to search for.
ccc_pair_t const *const	[in]	cmdGrp	Command class / command pair to search for.
uint8_t const	[in]	endpoint	Endpoint.

Returns

- Returns the group ID, if found. Otherwise 0xFF.

Configuration of CC Association Group Information

Configuration of CC Association Group Information

Modules

[cc_agi_group_t](#)
[cc_agi_config_t](#)

Functions

- const
[cc_agi_config_t](#) *

[cc_agi_get_config](#)(void)
Returns the address of the first element in the configuration array.
- uint8_t

[cc_agi_config_get_group_count_by_endpoint](#)(uint8_t endpoint)
Returns the number of configured groups for a given endpoint.
- const
[cc_agi_group_t](#) *

[cc_agi_get_rootdevice_groups](#)(void)
Returns the address of the first element in the array of root device groups.

Function Documentation

cc_agi_get_config

```
const cc_agi_config_t * cc_agi_get_config (void )
```

Returns the address of the first element in the configuration array.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Address of the first element in the CC AGI configuration.

cc_agi_config_get_group_count_by_endpoint

```
uint8_t cc_agi_config_get_group_count_by_endpoint (uint8_t endpoint)
```

Returns the number of configured groups for a given endpoint.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	endpoint	Endpoint.

Returns

- Number of groups.

cc_agi_get_rootdevice_groups

```
const cc_agi_group_t * cc_agi_get_rootdevice_groups (void )
```

Returns the address of the first element in the array of root device groups.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- The first element is group 2.

Returns

- Address of first element in root device group array.

cc_agi_group_t

Defines an AGI group.

Public Attributes

```
const agi_profile_t    profile
           const uint8_t  ccc_pair_count
const ccc_pair_t *    ccc_pairs
           const char *    name
           const uint8_t  name_length
```

Public Attribute Documentation

profile

```
const agi_profile_t cc_agi_group_t::profile
```

ccc_pair_count

```
const uint8_t cc_agi_group_t::ccc_pair_count
```

ccc_pairs

```
const ccc_pair_t* cc_agi_group_t::ccc_pairs
```

name

```
const char* cc_agi_group_t::name
```

name_length

```
const uint8_t cc_agi_group_t::name_length
```

cc_agi_config_t

Defines the AGI configuration of one endpoint.

Public Attributes

```
const cc_agi_group_t* groups
const uint8_t group_count
```

Public Attribute Documentation

groups

```
const cc_agi_group_t* cc_agi_config_t::groups
```

group_count

```
const uint8_t cc_agi_config_t::group_count
```

Basic

Basic

Functions

- JOB_STATUS

CC_Basic_Report_tx

(AGI_PROFILE *pProfile, uint8_t sourceEndpoint, uint8_t bValue, ZAF_TX_Callback_t callback)

Send unsolicited command class Basic report.
- zaf_job_status_t

CC_Basic_Set_tx

(agi_profile_t const *const p_profile, const uint8_t source_endpoint, const uint8_t value, const bool use_supervision, ZAF_TX_Callback_t callback)

Sends an unsolicited Basic Set command.

Function Documentation

CC_Basic_Report_tx

JOB_STATUS CC_Basic_Report_tx (AGI_PROFILE * pProfile, uint8_t sourceEndpoint, uint8_t bValue, ZAF_TX_Callback_t callback)

Send unsolicited command class Basic report.

Parameters

Type	Direction	Argument Name	Description
AGI_PROFILE *	[in]	pProfile	pointer to AGI profile
uint8_t	[in]	sourceEndpoint	source endpoint
uint8_t	[in]	bValue	Basic Report value
ZAF_TX_Callback_t	[out]	callback	callback function pointer returning status on job. Can be initialized to NULL.

Returns

- status of the job of type JOB_STATUS

CC_Basic_Set_tx

zaf_job_status_t CC_Basic_Set_tx (agi_profile_t const *const p_profile, const uint8_t source_endpoint, const uint8_t value, const bool use_supervision, ZAF_TX_Callback_t callback)

Sends an unsolicited Basic Set command.

Parameters

Type	Direction	Argument Name	Description
agi_profile_t const *const	[in]	p_profile	Pointer to an AGI profile defined in the configuration of CC AGI.

Type	Direction	Argument Name	Description
const uint8_t	[in]	source_endpoint	Source endpoint if the sending device use endpoints. Otherwise 0.
const uint8_t	[in]	value	Value as defined in the specification of the command.
const bool	[in]	use_supervision	Use Supervision for transmission of this Basic Set.
ZAF_TX_Callback_t	[out]	callback	Callback function pointer giving a result of the transmission.

Returns

- Status on whether the transmission could be initiated.

Battery

Battery

This command class can be used for reporting the device's battery level with additional features.

Additional functionality may be added to the application, if needed. See [CC_Battery_BatteryGet_handler\(\)](#) in `zwave_soc_sensor_pir's` `app.c` source file as an example.

Modules

[SBatteryData](#)

Enumerations

```
enum cc_battery_level_t {  
    CMD_CLASS_BATTERY_LEVEL_LOW = 0x00  
    CMD_CLASS_BATTERY_LEVEL_HIGH = 0x10  
    CMD_CLASS_BATTERY_LEVEL_FULL = 0x64  
    CMD_CLASS_BATTERY_LEVEL_WARNING = 0xFF  
}  
Battery value can range from 0 to 100 (0x00 to 0x64) representing the battery level in percentage from 0 to 100%.
```

Typedefs

```
typedef struct SBatteryData  
SBatteryData  
Used to store the last reported battery level.
```

Functions

```
bool CC_Battery_LevelReport_tx(const AGI_PROFILE *pProfile, uint8_t sourceEndpoint, void(*pCbFunc)  
(TRANSMISSION_RESULT *pTransmissionResult))  
Send unsolicited battery report.  
  
uint8_t CC_Battery_BatteryGet_handler(uint8_t endpoint)  
Get current battery level from application.  
  
bool cc_battery_check_level_changed(void)  
Function for periodically checking if the battery level differs from what was last reported.  
  
uint8_t cc_battery_config_get_reporting_decrements(void)  
Gets the reporting decrements value.  
  
bool cc_battery_write(SBatteryData *battery_data)  
  
bool cc_battery_read(SBatteryData *battery_data)
```

Enumeration Documentation

cc_battery_level_t

cc_battery_level_t

Battery value can range from 0 to 100 (0x00 to 0x64) representing the battery level in percentage from 0 to 100%.

The value 255 (0xFF) is used to indicate low-battery warning. All other values are reserved and SHALL be ignored by the receiving device.

Enumerator	
CMD_CLASS_BATTERY_LEVEL_LOW	battery level low
CMD_CLASS_BATTERY_LEVEL_HIGH	battery level high
CMD_CLASS_BATTERY_LEVEL_FULL	battery level full
CMD_CLASS_BATTERY_LEVEL_WARNING	battery low-level warning

Typedef Documentation

SBatteryData

```
typedef struct SBatteryData SBatteryData
```

Used to store the last reported battery level.

Function Documentation

CC_Battery_LevelReport_tx

```
bool CC_Battery_LevelReport_tx (const AGI_PROFILE * pProfile, uint8_t sourceEndpoint, void(*) (TRANSMISSION_RESULT *pTransmissionResult) pCbFunc)
```

Send unsolicited battery report.

Parameters

Type	Direction	Argument Name	Description
const AGI_PROFILE *	[in]	pProfile	pointer to AGI profile
uint8_t	[in]	sourceEndpoint	source endpoint
void(*) (TRANSMISSION_RESULT *pTransmissionResult)	[out]	pCbFunc	callback funtion returning status destination node receive job.

Returns

- true if a Battery Report was succesfully enqueued for transmission, false otherwise.

CC_Battery_BatteryGet_handler

```
uint8_t CC_Battery_BatteryGet_handler (uint8_t endpoint)
```

Get current battery level from application.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	binary switch endpoint

This function must be implemented in the application code. Called by battery command class handler.

Returns

- current battery level

cc_battery_check_level_changed

```
bool cc_battery_check_level_changed (void )
```

Function for periodically checking if the battery level differs from what was last reported.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- true if the battery level has changed since last reported
- false if the battery level hasn't changed

cc_battery_config_get_reporting_decrements

```
uint8_t cc_battery_config_get_reporting_decrements (void )
```

Gets the reporting decrements value.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- uint8_t reporting decrements value

cc_battery_write

```
bool cc_battery_write (SBatteryData * battery_data)
```

Parameters

Type	Direction	Argument Name	Description
SBatteryData *	N/A	battery_data	

cc_battery_read


```
bool cc_battery_read (SBatteryData * battery_data)
```

Parameters

Type	Direction	Argument Name	Description
SBatteryData *	N/A	battery_data	

SBatteryData

Used to store the last reported battery level.

Public Attributes

uint8_t [lastReportedBatteryLevel](#)

Public Attribute Documentation

lastReportedBatteryLevel

uint8_t SBatteryData::lastReportedBatteryLevel

Binary Switch

Binary Switch

Modules

`cc_binary_switch_t`

Enumerations

```
enum cc_binary_switch_event_t {  
    CC_BINARY_SWITCH_EVENT_START_LEVEL_CHANGE  
    CC_BINARY_SWITCH_EVENT_REACHED_FINAL_VALUE  
}
```

Typedefs

```
typedef void(* cc_binary_switch_callback_t)(struct cc_binary_switch_t *p_switch)  
  
typedef struct cc_binary_switch_t  
    t  
  
typedef enum cc_binary_switch_event_t  
    cc_binary_switch_event_t
```

Functions

```
e_cmd_handler_re  
turn_code_t cc_binary_switch_set_handler(uint8_t value, uint8_t duration, uint8_t endpoint)  
              Handler for mapping Basic Set to Multilevel Switch Set.  
  
void cc_binary_switch_set(cc_binary_switch_t *p_switch, uint8_t value)  
      Sets the value of a given binary switch.  
  
uint8_t cc_binary_switch_get_current_value(cc_binary_switch_t *p_switch)  
        Returns the current value of a given binary switch.  
  
void cc_binary_switch_handler(cc_binary_switch_t *p_switch)  
  
cc_binary_switch_t * cc_binary_switch_get_config(void)  
  
uint8_t cc_binary_switch_get_config_length(void)
```

Macros

```
#define BIN_SWITCH_DURATION_MIN_VAL_SECONDS 0x00  
  
#define BIN_SWITCH_DURATION_MAX_VAL_SECONDS 0x7F  
  
#define BIN_SWITCH_DURATION_MIN_VAL_MINUTES 0x80
```

```
#define BIN_SWITCH_DURATION_MAX_VAL_MINUTES 0xFE

#define BIN_SWITCH_DURATION_FACTORY_DEFAULT 0xFF

#define BIN_SWITCH_DURATION_MAX_MINUTES (BIN_SWITCH_DURATION_MAX_VAL_MINUTES -
BIN_SWITCH_DURATION_MIN_VAL_MINUTES)
```

Enumeration Documentation

cc_binary_switch_event_t

cc_binary_switch_event_t	
Enumerator	
CC_BINARY_SWITCH_EVENT_START_LEVEL_CHANGE	The switch pointed to by the event data has started changing level.
CC_BINARY_SWITCH_EVENT_REACHED_FINAL_VALUE	The switch pointed to by the event data has reached its final value.

Typedef Documentation

cc_binary_switch_callback_t

```
typedef void(* cc_binary_switch_callback_t) (struct cc_binary_switch_t *p_switch) )(struct cc_binary_switch_t *p_switch)
```

cc_binary_switch_t

```
typedef struct cc_binary_switch_t cc_binary_switch_t
```

cc_binary_switch_event_t

```
typedef enum cc_binary_switch_event_t cc_binary_switch_event_t
```

Function Documentation

cc_binary_switch_set_handler

```
e_cmd_handler_return_code_t cc_binary_switch_set_handler (uint8_t value, uint8_t duration, uint8_t endpoint)
```

Handler for mapping Basic Set to Multilevel Switch Set.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	value	Mapped value.
uint8_t	[in]	duration	Mapped duration.

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	Mapped endpoint.

This function MUST NOT be used for anything else than mapping. **Returns**

- Result of the set command. [E_CMD_HANDLER_RETURN_CODE_HANDLED](#) if the actual value was set to the given value right away. [E_CMD_HANDLER_RETURN_CODE_WORKING](#) if the actual value is transitioning to the given value. [E_CMD_HANDLER_RETURN_CODE_FAIL](#) if the parameters were incorrect.

cc_binary_switch_set

```
void cc_binary_switch_set (cc_binary_switch_t * p_switch, uint8_t value)
```

Sets the value of a given binary switch.

Parameters

Type	Direction	Argument Name	Description
cc_binary_switch_t *	N/A	p_switch	The switch of which the value is set.
uint8_t	N/A	value	The value to set.

This function can be used in an application that might have a button.

Must be used for local actuation only.

cc_binary_switch_get_current_value

```
uint8_t cc_binary_switch_get_current_value (cc_binary_switch_t * p_switch)
```

Returns the current value of a given binary switch.

Parameters

Type	Direction	Argument Name	Description
cc_binary_switch_t *	N/A	p_switch	The switch of which the value is returned.

Must be used for local actuation only. **Returns**

- Current value of the given switch.

cc_binary_switch_handler

```
void cc_binary_switch_handler (cc_binary_switch_t * p_switch)
```

Parameters

Type	Direction	Argument Name	Description
cc_binary_switch_t *	N/A	p_switch	

cc_binary_switch_get_config

cc_binary_switch_t * cc_binary_switch_get_config (void)

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- cc_binary_switch_t*

cc_binary_switch_get_config_length

uint8_t cc_binary_switch_get_config_length (void)

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- uint8_t

cc_binary_switch_t

Public Attributes

RECEIVE_OPTION_S_TYPE_EX	rxOpt Contains information required for Supervision and True Status.
uint8_t	endpoint The switch must be tied to an endpoint.
s_Actuator	actuator The ZAF actuator handles transitions between values.
cc_binary_switch_callback_t	callback That will be invoked on changes of the current value.
uint8_t	default_duration
uint8_t	old_value As this is a binary switch, the old value must be kept until the target value is reached (in case of a duration higher than zero).

Public Attribute Documentation

rxOpt

```
RECEIVE_OPTIONS_TYPE_EX cc_binary_switch_t::rxOpt
```

Contains information required for Supervision and True Status.

Must be the first element in this struct because TSE assumes this location.

endpoint

```
uint8_t cc_binary_switch_t::endpoint
```

The switch must be tied to an endpoint.

Must be set to 0 if no endpoints. The endpoint value MUST be located as the second element in this struct as ZAF depends on that for generation of the Node Information Frame.

actuator

```
s_Actuator cc_binary_switch_t::actuator
```

The ZAF actuator handles transitions between values.

callback

```
cc_binary_switch_callback_t cc_binary_switch_t::callback
```

That will be invoked on changes of the current value.

Will only be invoked if no callback is passed to `cc_binary_switch_init()`.

default_duration

```
uint8_t cc_binary_switch_t::default_duration
```

old_value

```
uint8_t cc_binary_switch_t::old_value
```

As this is a binary switch, the old value must be kept until the target value is reached (in case of a duration higher than zero).

CentralScene

CentralScene

Modules

- central_scene_configuration_t
- s_CC_centralScene_data_t
- _cc_central_scene_t

Typedefs

- typedef struct s_CC_centralScene_data_t s_CC_centralScene_data_t
- typedef struct cc_central_scene_t cc_central_scene_t
Structure that describes a single scene.

Functions

- JOB_STATUS cc_central_scene_notification_tx(AGI_PROFILE *pProfile, uint8_t keyAttribute, uint8_t sceneNumber, ZAF_TX_Callback_t pCbFunc)
Initiates the transmission of a "Central Scene Notification" command.
- void cc_central_scene_handle_notification_timer(bool start_timer, uint8_t scene_number)
Start or stop Central Scene Notification timer Used when event CENTRAL_SCENE_NOTIFICATION_KEY_ATTRIBUTES_KEY_HELD_DOWN happens.
- cc_central_scene_t * cc_central_scene_config_get_scenes_attributes(void)
Returns the array of central scenes.
- uint8_t cc_central_scene_config_get_number_of_scenes(void)
Returns number of supported scenes.
- uint8_t cc_central_scene_config_get_attribute_bitmask(uint8_t scene_number)
Returns supported attributes for a given scene.
- uint8_t cc_central_scene_config_get_identical(void)
Gets 'identical' attribute that describes supported key attributes.
- bool cc_central_scene_read(central_scene_configuration_t *pConfig)
- bool cc_central_scene_write(const central_scene_configuration_t *pConfig)
- void cc_central_scene_migrate(void)

Macros

- #define CC_CENTRAL_SCENE_SIZE_OF_SUPPORTED_SCENES_ATTRIBUTES sizeof(((cc_central_scene_t *)0)->scene_attributes)
Size of supported scenes.

Typedef Documentation

s_CC_centralScene_data_t

```
typedef struct s_CC_centralScene_data_t_ s_CC_centralScene_data_t
```

cc_central_scene_t

```
typedef struct _cc_central_scene_t cc_central_scene_t
```

Structure that describes a single scene.

Function Documentation

cc_central_scene_notification_tx

```
JOB_STATUS cc_central_scene_notification_tx (AGI_PROFILE * pProfile, uint8_t keyAttribute, uint8_t sceneNumber, ZAF_TX_Callback_t pCbFunc)
```

Initiates the transmission of a "Central Scene Notification" command.

Parameters

Type	Direction	Argument Name	Description
AGI_PROFILE *	[in]	pProfile	pointer to AGI profile or NULL for lifeline
uint8_t	[in]	keyAttribute	The key event.
uint8_t	[in]	sceneNumber	The scene number.
ZAF_TX_Callback_t	[in]	pCbFunc	Callback function to be called when transmission is done/failed.

Note

- Source Endpoint is always set to Root in this function

Returns

- Status of the job.

cc_central_scene_handle_notification_timer

```
void cc_central_scene_handle_notification_timer (bool start_timer, uint8_t scene_number)
```

Start or stop Central Scene Notification timer Used when event CENTRAL_SCENE_NOTIFICATION_KEY_ATTRIBUTES_KEY_HELD_DOWN happens.

Parameters

Type	Direction	Argument Name	Description
bool	N/A	start_timer	True if timer should be started, false will stop active timer.
uint8_t	N/A	scene_number	Number of active scene.

App will send notifications periodically as long as the event is active. Frequency depends on whether slow refresh is enabled or not.

cc_central_scene_config_get_scenes_attributes

```
cc_central_scene_t * cc_central_scene_config_get_scenes_attributes (void )
```

Returns the array of central scenes.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- If scenes are identical, the array will contain only one element

Returns

- Address of the first element in the central scene array.

cc_central_scene_config_get_number_of_scenes

```
uint8_t cc_central_scene_config_get_number_of_scenes (void )
```

Returns number of supported scenes.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Total amount of scenes supported by the app.

cc_central_scene_config_get_attribute_bitmask

```
uint8_t cc_central_scene_config_get_attribute_bitmask (uint8_t scene_number)
```

Returns supported attributes for a given scene.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	scene_number	Number of scene to get supported attributes for

Note

- This function is used to prepare CENTRAL_SCENE_SUPPORTED_REPORT It is assumption that argument scene_number is higher than 1 only in case when scenes are not identical.

Returns

Supported attributes as bitmask

cc_central_scene_config_get_identical

```
uint8_t cc_central_scene_config_get_identical (void )
```

Gets 'identical' attribute that describes supported key attributes.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if all keys are identical, False if different key attributes supported

cc_central_scene_read

```
bool cc_central_scene_read (central_scene_configuration_t * pConfig)
```

Parameters

Type	Direction	Argument Name	Description
central_scene_configuration_t *	N/A	pConfig	

cc_central_scene_write

```
bool cc_central_scene_write (const central_scene_configuration_t * pConfig)
```

Parameters

Type	Direction	Argument Name	Description
const central_scene_configuration_t *	N/A	pConfig	

cc_central_scene_migrate

```
void cc_central_scene_migrate (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

central_scene_configuration_t

Contains all configuration parameters.

Public Attributes

uint8_t slowRefresh

Public Attribute Documentation

slowRefresh

uint8_t central_scene_configuration_t::slowRefresh

s_CC_centralScene_data_t_

Public Attributes

RECEIVE_OPTION

S_TYPE_EX

rxOptions

rxOptions

Public Attribute Documentation

rxOptions

RECEIVE_OPTIONS_TYPE_EX s_CC_centralScene_data_t_:rxOptions

rxOptions

_cc_central_scene_t

Structure that describes a single scene.

Public Attributes

uint8_t	scene_number
uint8_t	scene_attributes Number of scene.

Public Attribute Documentation

scene_number

uint8_t _cc_central_scene_t::scene_number

scene_attributes

uint8_t _cc_central_scene_t::scene_attributes

Number of scene.

ColorSwitch

ColorSwitch

Modules

[colorComponent_](#)

Enumerations

```
enum EColorComponents\_ {  
    ECOLORCOMPONENT_WARM_WHITE  
    ECOLORCOMPONENT_COLD_WHITE  
    ECOLORCOMPONENT_RED  
    ECOLORCOMPONENT_GREEN  
    ECOLORCOMPONENT_BLUE  
    ECOLORCOMPONENT_AMBER  
    ECOLORCOMPONENT_CYAN  
    ECOLORCOMPONENT_PURPLE  
    ECOLORCOMPONENT_COLOR_UNDEFINED  
}  
Defined Color Components ID.
```

Typedefs

```
typedef enum EColorComponents\_ EColorComponent\_s\_;  
Defined Color Components ID.  
  
typedef struct s\_colorComponent colorComponent\_;  
Structure that holds Color Switch CC related data.
```

Functions

```
void cc\_color\_switch\_refresh\_cb(void)  
Can be used as single cb function for all colors.  
  
void cc\_color\_switch\_cb(s\_colorComponent *colorComponent)  
  
s\_colorComponent * cc\_color\_switch\_get\_colorComponents(void)  
  
uint8_t cc\_color\_switch\_get\_length\_colorComponents(void)  
  
uint8_t cc\_color\_switch\_get\_default\_duration(void)  
  
bool cc\_color\_switch\_write(uint8_t color_component_id, s\_colorComponent *color_component)  
  
bool cc\_color\_switch\_read(uint8_t color_component_id, s\_colorComponent *color_component)
```

Enumeration Documentation

EColorComponents_

EColorComponents_

Defined Color Components ID.

Enumerator	
ECOLORCOMPONENT_WARM_WHITE	ECOLORCOMPONENT_WARM_WHITE.
ECOLORCOMPONENT_COLD_WHITE	ECOLORCOMPONENT_COLD_WHITE.
ECOLORCOMPONENT_RED	ECOLORCOMPONENT_RED.
ECOLORCOMPONENT_GREEN	ECOLORCOMPONENT_GREEN.
ECOLORCOMPONENT_BLUE	ECOLORCOMPONENT_BLUE.
ECOLORCOMPONENT_AMBER	ECOLORCOMPONENT_AMBER.
ECOLORCOMPONENT_CYAN	ECOLORCOMPONENT_CYAN (For 6ch color mixing)
ECOLORCOMPONENT_PURPLE	ECOLORCOMPONENT_PURPLE (For 6ch color mixing)
ECOLORCOMPONENT_COLOR_UNDEFINED	

Typedef Documentation

EColorComponents

typedef enum EColorComponents_ EColorComponents

Defined Color Components ID.

s_colorComponent

typedef struct colorComponent_ s_colorComponent

Structure that holds Color Switch CC related data.

Function Documentation

cc_color_switch_refresh_cb

void cc_color_switch_refresh_cb (void)

Can be used as single cb function for all colors.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Defined in the app. Called when change takes place in driver and app needs to update LED status

cc_color_switch_cb

```
void cc_color_switch_cb (s_colorComponent * colorComponent)
```

Parameters

Type	Direction	Argument Name	Description
s_colorComponent *	N/A	colorComponent	

cc_color_switch_get_colorComponents

```
s_colorComponent * cc_color_switch_get_colorComponents (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_color_switch_get_length_colorComponents

```
uint8_t cc_color_switch_get_length_colorComponents (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_color_switch_get_default_duration

```
uint8_t cc_color_switch_get_default_duration (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_color_switch_write

```
bool cc_color_switch_write (uint8_t color_component_id, s_colorComponent * color_component)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	color_component_id	
s_colorComponent *	N/A	color_component	

cc_color_switch_read

```
bool cc_color_switch_read (uint8_t color_component_id, s_colorComponent * color_component)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	color_component_id	
s_colorComponent *	N/A	color_component	

colorComponent_

Structure that holds Color Switch CC related data.

Public Attributes

RECEIVE_OPTION S_TYPE_EX	rxOpt	saves Rx values, including data about supervision.
EColorComponent s	colorId	Color Component ID.
uint8_t	ep	Endpoint that supports colorId.
s_Actuator	obj	Actuator data.
void(*)	cb	pointer to app callback function that triggers driver changes

Public Attribute Documentation

rxOpt

RECEIVE_OPTIONS_TYPE_EX colorComponent_::rxOpt

saves Rx values, including data about supervision.
(Must be first member - required by TSE)

colorId

EColorComponents colorComponent_::colorId

Color Component ID.

ep

uint8_t colorComponent_::ep

Endpoint that supports colorId.

obj

s_Actuator colorComponent_::obj

Actuator data.

cb

```
void(* colorComponent_::cb) (struct colorComponent_ *color)
```

pointer to app callback function that triggers driver changes

Common

Common

Functions

JOB_STATUS

`cc_engine_multicast_request`(const AGI_PROFILE *pProfile, uint8_t endpoint, CMD_CLASS_GRP *pcmdGrp, uint8_t *pPayload, uint8_t size, uint8_t fSupervisionEnable, void(*pCbFunc)(TRANSMISSION_RESULT *pTransmissionResult))

Initiates the transmission of a multicast request.

Function Documentation

cc_engine_multicast_request

JOB_STATUS

`cc_engine_multicast_request` (const [AGI_PROFILE](#) * pProfile, uint8_t endpoint, [CMD_CLASS_GRP](#) * pcmdGrp, uint8_t * pPayload, uint8_t size, uint8_t fSupervisionEnable, void(*)([TRANSMISSION_RESULT](#) *pTransmissionResult) pCbFunc)

Initiates the transmission of a multicast request.

Parameters

Type	Direction	Argument Name	Description
const AGI_PROFILE *	[in]	pProfile	pointer to AGI profile or NULL for lifeline
uint8_t	[in]	endpoint	source endpoint
CMD_CLASS_GRP *	[in]	pcmdGrp	command class and command
uint8_t *	[in]	pPayload	payload
uint8_t	[in]	size	size of payload
uint8_t	[in]	fSupervisionEnable	enable or not supervision
void(*)(TRANSMISSION_RESULT *pTransmissionResult)	[in]	pCbFunc	Callback function to be called when transmission is done/failed.

Returns

- JOB_STATUS Status of the job.

Configuration

Configuration

Modules

[cc_config_parameter_value_t](#)
[_cc_config_parameter_attributes_t](#)
[cc_config_parameter_metadata_t](#)
[cc_config_parameter_buffer_t](#)
[cc_configuration_t](#)
[cc_configuration_io_interface_t](#)
[_cc_config_parameter_attributes_t.flags](#)

Enumerations

```
enum    cc_config_parameter_size_t {
    CC_CONFIG_PARAMETER_SIZE_8_BIT = 1
    CC_CONFIG_PARAMETER_SIZE_16_BIT = 2
    CC_CONFIG_PARAMETER_SIZE_32_BIT = 4
    CC_CONFIG_PARAMETER_SIZE_NOT_SPECIFIED = 0xFF
}
Defines the size options of a value.
```

```
enum    cc_config_parameter_format_t {
    CC_CONFIG_PARAMETER_FORMAT_SIGNED_INTEGER =
        CONFIGURATION_PROPERTIES_REPORT_FORMAT_SIGNED_INTEGER_V3
    CC_CONFIG_PARAMETER_FORMAT_UNSIGNED_INTEGER =
        CONFIGURATION_PROPERTIES_REPORT_FORMAT_UNSIGNED_INTEGER_V3
    CC_CONFIG_PARAMETER_FORMAT_ENUMERATED =
        CONFIGURATION_PROPERTIES_REPORT_FORMAT_ENUMERATED_V3
    CC_CONFIG_PARAMETER_FORMAT_BIT_FIELD =
        CONFIGURATION_PROPERTIES_REPORT_FORMAT_BIT_FIELD_V3
}
Defines the format options of a value.
```

```
enum    cc_config_configuration_set_return_value {
    CC_CONFIG_RETURN_CODE_NOT_SUPPORTED
    CC_CONFIG_RETURN_CODE_IO_FAIL
    CC_CONFIG_RETURN_CODE_OK
}
Return value of the configuration set process.
```

Typedefs

```
typedef struct    cc_config_parameter_attributes_t
_cc_config_parameter_attributes_t
```

```
typedef struct cc_config_parameter_buffer_t

typedef void * cc_configuration_handle_t
Defines a handle type for the Configuration CC.
```

Functions

```
bool CC_Configuration_SetValue(cc_configuration_handle_t handle, uint16_t number, cc_config_parameter_value_t value)
Sets a configuration parameter with the given number to the given value.

bool cc_configuration_get(uint16_t parameter_number, cc_config_parameter_buffer_t *parameter_buffer)
Initialize the configuration Command Class.

void cc_configuration_set_interface(cc_configuration_io_interface_t const *interface)
Sets new interface set to manipulate non volatile memory to store and read configuration values.

void cc_configuration_set_default_configuration(cc_configuration_t const *configuration)
Sets default configuration in the Command Class.

bool cc_configuration_limit_value(cc_config_parameter_buffer_t const *parameter_buffer, cc_config_parameter_value_t *pNewValue)
Limit a value by the parameter's limits.

const cc_configuration_t * cc_configuration_get_configuration(void)

bool cc_configuration_io_write(zpal_nvm_object_key_t file_id, uint8_t const *data, size_t size)
Write a specific amount data to nvm.

bool cc_configuration_io_read(zpal_nvm_object_key_t file_id, uint8_t *data, size_t size)
Read a specific amount data to nvm.
```

Enumeration Documentation

cc_config_parameter_size_t

```
cc_config_parameter_size_t
```

Defines the size options of a value.

Enumerator	
CC_CONFIG_PARAMETER_SIZE_8_BIT	CC_CONFIG_PARAMETER_SIZE_8_BIT.
CC_CONFIG_PARAMETER_SIZE_16_BIT	CC_CONFIG_PARAMETER_SIZE_16_BIT.
CC_CONFIG_PARAMETER_SIZE_32_BIT	CC_CONFIG_PARAMETER_SIZE_32_BIT.
CC_CONFIG_PARAMETER_SIZE_NOT_SPECIFIED	CC_CONFIG_PARAMETER_SIZE_NOT_SPECIFIED.

cc_config_parameter_format_t

```
cc_config_parameter_format_t
```

Defines the format options of a value.

Enumerator	
CC_CONFIG_PARAMETER_FORMAT_SIGNED_INTEGER	CC_CONFIG_PARAMETER_FORMAT_SIGNED_INTEGER.
CC_CONFIG_PARAMETER_FORMAT_UNSIGNED_INTEGER	CC_CONFIG_PARAMETER_FORMAT_UNSIGNED_INTEGER.
CC_CONFIG_PARAMETER_FORMAT_ENUMERATED	CC_CONFIG_PARAMETER_FORMAT_ENUMERATED.
CC_CONFIG_PARAMETER_FORMAT_BIT_FIELD	CC_CONFIG_PARAMETER_FORMAT_BIT_FIELD.

cc_config_configuration_set_return_value

```
cc_config_configuration_set_return_value
```

Return value of the configuration set process.

Enumerator	
CC_CONFIG_RETURN_CODE_NOT_SUPPORTED	
CC_CONFIG_RETURN_CODE_IO_FAIL	
CC_CONFIG_RETURN_CODE_OK	

Typedef Documentation

cc_config_parameter_attributes_t

```
typedef struct _cc_config_parameter_attributes_t cc_config_parameter_attributes_t
```

cc_config_parameter_buffer_t

```
typedef struct cc_config_parameter_buffer_t cc_config_parameter_buffer_t
```

cc_configuration_handle_t

```
typedef void* cc_configuration_handle_t
```

Defines a handle type for the Configuration CC.

Function Documentation

CC_Configuration_SetValue

```
bool CC_Configuration_SetValue (cc\_configuration\_handle\_t handle, uint16_t number, cc\_config\_parameter\_value\_t value)
```

Sets a configuration parameter with the given number to the given value.

Parameters

Type	Direction	Argument Name	Description
cc_configuration_handle_t	[in]	handle	CC Configuration handle.
uint16_t	[in]	number	The number of the parameter to be set.
cc_config_parameter_value_t	[in]	value	The value that the parameter must be set to.

- Upon invocation CC_Configuration_Set_handler will be called given that the number and value is valid. **Returns**
 - Returns true if the value was set, false otherwise.

cc_configuration_get

```
bool cc_configuration_get (uint16_t parameter_number, cc\_config\_parameter\_buffer\_t * parameter_buffer)
```

Initialize the configuration Command Class.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	parameter_number	number of the parameter.
cc_config_parameter_buffer_t *	[out]	parameter_buffer	pointer to a parameter buffer which will holds the metadata and current value of a parameter

Returns

- Returns true if successfully found and filled parameter buffer, false otherwise

cc_configuration_set_interface

```
void cc_configuration_set_interface (cc\_configuration\_io\_interface\_t const * interface)
```

Sets new interface set to manipulate non volatile memory to store and read configuration values.

Parameters

Type	Direction	Argument Name	Description
cc_configuration_io_interface_t const *	[in]	interface	pointer to the new interface set

cc_configuration_set_default_configuration

```
void cc_configuration_set_default_configuration (cc\_configuration\_t const * configuration)
```

Sets default configuration in the Command Class.

Parameters

Type	Direction	Argument Name	Description
cc_configuration_t const *	[in]	configuration	pointer to a configuration set

cc_configuration_limit_value

```
bool cc_configuration_limit_value (cc_config_parameter_buffer_t const * parameter_buffer, cc_config_parameter_value_t * pNewValue)
```

Limit a value by the parameter's limits.

Parameters

Type	Direction	Argument Name	Description
cc_config_parameter_buffer_t const *	[in]	parameter_buffer	pointer to the parameter buffer which holds the limit values
cc_config_parameter_value_t *	[out]	pNewValue	the new value to be limited

Returns

- Returns true if successfully limited, false otherwise

cc_configuration_get_configuration

```
const cc_configuration_t * cc_configuration_get_configuration (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_configuration_io_write

```
bool cc_configuration_io_write (zpal_nvm_object_key_t file_id, uint8_t const * data, size_t size)
```

Write a specific amount data to nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	file_id	Object key
uint8_t const *	[in]	data	pointer to the data to be written
size_t	[in]	size	amount of data in byte dimension

Returns

- Returns true if successfully wrote to nvm, else false

cc_configuration_io_read

```
bool cc_configuration_io_read (zpal_nvm_object_key_t file_id, uint8_t * data, size_t size)
```

Read a specific amount data to nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	file_id	Object key
uint8_t *	[in]	data	pointer to the buffer where the data will be read into
size_t	[in]	size	wanted amount of data in byte dimension

Returns

- Returns true if successfully read from nvm, else false

cc_config_parameter_value_t

Defines the different types that a value can take, e.g.
uint8_t, int16_t, etc.

Public Attributes

uint32_t	as_uint32
uint16_t	as_uint16
uint8_t	as_uint8
int32_t	as_int32
int16_t	as_int16
int8_t	as_int8
uint8_t	as_uint8_array

Public Attribute Documentation

as_uint32

uint32_t cc_config_parameter_value_t::as_uint32

as_uint16

uint16_t cc_config_parameter_value_t::as_uint16

as_uint8

uint8_t cc_config_parameter_value_t::as_uint8

as_int32

int32_t cc_config_parameter_value_t::as_int32

as_int16

int16_t cc_config_parameter_value_t::as_int16

as_int8

```
int8_t cc_config_parameter_value_t::as_int8
```

as_uint8_array

```
uint8_t cc_config_parameter_value_t::as_uint8_array[4]
```

_cc_config_parameter_attributes_t

Public Attributes

char const *const	name Human readable name.
char const *const	info Human readable info.
const cc_config_param ter_value_t	min_value Minimum value.
const cc_config_param ter_value_t	max_value Maximum value.
cc_config_param ter_size_t	size Size of the parameter in bytes.
const cc_config_param ter_format_t	format Type of the parameter, e.g.: signed, unsigned.
const cc_config_param ter_value_t	default_value Default data.
const struct _cc_config_param eter_attributes_t:: @0	flags

Public Attribute Documentation

name

char const* const _cc_config_parameter_attributes_t::name

Human readable name.

info

char const* const _cc_config_parameter_attributes_t::info

Human readable info.

min_value

const cc_config_parameter_value_t _cc_config_parameter_attributes_t::min_value

Minimum value.

max_value

```
const cc_config_parameter_value_t _cc_config_parameter_attributes_t::max_value
```

Maximum value.

size

```
cc_config_parameter_size_t _cc_config_parameter_attributes_t::size
```

Size of the parameter in bytes.

format

```
const cc_config_parameter_format_t _cc_config_parameter_attributes_t::format
```

Type of the parameter, e.g.: signed, unsigned.

default_value

```
const cc_config_parameter_value_t _cc_config_parameter_attributes_t::default_value
```

Default data.

flags

```
const struct _cc_config_parameter_attributes_t::@0 _cc_config_parameter_attributes_t::flags
```


cc_config_parameter_metadata_t

Public Attributes

<code>const uint16_t</code>	<code>number</code>	Unique id number of the parameter.
<code>uint16_t</code>	<code>next_number</code>	Next parameter number.
<code>const zpal_nvm_object_key_t</code>	<code>file_id</code>	ID of the configuration file.
<code>cc_config_parameter_attributes_t</code>	<code>attributes</code>	Attributes of the parameters.
<code>bool(*)</code>	<code>migration_handler</code>	Function pointer to handle the migration between versions.

Public Attribute Documentation

number

```
const uint16_t cc_config_parameter_metadata_t::number
```

Unique id number of the parameter.

next_number

```
uint16_t cc_config_parameter_metadata_t::next_number
```

Next parameter number.

file_id

```
const zpal_nvm_object_key_t cc_config_parameter_metadata_t::file_id
```

ID of the configuration file.

attributes

```
cc_config_parameter_attributes_t cc_config_parameter_metadata_t::attributes
```

Attributes of the parameters.

migration_handler

```
bool(* cc_config_parameter_metadata_t::migration_handler) (cc_config_parameter_buffer_t *parameter_buffer)
```

Function pointer to handle the migration between versions.

cc_config_parameter_buffer_t

Public Attributes

```
const metadata
cc_config_parameter_metadata_t *

cc_config_parameter_value_t data_buffer
```

Public Attribute Documentation

metadata

```
const cc_config_parameter_metadata_t* cc_config_parameter_buffer_t::metadata
```

data_buffer

```
cc_config_parameter_value_t cc_config_parameter_buffer_t::data_buffer
```

cc_configuration_t

Defines all data related to the Configuration CC.

Public Attributes

```
uint16_t numberOfParameters

const cc_config_parameter_metadata_t* parameters
```

Public Attribute Documentation

numberOfParameters

```
uint16_t cc_configuration_t::numberOfParameters
```

parameters

```
const cc_config_parameter_metadata_t* cc_configuration_t::parameters
```

cc_configuration_io_interface_t

Holds a write and a read interface defined by the user.

Public Attributes

`bool(*` [write_handler](#)
Write function which writes data with a specified size to non volatile memory.

`bool(*` [read_handler](#)
Read function which reads data with specified size from non volatile memory.

Public Attribute Documentation

write_handler

```
bool(* cc_configuration_io_interface_t::write_handler) (zpal_nvm_object_key_t file_id, uint8_t const *data, size_t size)
```

Write function which writes data with a specified size to non volatile memory.

read_handler

```
bool(* cc_configuration_io_interface_t::read_handler) (zpal_nvm_object_key_t file_id, uint8_t *data, size_t size)
```

Read function which reads data with specified size from non volatile memory.

_cc_config_parameter_attributes_t.flags

Public Attributes

- bool [altering_capabilities](#)
- bool [read_only](#)
- bool [advanced](#)

Public Attribute Documentation

altering_capabilities

read_only

advanced

DeviceResetLocally

DeviceResetLocally

Functions

```
void CC_DeviceResetLocally_done(TRANSMISSION_RESULT *pTransmissionResult)
    Transmission callback for CC_DeviceResetLocally_notification_tx().

void CC_DeviceResetLocally_notification_tx()
    Transmits a Device Reset Locally Notification.
```

Function Documentation

CC_DeviceResetLocally_done

```
void CC_DeviceResetLocally_done (TRANSMISSION_RESULT * pTransmissionResult)
```

Transmission callback for [CC_DeviceResetLocally_notification_tx\(\)](#).

Parameters

Type	Direction	Argument Name	Description
TRANSMISSION_RESULT *	N/A	pTransmissionResult	Result of each transmission.

CC_DeviceResetLocally_notification_tx

```
void CC_DeviceResetLocally_notification_tx ()
```

Transmits a Device Reset Locally Notification.

DoorLock

DoorLock

Modules

[cc_door_lock_capabilities_report_t](#)

[cc_door_lock_configuration_t](#)

[cc_door_lock_data_t](#)

[_door_lock_hw_data_t](#)

[DoorLockEvents](#)

Enumerations

```
enum door\_lock\_mode\_t {
    DOOR_MODE_UNSECURE = DOOR_LOCK_OPERATION_SET_DOOR_UNSECURED_V2
    DOOR_MODE_UNSECURE_TIMEOUT =
    DOOR_LOCK_OPERATION_SET_DOOR_UNSECURED_WITH_TIMEOUT_V2
    DOOR_MODE_UNSECURE_INSIDE =
    DOOR_LOCK_OPERATION_SET_DOOR_UNSECURED_FOR_INSIDE_DOOR_HANDLES_V2
    DOOR_MODE_UNSECURE_INSIDE_TIMEOUT =
    DOOR_LOCK_OPERATION_SET_DOOR_UNSECURED_FOR_INSIDE_DOOR_HANDLES_WITH_TIMEOUT_V2
    DOOR_MODE_UNSECURE_OUTSIDE =
    DOOR_LOCK_OPERATION_SET_DOOR_UNSECURED_FOR_OUTSIDE_DOOR_HANDLES_V2
    DOOR_MODE_UNSECURE_OUTSIDE_TIMEOUT =
    DOOR_LOCK_OPERATION_SET_DOOR_UNSECURED_FOR_OUTSIDE_DOOR_HANDLES_WITH_TIMEOUT_V2
    DOOR_MODE_UNKNOWN = DOOR_LOCK_OPERATION_REPORT_DOOR_MODE_UNKNOWN_V4
    DOOR_MODE_SECURED = DOOR_LOCK_OPERATION_SET_DOOR_SECURED_V2
}
Door Lock Mode (8 bit) will set the door lock device in unsecured or secured mode as well as other peripheral settings.
```

```
enum cc\_door\_lock\_operation\_type\_t {
    DOOR_OPERATION_CONST = 0x01
    DOOR_OPERATION_TIMED = 0x02
    DOOR_OPERATION_RESERVED = 0x03
}
Operation mode (1byte).
```

```
enum cc\_door\_lock\_door\_component\_t {
    DOOR_COMPONENT_DOOR = 0x01
    DOOR_COMPONENT_BOLT = 0x02
    DOOR_COMPONENT_LATCH = 0x04
}
Supported Door Components Bitmask of supported door components.
```

```
enum cc\_door\_lock\_handle\_t {
    DOOR_HANDLE_DISABLED = 0x00
    DOOR_HANDLE_1 = 0x01
    DOOR_HANDLE_2 = 0x02
    DOOR_HANDLE_3 = 0x04
    DOOR_HANDLE_4 = 0x08
}
Supported Door Handles Bitmask of supported door handles.
```


Typedefs

<code>typedef struct _door_lock_hw_data_t</code>	<code>door_lock_hw_data_t</code> Physical components of Door Lock.

Functions

<code>void</code>	<code>cc_door_lock_latch_set</code> (bool opened) Latch status handler.
<code>void</code>	<code>cc_door_lock_handle_set</code> (bool opened)
<code>void</code>	<code>cc_door_lock_bolt_set</code> (bool locked) Bolt status handler.
<code>void</code>	<code>CC_DoorLock_SetOutsideDoorHandleState</code> (cc_door_lock_handle_t handle) Sets the Outside Door Handle State for a specific handle.
<code>void</code>	<code>CC_DoorLock_ClearOutsideDoorHandleState</code> (cc_door_lock_handle_t handle) Clears the Outside Door Handle State for a specific handle.
<code>uint8_t</code>	<code>cc_door_lock_mode_hw_change</code> (door_lock_mode_t mode) Initiates change on the hardware that changes door lock mode.
<code>bool</code>	<code>door_lock_hw_bolt_is_unlocked</code> (void) Returns bolt state.
<code>bool</code>	<code>door_lock_hw_latch_is_closed</code> (void)
<code>bool</code>	<code>door_lock_hw_handle_is_pressed</code> (void)
<code>cc_door_lock_operation_type_t</code>	<code>cc_door_lock_get_operation_type</code> (void)
<code>cc_door_lock_handle_t</code>	<code>cc_door_lock_get_supported_inside_handles</code> (void)
<code>cc_door_lock_handle_t</code>	<code>cc_door_lock_get_supported_outside_handles</code> (void)
<code>uint8_t</code>	<code>cc_door_lock_get_options_flags</code> (void)
<code>uint16_t</code>	<code>cc_door_lock_get_max_auto_relock_time</code> (void)
<code>uint16_t</code>	<code>cc_door_lock_get_max_hold_and_release_time</code> (void)
<code>bool</code>	<code>cc_door_lock_write</code> (cc_door_lock_data_t *door_lock_data)
<code>bool</code>	<code>cc_door_lock_read</code> (cc_door_lock_data_t *door_lock_data)
<code>void</code>	<code>cc_door_lock_migrate</code> (void)

Macros

<code>#define</code>	<code>DOOR_LOCK_OPERATION_SET_TIMEOUT_NOT_SUPPORTED</code> 0xFE Disabling Lock timeout minutes and seconds.

Enumeration Documentation

door_lock_mode_t

door_lock_mode_t

Door Lock Mode (8 bit) will set the door lock device in unsecured or secured mode as well as other peripheral settings.

1) Constant mode. Door will be unsecured until set back to secured mode by Command. 2) Timeout mode. Fallback to secured mode after timeout has expired (set by Door Lock Configuration Set). 3) This is Read Only State, i.e. Bolt is not fully retracted/engaged

Enumerator	
DOOR_MODE_UNSECURE	Door Unsecured 1)
DOOR_MODE_UNSECURE_TIMEOUT	Door Unsecured with timeout 2)
DOOR_MODE_UNSECURE_INSIDE	Door Unsecured for inside Door Handles 1)
DOOR_MODE_UNSECURE_INSIDE_TIMEOUT	Door Unsecured for inside Door Handles with timeout 2)
DOOR_MODE_UNSECURE_OUTSIDE	Door Unsecured for outside Door Handles 1)
DOOR_MODE_UNSECURE_OUTSIDE_TIMEOUT	Door Unsecured for outside Door Handles with timeout 2)
DOOR_MODE_UNKNOWN	Door/Lock State Unknown (Version 4) this is an invalid set value.
DOOR_MODE_SECURED	Door Secured.

cc_door_lock_operation_type_t

cc_door_lock_operation_type_t

Operation mode (1byte).

The Operation Type field can be set to either constant or timed operation. When timed operation is set, the Lock Timer Minutes and Lock Timer Seconds fields MUST be set to valid values.

Enumerator	
DOOR_OPERATION_CONST	Constant operation.
DOOR_OPERATION_TIMED	Timed operation.
DOOR_OPERATION_RESERVED	0X03..0XFF Reserved

cc_door_lock_door_component_t

cc_door_lock_door_component_t

Supported Door Components Bitmask of supported door components.

Set to 1 if corresponding component is supported.

Enumerator	
DOOR_COMPONENT_DOOR	
DOOR_COMPONENT_BOLT	
DOOR_COMPONENT_LATCH	

cc_door_lock_handle_t

```
cc_door_lock_handle_t
```

Supported Door Handles Bitmask of supported door handles.

Set to 1 if corresponding handle is supported.

Enumerator

DOOR_HANDLE_DISABLED	
DOOR_HANDLE_1	
DOOR_HANDLE_2	
DOOR_HANDLE_3	
DOOR_HANDLE_4	

Typedef Documentation

door_lock_hw_data_t

```
typedef struct _door_lock_hw_data_t door_lock_hw_data_t
```

Physical components of Door Lock.

Structure follows but it's not limited to requirement CC:0062.04.03.11.006 Structure can be expanded with additional HW components.

Function Documentation

cc_door_lock_latch_set

```
void cc_door_lock_latch_set (bool opened)
```

Latch status handler.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	opened	Latch status. True if latch is opened, false if closed.

Called when latch status is changed.

cc_door_lock_handle_set

```
void cc_door_lock_handle_set (bool opened)
```

Parameters

Type	Direction	Argument Name	Description
bool	N/A	opened	

cc_door_lock_bolt_set

```
void cc_door_lock_bolt_set (bool locked)
```

Bolt status handler.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	locked	Bolt status. True if bolt is locked, false if unlocked.

Called when bolt status is changed.

CC_DoorLock_SetOutsideDoorHandleState

```
void CC_DoorLock_SetOutsideDoorHandleState (cc_door_lock_handle_t handle)
```

Sets the Outside Door Handle State for a specific handle.

Parameters

Type	Direction	Argument Name	Description
cc_door_lock_handle_t	N/A	handle	handle to be set

CC_DoorLock_ClearOutsideDoorHandleState

```
void CC_DoorLock_ClearOutsideDoorHandleState (cc_door_lock_handle_t handle)
```

Clears the Outside Door Handle State for a specific handle.

Parameters

Type	Direction	Argument Name	Description
cc_door_lock_handle_t	N/A	handle	handle to be cleared

cc_door_lock_mode_hw_change

```
uint8_t cc_door_lock_mode_hw_change (door_lock_mode_t mode)
```

Initiates change on the hardware that changes door lock mode.

Parameters

Type	Direction	Argument Name	Description
door_lock_mode_t	N/A	mode	Door Lock Mode to be set

Can be called after operation has been triggered locally, or valid Operation Set command was received

Returns

- Remaining duration in seconds

door_lock_hw_bolt_is_unlocked

bool door_lock_hw_bolt_is_unlocked (void)

Returns bolt state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if unlocked, false if locked

door_lock_hw_latch_is_closed

bool door_lock_hw_latch_is_closed (void)

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

-

door_lock_hw_handle_is_pressed

bool door_lock_hw_handle_is_pressed (void)

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

-

cc_door_lock_get_operation_type

cc_door_lock_operation_type_t cc_door_lock_get_operation_type (void)

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_door_lock_get_supported_inside_handles

```
cc_door_lock_handle_t cc_door_lock_get_supported_inside_handles (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_door_lock_get_supported_outside_handles

```
cc_door_lock_handle_t cc_door_lock_get_supported_outside_handles (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_door_lock_get_options_flags

```
uint8_t cc_door_lock_get_options_flags (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_door_lock_get_max_auto_relock_time

```
uint16_t cc_door_lock_get_max_auto_relock_time (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_door_lock_get_max_hold_and_release_time

```
uint16_t cc_door_lock_get_max_hold_and_release_time (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_door_lock_write

```
bool cc_door_lock_write (cc_door_lock_data_t * door_lock_data)
```

Parameters

Type	Direction	Argument Name	Description
cc_door_lock_data_t *	N/A	door_lock_data	

cc_door_lock_read

```
bool cc_door_lock_read (cc_door_lock_data_t * door_lock_data)
```

Parameters

Type	Direction	Argument Name	Description
cc_door_lock_data_t *	N/A	door_lock_data	

cc_door_lock_migrate

```
void cc_door_lock_migrate (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_door_lock_capabilities_report_t

Door Lock Capabilities Report data structure.

This allows the Application to inform the Command Class handler of the capabilities of the Door Lock.

The [supportedOperationTypeBitmask](#) field indicates the Supported Operations Modes:

- Bit 0 in Bit Mask 1: Reserved
- Bit 1 in Bit Mask 1: Constant Operation
- Bit 2 in Bit Mask 1: Timed Operation.

The [supportedDoorLockModeList](#) indicates the Supported Door Lock Modes:

- 0x00: Door Unsecured
- 0x01: Door unsecured with timeout
- 0x10: Door Unsecured for Inside Door Handles
- 0x11: Door Unsecured for Inside Door Handles with timeout
- 0x20: Door Unsecured for Outside Door Handles
- 0x21: Door Unsecured for Outside Door Handles with timeout
- 0xFF: Door Secured

Inside/outside Door Handles Modes supported These mode bits indicate which handles are supported on the Door Lock.

The mode of the four inside and four outside door handles are encoded in [supportedInsideHandleModes](#) and [supportedOutsideHandleModes](#) as:

- Bit 0: Handle 1
- Bit 1: Handle 2
- Bit 2: Handle 3
- Bit 3: Handle 4 Values:
- 0: disabled
- 1: enabled

Supported Door Components. The [supportedDoorComponents](#) field indicates the supported Door Lock components: such as bolt and latch states.

- Bit 0: Door
- Bit 1: Bolt
- Bit 2: Latch
- Bit 3-7: Reserved

Auto Relock Support The [autoRelockSupport](#) flag indicates if the Door Lock Supports Auto Relock (1=supported, 0=not supported).

Hold And Release Support The [holdAndReleaseSupport](#) flag indicates if the Door Lock Supports * Hold And Release (1=supported, 0=not supported),

Twist Assist Support The [twistAssistSupport](#) flag indicates if the Door Lock Supports Twist Assist (1=supported, 0=not supported),

Block To Block Support The [blockToBlockSupport](#) flag indicates if the Door Lock Supports Block To Block (1=supported, 0=not supported),

Public Attributes

uint8_t

[reserved](#)
Reserved (2 bits)

uint8_t	lengthSupportedOperationType Length of Supported Operation Type Bitmask (5 bits)
uint8_t	supportedOperationTypeBitmask Supported Operation Types.
uint8_t	lengthSupportedDoorLockModeList Length of supported Door Lock Mode List.
uint8_t	supportedDoorLockModeList List of Supported Door Lock Modes.
uint8_t	supportedOutsideHandleModes Supported Outside Handle Modes bitmask (4 bits)
uint8_t	supportedInsideHandleModes Supported Inside Handle Modes bitmask (4 bits)
uint8_t	supportedDoorComponents Supported Door Components.
uint8_t	autoRelockSupport Indicates if the lock supports Auto Relock.
uint8_t	holdAndReleaseSupport Indicates if the lock supports Hold And Release.
uint8_t	twistAssistSupport Indicates if the lock supports Twist Assist.
uint8_t	blockToBlockSupport Indicates if the lock supports Twist Assist.

Public Attribute Documentation

reserved

uint8_t cc_door_lock_capabilities_report_t::reserved

Reserved (2 bits)

lengthSupportedOperationType

uint8_t cc_door_lock_capabilities_report_t::lengthSupportedOperationType

Length of Supported Operation Type Bitmask (5 bits)

supportedOperationTypeBitmask

uint8_t cc_door_lock_capabilities_report_t::supportedOperationTypeBitmask

Supported Operation Types.

Max. length fixed to 1 byte. Enough to hold both defined types defined in SDS13781.

lengthSupportedDoorLockModeList

```
uint8_t cc_door_lock_capabilities_report_t::lengthSupportedDoorLockModeList
```

Length of supported Door Lock Mode List.

supportedDoorLockModeList

```
uint8_t cc_door_lock_capabilities_report_t::supportedDoorLockModeList[8]
```

List of Supported Door Lock Modes.
Max. Length fixed to 8 byte. Enough to hold all modes defined in SDS13781.

supportedOutsideHandleModes

```
uint8_t cc_door_lock_capabilities_report_t::supportedOutsideHandleModes
```

Supported Outside Handle Modes bitmask (4 bits)

supportedInsideHandleModes

```
uint8_t cc_door_lock_capabilities_report_t::supportedInsideHandleModes
```

Supported Inside Handle Modes bitmask (4 bits)

supportedDoorComponents

```
uint8_t cc_door_lock_capabilities_report_t::supportedDoorComponents
```

Supported Door Components.

autoRelockSupport

```
uint8_t cc_door_lock_capabilities_report_t::autoRelockSupport
```

Indicates if the lock supports Auto Relock.

holdAndReleaseSupport

```
uint8_t cc_door_lock_capabilities_report_t::holdAndReleaseSupport
```

Indicates if the lock supports Hold And Release.

twistAssistSupport

```
uint8_t cc_door_lock_capabilities_report_t::twistAssistSupport
```

Indicates if the lock supports Twist Assist.

blockToBlockSupport

```
uint8_t cc_door_lock_capabilities_report_t::blockToBlockSupport
```

Indicates if the lock supports Twist Assist.

cc_door_lock_configuration_t

Door Lock Configuration Get/Set Structure

Inside/outside Door Handles Mode These fields are used to configure if each individual outside and inside door handle can open the door locally.

The mode of the four inside and four outside door handles are encoded in [insideDoorHandleMode](#) and [outsideDoorHandleMode](#) as:

- Bit 0: Handle 1
- Bit 1: Handle 2
- Bit 2: Handle 3
- Bit 3: Handle 4 Values:
- 0: disabled
- 1: enabled

Public Attributes

uint8_t	type
uint8_t	insideDoorHandleMode Operation mode - cc_door_lock_operation_type_t .
uint8_t	outsideDoorHandleMode Outside Door Handles Mode (4 bits)
uint8_t	lockTimeoutMin Lock Timeout Minutes, valid values 1-254 decimal.
uint8_t	lockTimeoutSec Lock Timeout Seconds, valid 1-59 decimal.
uint8_t	autoRelockTime1 Auto-relock time in seconds MSB.
uint8_t	autoRelockTime2 Auto-relock time in seconds LSB.
uint8_t	holdAndReleaseTime1 Hold and release time in seconds MSB.
uint8_t	holdAndReleaseTime2 Hold and release time in seconds LSB.
uint8_t	reservedOptionsFlags Bit 0: Twist assist, bit 1: Block to block, others reserved.

Public Attribute Documentation

type

```
uint8_t cc_door_lock_configuration_t::type
```

insideDoorHandleMode

```
uint8_t cc_door_lock_configuration_t::insideDoorHandleMode
```

Operation mode - [cc_door_lock_operation_type_t](#).

Inside Door Handles Mode (4 bits)

outsideDoorHandleMode

```
uint8_t cc_door_lock_configuration_t::outsideDoorHandleMode
```

Outside Door Handles Mode (4 bits)

lockTimeoutMin

```
uint8_t cc_door_lock_configuration_t::lockTimeoutMin
```

Lock Timeout Minutes, valid values 1-254 decimal.

lockTimeoutSec

```
uint8_t cc_door_lock_configuration_t::lockTimeoutSec
```

Lock Timeout Seconds, valid 1-59 decimal.

autoRelockTime1

```
uint8_t cc_door_lock_configuration_t::autoRelockTime1
```

Auto-relock time in seconds MSB.

autoRelockTime2

```
uint8_t cc_door_lock_configuration_t::autoRelockTime2
```

Auto-relock time in seconds LSB.

holdAndReleaseTime1

```
uint8_t cc_door_lock_configuration_t::holdAndReleaseTime1
```

Hold and release time in seconds MSB.

holdAndReleaseTime2

uint8_t cc_door_lock_configuration_t::holdAndReleaseTime2

Hold and release time in seconds LSB.

reservedOptionsFlags

uint8_t cc_door_lock_configuration_t::reservedOptionsFlags

Bit 0: Twist assist, bit 1: Block to block, others reserved.

cc_door_lock_data_t

Cmd Class Door Lock data structure.

- Inside/outside Door Handles Mode These mode bits indicate if the actual handle can open the door locally. The mode of the four inside and four outside door handles are encoded in [insideDoorHandleMode](#) and [outsideDoorHandleMode](#) as:
 - Bit 0: Handle 1
 - Bit 1: Handle 2
 - Bit 2: Handle 3
 - Bit 3: Handle 4
 - Values:
 - 0: disabled
 - 1: enabled
- Inside/outside Door Handles State These fields indicate if a door handle has been activated. The state of the four inside and four outside door handles are encoded in [insideDoorHandleState](#) and [outsideDoorHandleState](#) as follows:
 - Bit 0: Handle 1
 - Bit 1: Handle 2
 - Bit 2: Handle 3
 - Bit 3: Handle 4
 - Values:
 - 0: inactive
 - 1: activated

Note

- Field [condition](#) is used only for saving data to NVM. For needs of Operation Report, this bitmask is calculated based on actual latch/bolt/door values. It's recommended to use [door_lock_hw_data_t](#) for hardware data.

Door condition. The [condition](#) field indicates the status of the door lock components such as bolt and latch states.

- Bit 0: Door
 - 0: Open
 - 1: Closed
- Bit 1: Bolt
 - 0: Locked
 - 1: Unlocked
- Bit 2: Latch
 - 0: Open
 - 1: Closed
- Bit 3-7: Reserved

Public Attributes

door_lock_mode_t	mode
cc_door_lock_operation_type_t	type
uint8_t	insideDoorHandleMode Inside Door Handles Mode (4 bits)
uint8_t	outsideDoorHandleMode Outside Door Handles Mode (4 bits)
uint8_t	insideDoorHandleState Inside Door Handles State (4 bits)

uint8_t	outsideDoorHandleState	Outside Door Handles State (4 bits)
uint8_t	condition	Door condition (8 bits)
uint8_t	lockTimeoutMin	Lock Timeout Minutes, valid values 1-254 decimal.
uint8_t	lockTimeoutSec	Lock Timeout Seconds, valid 1-59 decimal.

Public Attribute Documentation

mode

```
door_lock_mode_t cc_door_lock_data_t::mode
```

type

```
cc_door_lock_operation_type_t cc_door_lock_data_t::type
```

insideDoorHandleMode

```
uint8_t cc_door_lock_data_t::insideDoorHandleMode
```

Inside Door Handles Mode (4 bits)

outsideDoorHandleMode

```
uint8_t cc_door_lock_data_t::outsideDoorHandleMode
```

Outside Door Handles Mode (4 bits)

insideDoorHandleState

```
uint8_t cc_door_lock_data_t::insideDoorHandleState
```

Inside Door Handles State (4 bits)

outsideDoorHandleState

```
uint8_t cc_door_lock_data_t::outsideDoorHandleState
```

Outside Door Handles State (4 bits)

condition

uint8_t cc_door_lock_data_t::condition

Door condition (8 bits)

lockTimeoutMin

uint8_t cc_door_lock_data_t::lockTimeoutMin

Lock Timeout Minutes, valid values 1-254 decimal.

lockTimeoutSec

uint8_t cc_door_lock_data_t::lockTimeoutSec

Lock Timeout Seconds, valid 1-59 decimal.

_door_lock_hw_data_t

Physical components of Door Lock.

Structure follows but it's not limited to requirement CC:0062.04.03.11.006 Structure can be expanded with additional HW components.

Public Attributes

bool	bolt_unlocked	
bool	latch_closed	True for unlocked, false for locked.
bool	handle_pressed	True for Closed, false for Open.

Public Attribute Documentation

bolt_unlocked

bool _door_lock_hw_data_t::bolt_unlocked

latch_closed

bool _door_lock_hw_data_t::latch_closed

True for unlocked, false for locked.

handle_pressed

bool _door_lock_hw_data_t::handle_pressed

True for Closed, false for Open.

DoorLockEvents

DoorLockEvents

Macros

```
#define CC_DOOR_LOCK_CODE_EVENT_TOGGLE 1

#define CC_DOOR_LOCK_EVENT_OPERATION_SET_DONE 2

#define CC_DOOR_LOCK_EVENT_HW_OPERATION_DONE 3
```

Firmware Update Meta Data

Firmware Update Meta Data

Command Class Firmware Update Meta Data enables a device to have its firmware updated remotely.

CC Firmware Update MD can be configured in [cc_firmware_update_config.h](#).

This CC offers the option to define two functions in the application that will override the weak implementations:

- [CC_FirmwareUpdate_SetStartCallback\(\)](#)
- [CC_FirmwareUpdate_SetFinishCallback\(\)](#)

Note

- **Important:** Performing an OTA update is not possible in debug builds due to memory and storage constraints. Debug builds have reduced bootloader storage space and additional debug information that prevents successful OTA operations. Always use release builds for OTA update testing and deployment.
- **Troubleshooting:** When troubleshooting OTA updates in release mode, the `zw_log` component is highly recommended. It provides comprehensive logging capabilities that can help identify issues during the OTA process, including transfer failures, CRC errors, and bootloader problems. Enable the `zw_log` component and configure appropriate log levels to capture detailed OTA-related debug information.

The specification of CC Firmware Update MD can be found in https://github.com/Z-Wave-Alliance/AWG/tree/main/source/management_command_classes/command_class_definitions

Enumerations

```
enum    OTA_STATUS {
        OTA_STATUS_DONE = 0
        OTA_STATUS_ABORT = 1
        OTA_STATUS_TIMEOUT = 2
    }
enum type OTA_STATUS use to
```

Typedefs

```
typedef bool(*    CC_FirmwareUpdate_start_callback_t)(uint16_t fwld, uint16_t CRC)
Please see description of CC_FirmwareUpdate_Init().

typedef void(*    CC_FirmwareUpdate_finish_callback_t)(OTA_STATUS status)
Please see description of CC_FirmwareUpdate_Init().
```

Functions

```
bool    CC_FirmwareUpdate_SetStartCallback(uint16_t fwld, uint16_t CRC)
Invoked when another node initiates a firmware update of this device.

void    CC_FirmwareUpdate_SetFinishCallback(OTA_STATUS pOtaFinish)
Invoked when the firmware update process finishes either successfully or with an error.
```

bool	CC_FirmwareUpdate_Init (CC_FirmwareUpdate_start_callback_t pOtaStart, CC_FirmwareUpdate_finish_callback_t pOtaFinish, bool support_activation) Initializes the Firmware Update Meta Data Command Class.
void	ZCB_CmdClassFwUpdateMdReqReport (transmission_result_t *pTxResult) ZCB_CmdClassFwUpdateMdReqReport Callback function receive status on Send data FIRMWARE_UPDATE_MD_REQUEST_REPORT_V3.
bool	CC_FirmwareUpdate_ActivationSet_handler (ZW_FIRMWARE_UPDATE_ACTIVATION_SET_V5_FRAME *pFrame, uint8_t *pStatus) Handles an Activation Set command.
uint16_t	handleCommandClassFirmwareUpdateMaxFragmentSize (void) Returns the maximum fragment size.
uint8_t	getFWUpdateMDGetNumberOfReports (void) Gets maximum number of reports that can be sent in FW Update MD Get.
uint16_t	handleFirmWareIdGetExtended (uint8_t n) This function returns a firmware ID based on a given target index.
void	handleCmdClassFirmwareUpdateMdReqGet (RECEIVE_OPTIONS_TYPE_EX *rxOpt, ZW_FIRMWARE_UPDATE_MD_REQUEST_GET_V5_FRAME *pFrame, uint8_t cmdLength, uint8_t *pStatus) Initiates a firmware update.
void	handleCmdClassFirmwareUpdateMdReport (uint16_t crc16Result, uint16_t firmwareUpdateReportNumber, uint8_t properties, uint8_t *pData, uint8_t fw_actualFrameSize) handleCmdClassFirmwareUpdateMdReport Application function to handle incoming frame Firmware update MD Report

Macros

#define	WAITTIME_FWU_FAIL 2 Defines for WaitTime field used in command = FIRMWARE_UPDATE_MD_STATUS_REPORT.

Enumeration Documentation

OTA_STATUS

OTA_STATUS	
enum type OTA_STATUS use to	
Enumerator	
OTA_STATUS_DONE	
OTA_STATUS_ABORT	
OTA_STATUS_TIMEOUT	

Typedef Documentation

CC_FirmwareUpdate_start_callback_t

typedef bool(* CC_FirmwareUpdate_start_callback_t) (uint16_t fwld, uint16_t CRC))(uint16_t fwld, uint16_t CRC)

Please see description of CC_FirmwareUpdate_Init().

CC_FirmwareUpdate_finish_callback_t

```
typedef void(* CC_FirmwareUpdate_finish_callback_t) (OTA_STATUS status) )(OTA_STATUS status)
```

Please see description of CC_FirmwareUpdate_Init().

Function Documentation

CC_FirmwareUpdate_SetStartCallback

```
bool CC_FirmwareUpdate_SetStartCallback (uint16_t fwId, uint16_t CRC)
```

Invoked when another node initiates a firmware update of this device.

Parameters

Type	Direction	Argument Name	Description
uint16_t	N/A	fwId	ID of the firmware.
uint16_t	N/A	CRC	CRC of the firmware.

If the function returns true, the process will continue. If the function returns false, the node will report to the initiator that the firmware update requires authentication.

The function must take two arguments where the first one is the firmware ID and the second one is the CRC value of the firmware.

Note

- This function is weakly defined to return true, but can be defined by the application if desired.

Returns

- Return true to continue the firmware update process and false to cancel it.

CC_FirmwareUpdate_SetFinishCallback

```
void CC_FirmwareUpdate_SetFinishCallback (OTA\_STATUS pOtaFinish)
```

Invoked when the firmware update process finishes either successfully or with an error.

Parameters

Type	Direction	Argument Name	Description
OTA_STATUS	N/A	pOtaFinish	Status of the firmware update.

If the device reboots right after the firmware update, the function is invoked before the reboot.

The function must take an argument that is the status of the firmware update.

Note

- This function is weakly defined to do nothing and can be defined by the application if desired.

CC_FirmwareUpdate_Init

```
bool CC_FirmwareUpdate_Init (CC_FirmwareUpdate_start_callback_t pOtaStart, CC_FirmwareUpdate_finish_callback_t pOtaFinish, bool support_activation)
```

Initializes the Firmware Update Meta Data Command Class.

Parameters

Type	Direction	Argument Name	Description
CC_FirmwareUpdate_start_callback_t	[in]	pOtaStart	Pointer to a function that is invoked when another node initiates a firmware update. If the function returns true, the process will continue. If the function returns false, the node will report to the initiator that the firmware update requires authentication. The function must take two arguments where the first one is the firmware ID and the second one is the CRC value of the firmware. The argument is not required and can be set to NULL. In that case the firmware update process will continue without invoking the function.
CC_FirmwareUpdate_finish_callback_t	[in]	pOtaFinish	Pointer to a function that is invoked when the firmware update process finishes either successfully or with an error. If the device reboots right after the firmware update, the function is invoked before the reboot. The function must take an argument that is the status of the firmware update. The argument is not required and can be set to NULL.
bool	[in]	support_activation	Lets the application decide whether delayed activation of the firmware is supported. If set to true and the Activation bit in Request Get is set to 1, the node will not reboot to the new firmware image, but will instead wait for an Activation Set command.

This function must be invoked before a firmware update can be initiated.

Returns

- 1 if NVM is supported else 0.

ZCB_CmdClassFwUpdateMdReqReport

```
void ZCB_CmdClassFwUpdateMdReqReport (transmission_result_t * pTxResult)
```

ZCB_CmdClassFwUpdateMdReqReport Callback function receive status on Send data FIRMWARE_UPDATE_MD_REQUEST_REPORT_V3.

Parameters

Type	Direction	Argument Name	Description
transmission_result_t *	N/A	pTxResult	: TRANSMIT_COMPLETE_OK, TRANSMIT_COMPLETE_NO_ACK, TRANSMIT_COMPLETE_FAIL...

CC_FirmwareUpdate_ActivationSet_handler

```
bool CC_FirmwareUpdate_ActivationSet_handler (ZW_FIRMWARE_UPDATE_ACTIVATION_SET_V5_FRAME * pFrame, uint8_t * pStatus)
```

Handles an Activation Set command.

Parameters

Type	Direction	Argument Name	Description
ZW_FIRMWARE_UPDATE_ACTIVATION_SET_V5_FRAME *	N/A	pFrame	The Activation Set frame.
uint8_t *	N/A	pStatus	Status if the activation failed.

If the fields in the frame match the firmware that is ready to be activated, the device will reboot into the new image and transmit an Activation Report. If the fields do not match, the function will return false. **Returns**

- Returns false if the received values do not match with the stored firmware image. If they match, the function will not return, but the device will reboot from the new image.

handleCommandClassFirmwareUpdateMaxFragmentSize

```
uint16_t handleCommandClassFirmwareUpdateMaxFragmentSize (void )
```

Returns the maximum fragment size.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Maximum fragment size.

getFWUpdateMDGetNumberOfReports

```
uint8_t getFWUpdateMDGetNumberOfReports (void )
```

Gets maximum number of reports that can be sent in FW Update MD Get.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Number of Reports, greater of equal to 1.

handleFirmWareIdGetExtended

```
uint16_t handleFirmWareIdGetExtended (uint8_t n)
```

This function returns a firmware ID based on a given target index.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	n	the target index (0,1..N-1)

Returns

- target n firmware ID

handleCmdClassFirmwareUpdateMdReqGet

```
void handleCmdClassFirmwareUpdateMdReqGet (RECEIVE_OPTIONS_TYPE_EX * rxOpt,
ZW_FIRMWARE_UPDATE_MD_REQUEST_GET_V5_FRAME * pFrame, uint8_t cmdLength, uint8_t * pStatus)
```

Initiates a firmware update.

Parameters

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE_EX *	[in]	rxOpt	The options that the Firmware Update MD Request Get options must be passed because they are used later w report.
ZW_FIRMWARE_UPDATE_MD_REQUEST_GET_V5_FRAME *	[in]	pFrame	Pointer to the Firmware Update MD Request Get frame
uint8_t	[in]	cmdLength	Length of the frame.
uint8_t *	[out]	pStatus	Pointer to a value where the status of the initiation can can take one of the following values: FIRMWARE_UPDATE_MD_REQUEST_REPORT_INVALID_C FIRMWARE_UPDATE_MD_REQUEST_REPORT_REQUIRES FIRMWARE_UPDATE_MD_REQUEST_REPORT_INVALID_F FIRMWARE_UPDATE_MD_REQUEST_REPORT_NOT_UPGR FIRMWARE_UPDATE_MD_REQUEST_REPORT_INVALID_H or FIRMWARE_UPDATE_MD_REQUEST_REPORT_VALID_C

handleCmdClassFirmwareUpdateMdReport

```
void handleCmdClassFirmwareUpdateMdReport (uint16_t crc16Result, uint16_t firmwareUpdateReportNumber, uint8_t properties, uint8_t * pData, uint8_t fw_actualFrameSize)
```

handleCmdClassFirmwareUpdateMdReport Application function to handle incoming frame Firmware update MD Report

Parameters

Type	Direction	Argument Name	Description
uint16_t	N/A	crc16Result	
uint16_t	N/A	firmwareUpdateReportNumber	
uint8_t	N/A	properties	
uint8_t *	N/A	pData	
uint8_t	N/A	fw_actualFrameSize	

Indicator

Indicator

Indicator CC is mandatory since Z-Wave+ V2.

It can be used to identify a device in the network, by sending command that will, for example, set the LED indicator to ON over a certain period.

Modules

[s_CC_indicator_data_t](#)

Typedefs

typedef struct

[s_CC_indicator_data_t](#)

[s_CC_indicator_data_t](#)

Struct used to pass operational data to TSE module.

Functions

- void

[CC_Indicator_RefreshIndicatorProperties](#)(void)

Resets the indicator CC properties to their "off" values.
- void

[cc_indicator_handler](#)(uint32_t on_time_ms, uint32_t off_time_ms, uint32_t num_cycles)

CC Indicator handler that MUST be defined by the platform specific application source code to signal patterns.

Macros

- #define

INDICATOR_IND_NA

INDICATOR_SET_NA_V3
- #define

INDICATOR_IND_NODE_IDENTIFY

INDICATOR_SET_NODE_IDENTIFY_V3
- #define

INDICATOR_PROP_ON_OFF_PERIOD

INDICATOR_SET_ON_OFF_PERIOD_V3
- #define

INDICATOR_PROP_ON_OFF_CYCLES

INDICATOR_SET_ON_OFF_CYCLES_V3
- #define

INDICATOR_PROP_ON_TIME

INDICATOR_SET_ONE_TIME_ON_OFF_PERIOD_V3
- #define

INDICATOR_OBJECT_COUNT_MASK

INDICATOR_SET_PROPERTIES1_INDICATOR_OBJECT_COUNT_MASK_V3
- #define

INDICATOR_RESERVED_MASK

INDICATOR_SET_PROPERTIES1_RESERVED_MASK_V3
- #define

INDICATOR_RESERVED_SHIFT

INDICATOR_SET_PROPERTIES1_RESERVED_SHIFT_V3

Typedef Documentation

s_CC_indicator_data_t

typedef struct s_CC_indicator_data_t s_CC_indicator_data_t

Struct used to pass operational data to TSE module.

- See [ZAF_TSE_Trigger\(\)](#) for more information.
- Receive options MUST be the first element as TSE relies on the receive options.

Function Documentation

CC_Indicator_RefreshIndicatorProperties

```
void CC_Indicator_RefreshIndicatorProperties (void )
```

Resets the indicator CC properties to their "off" values.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

If the application manually de-activates the Indicator (e.g. learn mode), it should call this function to have the CC Indicator values refreshed before it reports its state to the Lifeline destinations.

When starting a blink sequence we simply off load the blinking to a board specific indicator LED implementation. The indicator command class does not keep time on when the indicator LED has completed its blink sequence, instead it queries the board when a GET request is received.

cc_indicator_handler

```
void cc_indicator_handler (uint32_t on_time_ms, uint32_t off_time_ms, uint32_t num_cycles)
```

CC Indicator handler that MUST be defined by the platform specific application source code to signal patterns.

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	on_time_ms	ON duration (in milliseconds) for a single blink cycle. If on_time_ms is zero the indicator should be turned off.
uint32_t	N/A	off_time_ms	OFF duration (in milliseconds) for a single blink cycle.
uint32_t	N/A	num_cycles	Number of blink cycles. If num_cycles is zero the indicator LED should blink "forever" or until the next time this function is called.

Even though on/off time parameters are given in milliseconds, the resolution is 1/10'th of a second.

s_CC_indicator_data_t_

Struct used to pass operational data to TSE module.

See [ZAF_TSE_Trigger\(\)](#) for more information.

- Receive options MUST be the first element as TSE relies on the receive options.

Public Attributes

RECEIVE_OPTION	rxOptions
S_TYPE_EX	rxOptions
uint8_t	indicatorId
	Indicator Id.

Public Attribute Documentation

rxOptions

RECEIVE_OPTIONS_TYPE_EX s_CC_indicator_data_t_::rxOptions

rxOptions

indicatorId

uint8_t s_CC_indicator_data_t_::indicatorId

Indicator Id.

Multichan

Multichan

Modules

[cc_multi_channel_config_t](#)

Functions

bool	ZAF_CC_MultiChannel_IsCCSupported (RECEIVE_OPTIONS_TYPE_EX *pRxOpt, ZW_APPLICATION_TX_BUFFER *pCmd) Given a frame and the associated receive options this function returns whether the command is supported or not.
void	CmdClassMultiChannelEncapsulate (uint8_t **ppData, size_t *dataLength, TRANSMIT_OPTIONS_TYPE_SINGLE_EX *pTxOptionsEx) Encapsulates a frame with Multi Channel.
cc_multi_channel_config_t const *	cc_multi_channel_get_config (void)
cc_multi_channel_config_t const *	cc_multi_channel_get_config_endpoint (uint8_t endpoint)
bool	cc_multi_channel_are_endpoints_identical (void)

Function Documentation

ZAF_CC_MultiChannel_IsCCSupported

```
bool ZAF_CC_MultiChannel_IsCCSupported (RECEIVE_OPTIONS_TYPE_EX * pRxOpt, ZW_APPLICATION_TX_BUFFER * pCmd)
```

Given a frame and the associated receive options this function returns whether the command is supported or not.

Parameters

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE_EX *	[in]	pRxOpt	Receive options.
ZW_APPLICATION_TX_BUFFER *	[in]	pCmd	Frame including command.

Returns

- true if the command is supported, false otherwise.

CmdClassMultiChannelEncapsulate

```
void CmdClassMultiChannelEncapsulate (uint8_t ** ppData, size_t * dataLength, TRANSMIT\_OPTIONS\_TYPE\_SINGLE\_EX * pTxOptionsEx)
```

Encapsulates a frame with Multi Channel.

Parameters

Type	Direction	Argument Name	Description
uint8_t **	[inout]	ppData	Pointer to data.
size_t *	[inout]	dataLength	Pointer to data length.
TRANSMIT_OPTIONS_TYPE_SINGLE_EX *	[in]	pTxOptionsEx	Pointer to transmit options.

cc_multi_channel_get_config

```
cc_multi_channel_config_t const * cc_multi_channel_get_config (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multi_channel_get_config_endpoint

```
cc_multi_channel_config_t const * cc_multi_channel_get_config_endpoint (uint8_t endpoint)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	endpoint	

cc_multi_channel_are_endpoints_identical

```
bool cc_multi_channel_are_endpoints_identical (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multi_channel_config_t

Public Attributes

- uint8_t [generic_type](#)
- uint8_t [specific_type](#)

Public Attribute Documentation

generic_type

uint8_t cc_multi_channel_config_t::generic_type

specific_type

uint8_t cc_multi_channel_config_t::specific_type

MultilevelSensor

MultilevelSensor

Modules

[_sensor_type](#)

[_sensor_read_result](#)

[_sensor_interface](#)

[_sensor_administration](#)

Enumerations

```
enum sensor\_name {  
    SENSOR_NAME_AIR_TEMPERATURE = 0x00  
    SENSOR_NAME_GENERAL_PURPOSE  
    SENSOR_NAME_ILLUMINANCE  
    SENSOR_NAME_POWER  
    SENSOR_NAME_HUMIDITY  
    SENSOR_NAME_ACCELERATION_X  
    SENSOR_NAME_ACCELERATION_Y  
    SENSOR_NAME_ACCELERATION_Z  
    SENSOR_NAME_MAX_COUNT  
}  
Defined possible sensor types.
```

```
enum sensor\_read\_result\_size\_t {  
    SENSOR_READ_RESULT_SIZE_1 = 1  
    SENSOR_READ_RESULT_SIZE_2 = 2  
    SENSOR_READ_RESULT_SIZE_4 = 4  
    SENSOR_READ_RESULT_SIZE_END = 5  
}  
Defined possible sensor read size.
```

```
enum sensor\_read\_result\_precision {  
    SENSOR_READ_RESULT_PRECISION_1 = 1  
    SENSOR_READ_RESULT_PRECISION_2  
    SENSOR_READ_RESULT_PRECISION_3  
    SENSOR_READ_RESULT_PRECISION_END  
}  
Defined possible sensor read precision.
```

```
enum sensor\_interface\_return\_value\_t {  
    SENSOR_INTERFACE_RETURN_VALUE_OK  
    SENSOR_INTERFACE_RETURN_VALUE_ALREADY_SET  
    SENSOR_INTERFACE_RETURN_VALUE_INVALID_SCALE_VALUE  
    SENSOR_INTERFACE_RETURN_VALUE_ERROR  
    SENSOR_INTERFACE_RETURN_VALUE_END  
}  
Defined sensor interface return values.
```

```
enum cc\_multilevel\_sensor\_return\_value {  
    CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK
```

```
CC_MULTILEVEL_SENSOR_RETURN_VALUE_ERROR
CC_MULTILEVEL_SENSOR_RETURN_VALUE_NOT_FOUND
CC_MULTILEVEL_SENSOR_RETURN_VALUE_ALREADY_REGISTERED
CC_MULTILEVEL_SENSOR_RETURN_VALUE_REGISTRATION_LIMIT_REACHED
CC_MULTILEVEL_SENSOR_RETURN_VALUE_END
}
```

Defined Multilevel Sensor return values.

Typedefs

<code>typedef enum</code> <code>sensor_name</code>	<code>sensor_name_t</code> Defined possible sensor types.
<code>typedef struct</code> <code>_sensor_type</code>	<code>sensor_type_t</code> Structure that holds the attributes of a sensor type.
<code>typedef struct</code> <code>_sensor_read_result</code> <code>ult</code>	<code>sensor_read_result_t</code>
<code>typedef struct</code> <code>_sensor_interface</code>	<code>sensor_interface_t</code> Structure that holds a read result from the read interface.
<code>typedef struct</code> <code>_sensor_interface</code>	<code>sensor_interface_iterator_t</code>
<code>typedef struct</code> <code>_sensor_administration</code>	<code>sensor_administration_t</code> Structure that administrate all of the registered sensors.

Functions

<code>void</code>	<code>cc_multilevel_sensor_send_sensor_data(void)</code> This function will report the registered sensor's measured datas to the Lifeline group.
<code>void</code>	<code>cc_multilevel_sensor_config_register_instances(void)</code>
<code>void</code>	<code>cc_multilevel_sensor_init_all_sensor(void)</code> Calls the init function for all of the registered sensor interfaces.
<code>uint8_t</code>	<code>cc_multilevel_sensor_check_scale(const sensor_interface_t *i_interface, uint8_t i_scale)</code> Checks if the wanted scale is a legal one for an interface.
<code>cc_multilevel_sensor_return_value</code>	<code>cc_multilevel_sensor_check_sensor_type_registered(uint8_t endpoint, uint8_t sensor_type_value)</code> Checks if the wanted sensor type has a registered interface or not.
<code>cc_multilevel_sensor_return_value</code>	<code>cc_multilevel_sensor_registration(sensor_interface_t *i_new_sensor)</code> Registers an interface to a specific sensor type.
<code>cc_multilevel_sensor_return_value</code>	<code>cc_multilevel_sensor_get_supported_sensors(uint8_t *o_supported_sensor_buffer)</code> Fills an input buffer with flags where each flag represents a supported sensor.
<code>cc_multilevel_sensor_return_value</code>	<code>cc_multilevel_sensor_get_supported_scale(uint8_t sensor_type_value, uint8_t *o_supported_scale)</code> Fills an input buffer with flags where each flag represents a supported scale.
<code>cc_multilevel_sensor_return_value</code>	<code>cc_multilevel_sensor_get_interface(uint8_t endpoint, uint8_t sensor_type_value, sensor_interface_t **o_interface)</code> Getter function for a sensor type's interface.
<code>cc_multilevel_sensor_return_value</code>	<code>cc_multilevel_sensor_get_default_sensor_type(uint8_t *o_default_sensor_type)</code> Gets the default sensor type value.

<code>cc_multilevel_sensor_return_value</code>	<code>cc_multilevel_sensor_init_iterator</code> (sensor_interface_iterator_t **i_iterator) Initialize an iterator struct for the registered sensor interfaces.
<code>void</code>	<code>cc_multilevel_sensor_next_iterator</code> (sensor_interface_iterator_t **i_iterator) Moves the iterator reference forward to the next registered interface, if the current is the last one then NULL. If there is any registered interface the iterator will be NULL.
<code>uint8_t</code>	<code>cc_multilevel_sensor_get_number_of_registered_sensors</code> (void) Returns the number of registered sensor interfaces.
<code>void</code>	<code>cc_multilevel_sensor_reset_administration</code> (void) Removes all of the registered sensor interfaces from the administration.
<code>const sensor_type_t *</code>	<code>cc_multilevel_sensor_get_sensor_type</code> (sensor_name_t i_sensor_name) Returns a sensor_type_t struct from the sensor_types config table.
<code>sensor_interface_return_value_t</code>	<code>cc_multilevel_sensor_init_interface</code> (sensor_interface_t *i_instance, sensor_name_t i_name) Initialize a sensor interface instance.
<code>sensor_interface_return_value_t</code>	<code>cc_multilevel_sensor_add_supported_scale_interface</code> (sensor_interface_t *i_instance, uint8_t i_scale) Adds a new scale to an existing interface.

Macros

<code>#define</code>	<code>MULTILEVEL_SENSOR_DEFAULT_AUTOREPORT_PERIOD_SEC</code> (<code>MULTILEVEL_SENSOR_DEFAULT_AUTOREPORT_PERIOD_MIN</code> * 60)
<code>#define</code>	<code>MULTILEVEL_SENSOR_DEFAULT_AUTOREPORT_PERIOD_MS</code> (<code>MULTILEVEL_SENSOR_DEFAULT_AUTOREPORT_PERIOD_SEC</code> * 1000)
<code>#define</code>	<code>SENSOR_SCALE_DEFAULT</code> 0x00
<code>#define</code>	<code>SENSOR_SCALE_CELSIUS</code> 0x00
<code>#define</code>	<code>SENSOR_SCALE_FAHRENHEIT</code> 0x01
<code>#define</code>	<code>SENSOR_SCALE_PERCENTAGE</code> 0x00
<code>#define</code>	<code>SENSOR_SCALE_DIMENSIONLESS</code> 0x01
<code>#define</code>	<code>SENSOR_SCALE_LUX</code> 0x01
<code>#define</code>	<code>SENSOR_SCALE_WATT</code> 0x00
<code>#define</code>	<code>SENSOR_SCALE_BTU_H</code> 0x01 /*Btu/h*/
<code>#define</code>	<code>SENSOR_SCALE_ABSOLUTE_HUMIDITY</code> 0x01 /*g/m3*/
<code>#define</code>	<code>SENSOR_SCALE_METER_SQUARE_SECOND</code> 0x00 /*m/s^2*/
<code>#define</code>	<code>SLI_MAX_RAW_RESULT_BYTES</code> 4 Structure that holds a read result from the read interface.

Enumeration Documentation

sensor_name

<code>sensor_name</code>

Defined possible sensor types.

Enumerator	
SENSOR_NAME_AIR_TEMPERATURE	
SENSOR_NAME_GENERAL_PURPOSE	
SENSOR_NAME_ILLUMINANCE	
SENSOR_NAME_POWER	
SENSOR_NAME_HUMIDITY	
SENSOR_NAME_ACCELERATION_X	
SENSOR_NAME_ACCELERATION_Y	
SENSOR_NAME_ACCELERATION_Z	
SENSOR_NAME_MAX_COUNT	

sensor_read_result_size_t

sensor_read_result_size_t

Defined possible sensor read size.

Enumerator	
SENSOR_READ_RESULT_SIZE_1	
SENSOR_READ_RESULT_SIZE_2	
SENSOR_READ_RESULT_SIZE_4	
SENSOR_READ_RESULT_SIZE_END	

sensor_read_result_precision

sensor_read_result_precision

Defined possible sensor read precision.

Enumerator	
SENSOR_READ_RESULT_PRECISION_1	
SENSOR_READ_RESULT_PRECISION_2	
SENSOR_READ_RESULT_PRECISION_3	
SENSOR_READ_RESULT_PRECISION_END	

sensor_interface_return_value_t

sensor_interface_return_value_t

Defined sensor interface return values.

Enumerator	
SENSOR_INTERFACE_RETURN_VALUE_OK	
SENSOR_INTERFACE_RETURN_VALUE_ALREADY_SET	
SENSOR_INTERFACE_RETURN_VALUE_INVALID_SCALE_VALUE	
SENSOR_INTERFACE_RETURN_VALUE_ERROR	
SENSOR_INTERFACE_RETURN_VALUE_END	

cc_multilevel_sensor_return_value

```
cc_multilevel_sensor_return_value
```

Defined Multilevel Sensor return values.

Enumerator	
CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK	
CC_MULTILEVEL_SENSOR_RETURN_VALUE_ERROR	
CC_MULTILEVEL_SENSOR_RETURN_VALUE_NOT_FOUND	
CC_MULTILEVEL_SENSOR_RETURN_VALUE_ALREADY_REGISTERED	
CC_MULTILEVEL_SENSOR_RETURN_VALUE_REGISTRATION_LIMIT_REACHED	
CC_MULTILEVEL_SENSOR_RETURN_VALUE_END	

Typedef Documentation

sensor_name_t

```
typedef enum sensor_name sensor_name_t
```

Defined possible sensor types.

sensor_type_t

```
typedef struct _sensor_type sensor_type_t
```

Structure that holds the attributes of a sensor type.

sensor_read_result_t

```
typedef struct _sensor_read_result sensor_read_result_t
```

sensor_interface_t

```
typedef struct _sensor_interface sensor_interface_t
```

Structure that holds a read result from the read interface.

sensor_interface_iterator_t

```
typedef struct _sensor_interface sensor_interface_iterator_t
```

sensor_administration_t

```
typedef struct _sensor_administration sensor_administration_t
```

Structure that administrate all of the registered sensors.

Function Documentation

cc_multilevel_sensor_send_sensor_data

```
void cc_multilevel_sensor_send_sensor_data (void )
```

This function will report the registered sensor's measured datas to the Lifeline group.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multilevel_sensor_config_register_instances

```
void cc_multilevel_sensor_config_register_instances (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multilevel_sensor_init_all_sensor

```
void cc_multilevel_sensor_init_all_sensor (void )
```

Calls the init function for all of the registered sensor interfaces.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multilevel_sensor_check_scale

```
uint8_t cc_multilevel_sensor_check_scale (const sensor\_interface\_t * i_interface, uint8_t i_scale)
```

Checks if the wanted scale is a legal one for an interface.

Parameters

Type	Direction	Argument Name	Description
const sensor_interface_t *	[in]	i_interface	Pointer to an existing sensor interface
uint8_t	[in]	i_scale	Scale value to check

If the scale is not set the function will return with the first available scale. The function will iterate from bit 0 to 7 to look for the first scale which is supported. That will be the default scale. **Returns**

- a usable scale value, if the input i_scale is valid that will be return, else the default one.

cc_multilevel_sensor_check_sensor_type_registered

```
cc_multilevel_sensor_return_value cc_multilevel_sensor_check_sensor_type_registered (uint8_t endpoint, uint8_t sensor_type_value)
```

Checks if the wanted sensor type has a registered interface or not.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	Sensor type value to check, this is the value in sensor_type_t struct's value
uint8_t	N/A	sensor_type_value	

Returns

- CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK if interface is registered for a sensor, else CC_MULTILEVEL_SENSOR_RETURN_VALUE_NOT_FOUND.

cc_multilevel_sensor_registration

```
cc_multilevel_sensor_return_value cc_multilevel_sensor_registration (sensor\_interface\_t * i_new_sensor)
```

Registers an interface to a specific sensor type.

Parameters

Type	Direction	Argument Name	Description
sensor_interface_t *	[in]	i_new_sensor	Pointer to a filled sensor interface struct

The number of registered sensor types is maximized with REGISTERED_SENSOR_NUMBER_LIMIT **Returns**

- CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK if interface is registered for a sensor.

cc_multilevel_sensor_get_supported_sensors

```
cc_multilevel_sensor_return_value cc_multilevel_sensor_get_supported_sensors (uint8_t * o_supported_sensor_buffer)
```

Fills an input buffer with flags where each flag represents a supported sensor.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	o_supported_sensor_buffer	Pointer to a buffer where the supported sensors' are represented as flags in a byte array. The position of each flag is calculated based on SDS13812. The buffer must be 11 byte long at least.

Returns

- CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK if success, else CC_MULTILEVEL_SENSOR_RETURN_VALUE_ERROR.

cc_multilevel_sensor_get_supported_scale

```
cc_multilevel_sensor_return_value cc_multilevel_sensor_get_supported_scale (uint8_t sensor_type_value, uint8_t * o_supported_scale)
```

Fills an input buffer with flags where each flag represents a supported scale.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	sensor_type_value	Sensor type value to check, this is the value in sensor_type_t struct's value
uint8_t *	[out]	o_supported_scale	Pointer to a buffer where the supported scales will be represented for a sensor type

Returns

- CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK if interface is registered for a sensor.

cc_multilevel_sensor_get_interface

```
cc_multilevel_sensor_return_value cc_multilevel_sensor_get_interface (uint8_t endpoint, uint8_t sensor_type_value, sensor_interface_t ** o_interface)
```

Getter function for a sensor type's interface.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	Sensor type value to check, this is the value in sensor_type_t struct's value
uint8_t	[out]	sensor_type_value	This is a double pointer which will hold an interface reference
sensor_interface_t **	N/A	o_interface	

Returns

- CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK if interface is registered for a sensor, else CC_MULTILEVEL_SENSOR_RETURN_VALUE_NOT_FOUND.

cc_multilevel_sensor_get_default_sensor_type

```
cc_multilevel_sensor_return_value cc_multilevel_sensor_get_default_sensor_type (uint8_t * o_default_sensor_type)
```


Gets the default sensor type value.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[in]	o_default_sensor_type	Pointer to a byte buffer which will hold the default sensor type value

Basically the first registered sensor interface' type is the default one. **Returns**

- CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK if there is any registered interface, else CC_MULTILEVEL_SENSOR_RETURN_VALUE_ERROR.

cc_multilevel_sensor_init_iterator

```
cc_multilevel_sensor_return_value cc_multilevel_sensor_init_iterator (sensor_interface_iterator_t ** i_iterator)
```

Initialize an iterator struct for the registered sensor interfaces.

Parameters

Type	Direction	Argument Name	Description
sensor_interface_iterator_t **	[out]	i_iterator	This is a double pointer which will hold an interface reference to the first registered interface

If there is any registered interface the iterator will be NULL. **Returns**

- CC_MULTILEVEL_SENSOR_RETURN_VALUE_OK if success, else CC_MULTILEVEL_SENSOR_RETURN_VALUE_ERROR..

cc_multilevel_sensor_next_iterator

```
void cc_multilevel_sensor_next_iterator (sensor_interface_iterator_t ** i_iterator)
```

Moves the iterator reference forward to the next registered interface, if the current is the last one then NULL. If there is any registered interface the iterator will be NULL.

Parameters

Type	Direction	Argument Name	Description
sensor_interface_iterator_t **	[out]	i_iterator	This is a double pointer which will hold an interface reference

cc_multilevel_sensor_get_number_of_registered_sensors

```
uint8_t cc_multilevel_sensor_get_number_of_registered_sensors (void )
```

Returns the number of registered sensor interfaces.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Number of registered sensor interfaces.

cc_multilevel_sensor_reset_administration

```
void cc_multilevel_sensor_reset_administration (void )
```

Removes all of the registered sensor interfaces from the administration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multilevel_sensor_get_sensor_type

```
const sensor_type_t * cc_multilevel_sensor_get_sensor_type (sensor_name_t i_sensor_name)
```

Returns a sensor_type_t struct from the sensor_types config table.

Parameters

Type	Direction	Argument Name	Description
sensor_name_t	[in]	i_sensor_name	The name of the sensor type, the sensor type attributes' reference will be look for based on this value

Returns

- sensor_type_t struct which includes the type attributes.

cc_multilevel_sensor_init_interface

```
sensor_interface_return_value_t cc_multilevel_sensor_init_interface (sensor_interface_t * i_instance, sensor_name_t i_name)
```

Initialize a sensor interface instance.

Parameters

Type	Direction	Argument Name	Description
sensor_interface_t *	[in]	i_instance	Pointer to an existing sensor interface to init
sensor_name_t	[in]	i_name	The name of the sensor type, the sensor type attributes' reference will be look for based on this value

Sets the whole struct to zero and sets the reference to the appropriate sensor_type_t struct. **Returns**

- Status of the initialization.

cc_multilevel_sensor_add_supported_scale_interface

```
sensor_interface_return_value_t cc_multilevel_sensor_add_supported_scale_interface (sensor_interface_t * i_instance,
uint8_t i_scale)
```

Adds a new scale to an existing interface.

Parameters

Type	Direction	Argument Name	Description
sensor_interface_t *	[in]	i_instance	Pointer to an existing sensor interface which the new scale will be registered to
uint8_t	[in]	i_scale	The new scale which will be registered

Supported scales must be set by this function during initialization. **Returns**

- Status of the new scale value registration.

_sensor_type

Structure that holds the attributes of a sensor type.

Public Attributes

uint8_t	value	Sensor type id from SDS13812.
uint8_t	byte_offset	Supported bitmask byte number from SDS13812.
uint8_t	bit_mask	Supported bitmask bit number from SDS13812.
uint8_t	max_scale_value	Maximum supported scale number from SDS13812.

Public Attribute Documentation

value

uint8_t _sensor_type::value

Sensor type id from SDS13812.

byte_offset

uint8_t _sensor_type::byte_offset

Supported bitmask byte number from SDS13812.

bit_mask

uint8_t _sensor_type::bit_mask

Supported bitmask bit number from SDS13812.

max_scale_value

uint8_t _sensor_type::max_scale_value

Maximum supported scale number from SDS13812.

_sensor_read_result

Public Attributes

<code>uint8_t</code>	<code>raw_result</code>	The raw buffer which holds the result.
<code>sensor_read_result_precision</code>	<code>precision</code>	The precision which the raw result should be interpret with.
<code>sensor_read_result_size_t</code>	<code>size_bytes</code>	The size which the raw result should be interpret with.

Public Attribute Documentation

raw_result

`uint8_t _sensor_read_result::raw_result[SLI_MAX_RAW_RESULT_BYTES]`

The raw buffer which holds the result.

precision

`sensor_read_result_precision _sensor_read_result::precision`

The precision which the raw result should be interpret with.

size_bytes

`sensor_read_result_size_t _sensor_read_result::size_bytes`

The size which the raw result should be interpret with.

_sensor_interface

Structure that holds a read result from the read interface.

Public Attributes

RECEIVE_OPTION S_TYPE_EX	rxOpt	Contains information required for Supervision and True Status.
uint8_t	endpoint	The sensor must be tied to an endpoint.
const sensor_type_t *	sensor_type	Reference of a sensor type structure.
uint8_t	supported_scale	Each bit represents a supported scale.
bool(*	init	Function pointer to initialize a sensor.
bool(*	deinit	Function pointer to deinitialize a sensor.
bool(*	read_value	Function pointer to read a sensor value.

Public Attribute Documentation

rxOpt

RECEIVE_OPTIONS_TYPE_EX _sensor_interface::rxOpt

Contains information required for Supervision and True Status.
Must be the first element in this struct because TSE assumes this location.

endpoint

uint8_t _sensor_interface::endpoint

The sensor must be tied to an endpoint.
Must be set to 0 if no endpoints. The endpoint value MUST be located as the second element in this struct as ZAF depends on that for generation of the Node Information Frame.

sensor_type

const sensor_type_t* _sensor_interface::sensor_type

Reference of a sensor type structure.

supported_scale

```
uint8_t _sensor_interface::supported_scale
```

Each bit represents a supported scale.

init

```
bool(* _sensor_interface::init) (void)
```

Function pointer to initialize a sensor.

deinit

```
bool(* _sensor_interface::deinit) (void)
```

Function pointer to deinitialize a sensor.

read_value

```
bool(* _sensor_interface::read_value) (sensor_read_result_t *o_result, uint8_t i_scale)
```

Function pointer to read a sensor value.

_sensor_administration

Structure that administrate all of the registered sensors.

Public Attributes

<code>sensor_interface_t *</code>	<code>registrated_sensors</code> Reference of the sensor interfaces.
<code>uint8_t</code>	<code>number_of_registrated_sensors</code> Stores how many sensors are registered.

Public Attribute Documentation

registrated_sensors

```
sensor_interface_t*
_sensor_administration::registrated_sensors[MULTILEVEL_SENSOR_REGISTERED_SENSOR_NUMBER_LIMIT]
```

Reference of the sensor interfaces.

number_of_registrated_sensors

```
uint8_t _sensor_administration::number_of_registrated_sensors
```

Stores how many sensors are registered.

MultilevelSwitch

MultilevelSwitch

Modules

```
cc_multilevel_switch_t
```

Enumerations

```
enum CCMLS_PRIMARY_SWITCH_T {
    CCMLS_PRIMARY_SWITCH_UP
    CCMLS_PRIMARY_SWITCH_DOWN
    CCMLS_PRIMARY_SWITCH_RESERVED
    CCMLS_PRIMARY_SWITCH_NO_UP_DOWN_MOTION
}
Enumeration for "Start Level Change" command.

enum CCMLS_IGNORE_START_LEVEL_T {
    CCMLS_IGNORE_START_LEVEL_FALSE
    CCMLS_IGNORE_START_LEVEL_TRUE
}
Enumeration for "Start Level Change" command.

enum CCMLS_SECONDARY_SWITCH_T {
    CCMLS_SECONDARY_SWITCH_INCREMENT
    CCMLS_SECONDARY_SWITCH_DECREMENT
    CCMLS_SECONDARY_SWITCH_RESERVED
    CCMLS_SECONDARY_SWITCH_NO_INC_DEC
}
Enumeration for "Start Level Change" command.

enum cc_multilevel_switch_event_t {
    CC_MULTILEVEL_SWITCH_EVENT_START_LEVEL_CHANGE
    CC_MULTILEVEL_SWITCH_EVENT_STOP_LEVEL_CHANGE
    CC_MULTILEVEL_SWITCH_EVENT_REACHED_FINAL_VALUE
}
```

Typedefs

```
typedef struct cc_multilevel_switch_t
cc_multilevel_switch_t

typedef enum cc_multilevel_switch_event_t
cc_multilevel_switch_event_t

typedef void(*) cc_multilevel_switch_callback_t(struct cc_multilevel_switch_t *p_switch)
```

Functions

JOB_STATUS	CmdClassMultilevelSwitchStartLevelChange (AGI_PROFILE *pProfile, uint8_t sourceEndpoint, void(*pCbFunc) (TRANSMISSION_RESULT *pTransmissionResult), CCMLS_PRIMARY_SWITCH_T primarySwitch, CCMLS_IGNORE_START_LEVEL_T fIgnoreStartLevel, CCMLS_SECONDARY_SWITCH_T secondarySwitch, uint8_t primarySwitchStartLevel, uint8_t duration, uint8_t secondarySwitchStepSize) Initiates the transmission of a "Multilevel Switch Start Level Change" command.
JOB_STATUS	CmdClassMultilevelSwitchStopLevelChange (AGI_PROFILE *pProfile, uint8_t sourceEndpoint, void(*pCbFunc) (TRANSMISSION_RESULT *pTransmissionResult)) Initiates the transmission of a "Multilevel Switch Stop Level Change" command.
JOB_STATUS	CmdClassMultilevelSwitchSetTransmit (AGI_PROFILE *pProfile, uint8_t sourceEndpoint, void(*pCbFunc) (TRANSMISSION_RESULT *pTransmissionResult), uint8_t value, uint8_t duration) Initiates the transmission of a "Multilevel Switch Set" command.
void	cc_multilevel_switch_set (cc_multilevel_switch_t *p_switch, uint8_t value, uint8_t duration) Sets the value of a given multilevel switch.
void	cc_multilevel_switch_start_level_change (cc_multilevel_switch_t *p_switch, bool up, bool ignore_start_level, uint8_t start_level, uint8_t duration) Starts changing the value of a given multilevel switch.
void	cc_multilevel_switch_set_level (cc_multilevel_switch_t *p_switch, uint8_t level, uint8_t duration) Sets the level of a given multilevel switch.
void	cc_multilevel_switch_stop_level_change (cc_multilevel_switch_t *p_switch) Stops an ongoing change of a given multilevel switch.
uint8_t	cc_multilevel_switch_get_current_value (cc_multilevel_switch_t *p_switch) Returns the current value of a given multilevel switch.
uint8_t	cc_multilevel_switch_get_last_on_value (cc_multilevel_switch_t *p_switch) Returns the last on-value of a given multilevel switch.
uint8_t	cc_multilevel_switch_get_min_value (void) Returns the min value of all multilevel switches.
uint8_t	cc_multilevel_switch_get_max_value (void) Returns the max value of all multilevel switches.
void	cc_multilevel_switch_support_cb (struct cc_multilevel_switch_t *p_switch)
cc_multilevel_switch_t *	cc_multilevel_switch_support_config_get_switches (void)
uint8_t	cc_multilevel_switch_support_config_get_length_switches (void)
uint8_t	cc_multilevel_switch_support_config_get_default_duration (void)
bool	cc_multilevel_switch_write (uint8_t multilevel_component_id, cc_multilevel_switch_t *multilevel_switch)
bool	cc_multilevel_switch_read (uint8_t multilevel_component_id, cc_multilevel_switch_t *multilevel_switch)

Enumeration Documentation

CCMLS_PRIMARY_SWITCH_T

CCMLS_PRIMARY_SWITCH_T

Enumeration for "Start Level Change" command.

Enumerator

CCMLS_PRIMARY_SWITCH_UP	CCMLS_PRIMARY_SWITCH_UP.
CCMLS_PRIMARY_SWITCH_DOWN	CCMLS_PRIMARY_SWITCH_DOWN.
CCMLS_PRIMARY_SWITCH_RESERVED	CCMLS_PRIMARY_SWITCH_RESERVED.
CCMLS_PRIMARY_SWITCH_NO_UP_DOWN_MOTION	CCMLS_PRIMARY_SWITCH_NO_UP_DOWN_MOTION.

CCMLS_IGNORE_START_LEVEL_T

CCMLS_IGNORE_START_LEVEL_T

Enumeration for "Start Level Change" command.

Enumerator	
CCMLS_IGNORE_START_LEVEL_FALSE	CCMLS_IGNORE_START_LEVEL_FALSE.
CCMLS_IGNORE_START_LEVEL_TRUE	CCMLS_IGNORE_START_LEVEL_TRUE.

CCMLS_SECONDARY_SWITCH_T

CCMLS_SECONDARY_SWITCH_T

Enumeration for "Start Level Change" command.

Enumerator	
CCMLS_SECONDARY_SWITCH_INCREMENT	CCMLS_SECONDARY_SWITCH_INCREMENT.
CCMLS_SECONDARY_SWITCH_DECREMENT	CCMLS_SECONDARY_SWITCH_DECREMENT.
CCMLS_SECONDARY_SWITCH_RESERVED	CCMLS_SECONDARY_SWITCH_RESERVED.
CCMLS_SECONDARY_SWITCH_NO_INC_DEC	CCMLS_SECONDARY_SWITCH_NO_INC_DEC.

cc_multilevel_switch_event_t

cc_multilevel_switch_event_t

Enumerator	
CC_MULTILEVEL_SWITCH_EVENT_START_LEVEL_CHANGE	The switch pointed to by the event data has started changing level.
CC_MULTILEVEL_SWITCH_EVENT_STOP_LEVEL_CHANGE	The switch pointed to by the event data has stopped changing level.
CC_MULTILEVEL_SWITCH_EVENT_REACHED_FINAL_VALUE	The switch pointed to by the event data has reached its final value.

Typedef Documentation

cc_multilevel_switch_t

typedef struct cc_multilevel_switch_t cc_multilevel_switch_t

cc_multilevel_switch_event_t

```
typedef enum cc_multilevel_switch_event_t cc_multilevel_switch_event_t
```

cc_multilevel_switch_callback_t

```
typedef void(* cc_multilevel_switch_callback_t) (struct cc_multilevel_switch_t *p_switch) )(struct cc_multilevel_switch_t *p_switch)
```

Function Documentation

CmdClassMultilevelSwitchStartLevelChange

```
JOB_STATUS CmdClassMultilevelSwitchStartLevelChange (AGI_PROFILE * pProfile, uint8_t sourceEndpoint, void(*) (TRANSMISSION_RESULT *pTransmissionResult) pCbFunc, CCMLS_PRIMARY_SWITCH_T primarySwitch, CCMLS_IGNORE_START_LEVEL_T flgnoreStartLevel, CCMLS_SECONDARY_SWITCH_T secondarySwitch, uint8_t primarySwitchStartLevel, uint8_t duration, uint8_t secondarySwitchStepSize)
```

Initiates the transmission of a "Multilevel Switch Start Level Change" command.

Parameters

Type	Direction	Argument Name	Description
AGI_PROFILE *	[in]	pProfile	pointer to AGI profile or NULL for lifeline
uint8_t	[in]	sourceEndpoint	source endpoint
void(*) (TRANSMISSION_RESULT *pTransmissionResult)	[out]	pCbFunc	Callback function to be called when transmission is done/failed.
CCMLS_PRIMARY_SWITCH_T	[in]	primarySwitch	Controls the primary device functionality.
CCMLS_IGNORE_START_LEVEL_T	[in]	flgnoreStartLevel	Ignore start level.
CCMLS_SECONDARY_SWITCH_T	[in]	secondarySwitch	Controls the secondary device functionality.
uint8_t	[in]	primarySwitchStartLevel	Start level for the primary device functionality.
uint8_t	[in]	duration	The duration from lowest to highest value.
uint8_t	[in]	secondarySwitchStepSize	Step size for secondary device functionality.

Returns

- Status of the job.

CmdClassMultilevelSwitchStopLevelChange

```
JOB_STATUS CmdClassMultilevelSwitchStopLevelChange (AGI_PROFILE * pProfile, uint8_t sourceEndpoint, void(*) (TRANSMISSION_RESULT *pTransmissionResult) pCbFunc)
```

Initiates the transmission of a "Multilevel Switch Stop Level Change" command.

Parameters

Type	Direction	Argument Name	Description
AGI_PROFILE *	[in]	pProfile	pointer to AGI profile
uint8_t	[in]	sourceEndpoint	source endpoint
void(*)(TRANSMISSION_RESULT *pTransmissionResult)	[out]	pCbFunc	Callback function to be called when transmission is done/failed.

Returns

- Status of the job.

CmdClassMultilevelSwitchSetTransmit

```
JOB_STATUS CmdClassMultilevelSwitchSetTransmit (AGI\_PROFILE * pProfile, uint8_t sourceEndpoint, void(*)(TRANSMISSION\_RESULT *pTransmissionResult) pCbFunc, uint8_t value, uint8_t duration)
```

Initiates the transmission of a "Multilevel Switch Set" command.

Parameters

Type	Direction	Argument Name	Description
AGI_PROFILE *	[in]	pProfile	pointer to AGI profile
uint8_t	[in]	sourceEndpoint	source endpoint
void(*)(TRANSMISSION_RESULT *pTransmissionResult)	[out]	pCbFunc	Callback function to be called when transmission is done/failed.
uint8_t	[in]	value	Multilevel value.
uint8_t	[in]	duration	The duration from current value to the new given value.

Returns

- Status of the job.

cc_multilevel_switch_set

```
void cc_multilevel_switch_set (cc\_multilevel\_switch\_t * p_switch, uint8_t value, uint8_t duration)
```

Sets the value of a given multilevel switch.

Parameters

Type	Direction	Argument Name	Description
cc_multilevel_switch_t *	N/A	p_switch	The switch of which the value is set.
uint8_t	N/A	value	The value to set.
uint8_t	N/A	duration	The time it takes to reach the value.

This function can be used in an application that might have a button.

Must be used for local actuation only.

cc_multilevel_switch_start_level_change

```
void cc_multilevel_switch_start_level_change (cc_multilevel_switch_t * p_switch, bool up, bool ignore_start_level, uint8_t start_level, uint8_t duration)
```

Starts changing the value of a given multilevel switch.

Parameters

Type	Direction	Argument Name	Description
cc_multilevel_switch_t *	N/A	p_switch	The multilevel switch of which the value is changed.
bool	N/A	up	The direction of the change. If true the value will decrease. If false, the value will increase.
bool	N/A	ignore_start_level	Decides whether the start level is ignored or not. If set to true, the change will start from the current value. If set to false, the value will start it's change from the given start level.
uint8_t	N/A	start_level	The value to start from if ignore_start_level is set to false.
uint8_t	N/A	duration	The duration of the change.

Must be used for local actuation only.

cc_multilevel_switch_set_level

```
void cc_multilevel_switch_set_level (cc_multilevel_switch_t * p_switch, uint8_t level, uint8_t duration)
```

Sets the level of a given multilevel switch.

Parameters

Type	Direction	Argument Name	Description
cc_multilevel_switch_t *	N/A	p_switch	The multilevel switch of which the level will be set.
uint8_t	N/A	level	The level to set. (8 bit value)
uint8_t	N/A	duration	The time it takes to reach the level in seconds.

cc_multilevel_switch_stop_level_change

```
void cc_multilevel_switch_stop_level_change (cc_multilevel_switch_t * p_switch)
```

Stops an ongoing change of a given multilevel switch.

Parameters

Type	Direction	Argument Name	Description
cc_multilevel_switch_t *	N/A	p_switch	The multilevel switch of which the change will be stopped.

If no change is ongoing, nothing happens.

Must be used for local actuation only.

cc_multilevel_switch_get_current_value

```
uint8_t cc_multilevel_switch_get_current_value (cc_multilevel_switch_t * p_switch)
```

Returns the current value of a given multilevel switch.

Parameters

Type	Direction	Argument Name	Description
cc_multilevel_switch_t *	N/A	p_switch	The multilevel switch of which the value is returned.

Must be used for local actuation only. **Returns**

- Current value of the given multilevel switch.

cc_multilevel_switch_get_last_on_value

```
uint8_t cc_multilevel_switch_get_last_on_value (cc_multilevel_switch_t * p_switch)
```

Returns the last on-value of a given multilevel switch.

Parameters

Type	Direction	Argument Name	Description
cc_multilevel_switch_t *	N/A	p_switch	The multilevel switch of which the last on-value is returned.

Must be used for local actuation only. **Returns**

- Last on value of the given multilevel switch.

cc_multilevel_switch_get_min_value

```
uint8_t cc_multilevel_switch_get_min_value (void )
```

Returns the min value of all multilevel switches.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- uint8_t Min value of all multilevel switches.

cc_multilevel_switch_get_max_value

```
uint8_t cc_multilevel_switch_get_max_value (void )
```

Returns the max value of all multilevel switches.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- uint8_t Max value of all multilevel switches.

cc_multilevel_switch_support_cb

```
void cc_multilevel_switch_support_cb (struct cc_multilevel_switch_t * p_switch)
```

Parameters

Type	Direction	Argument Name	Description
struct cc_multilevel_switch_t *	N/A	p_switch	

cc_multilevel_switch_support_config_get_switches

```
cc_multilevel_switch_t * cc_multilevel_switch_support_config_get_switches (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multilevel_switch_support_config_get_length_switches

```
uint8_t cc_multilevel_switch_support_config_get_length_switches (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multilevel_switch_support_config_get_default_duration

```
uint8_t cc_multilevel_switch_support_config_get_default_duration (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

cc_multilevel_switch_write

```
bool cc_multilevel_switch_write (uint8_t multilevel_component_id, cc_multilevel_switch_t * multilevel_switch)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	multilevel_component_id	
cc_multilevel_switch_t *	N/A	multilevel_switch	

cc_multilevel_switch_read

```
bool cc_multilevel_switch_read (uint8_t multilevel_component_id, cc_multilevel_switch_t * multilevel_switch)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	multilevel_component_id	
cc_multilevel_switch_t *	N/A	multilevel_switch	

cc_multilevel_switch_t

Public Attributes

RECEIVE_OPTION S_TYPE_EX	rxOpt
uint8_t	endpoint
s_Actuator	actuator

Public Attribute Documentation

rxOpt

RECEIVE_OPTIONS_TYPE_EX cc_multilevel_switch_t::rxOpt

endpoint

uint8_t cc_multilevel_switch_t::endpoint

actuator

s_Actuator cc_multilevel_switch_t::actuator

Notification

Notification

Command Class Notification enables a device to send notifications like "smoke detected" or "window opened".

CC Notification is typically used in sensor applications because it supports many different notification/sensor types.

CC Notification requires at least one notification to be configured by implementing the functions declared in [cc_notification_config_api.h](#). Simplicity Studio / SLC generates this configuration automatically based on user configuration in *.cc_config files.

This command class offers one single API function to be used by the application:

- [CC_Notification_TriggerAndTransmit\(\)](#)

The specification of CC Notification can be found in https://github.com/Z-Wave-Alliance/AWG/blob/main/source/application_command_classes/command_class_definitions/notification_command_class_version_3-8.rst.

Modules

[s_CC_notification_data_t_](#)

[_cc_notification_t](#)

[Configuration of CC Notification](#)

[Storage for CC Notification](#)

Enumerations

```
enum NOTIFICATION_EVENT {  
    NOTIFICATION_EVENT_NO_EVENT  
    NOTIFICATION_EVENT_UNKNOWN_EVENT = 0xFE  
    NOTIFICATION_EVENT_NO_EVENT  
    NOTIFICATION_EVENT_UNKNOWN_EVENT = 0xFE  
}  
General purpose notification event.
```

```
enum notification_type_t {  
    NOTIFICATION_TYPE_NONE  
    NOTIFICATION_TYPE_SMOKE_ALARM = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_SMOKE_ALARM)  
    NOTIFICATION_TYPE_CO_ALARM = (0xFF & ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_CO_ALARM)  
    NOTIFICATION_TYPE_CO2_ALARM = (0xFF & ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_CO2_ALARM)  
    NOTIFICATION_TYPE_HEAT_ALARM = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_HEAT_ALARM)  
    NOTIFICATION_TYPE_WATER_ALARM = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_WATER_ALARM)  
    NOTIFICATION_TYPE_ACCESS_CONTROL = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_ACCESS_CONTROL)  
    NOTIFICATION_TYPE_HOME_SECURITY = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_HOME_SECURITY)  
    NOTIFICATION_TYPE_POWER_MANAGEMENT = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_POWER_MANAGEMENT)  
    NOTIFICATION_TYPE_SYSTEM = (0xFF & ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_SYSTEM)  
    NOTIFICATION_TYPE_EMERGENCY_ALARM = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_EMERGENCY_ALARM)  
    NOTIFICATION_TYPE_CLOCK = (0xFF & ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_CLOCK)  
    NOTIFICATION_TYPE_MULTIDEVICE = (0xFF &  
    ICON_TYPE_SPECIFIC_SENSOR_NOTIFICATION_MULTIDEVICE)  
}  
Notification type (8 bit).  
  
enum NOTIFICATION_EVENT {  
    NOTIFICATION_EVENT_NO_EVENT  
    NOTIFICATION_EVENT_UNKNOWN_EVENT = 0xFE  
    NOTIFICATION_EVENT_NO_EVENT  
    NOTIFICATION_EVENT_UNKNOWN_EVENT = 0xFE  
}  
General purpose notification event.
```

Typedefs

```
typedef struct s_CC_notification_data_t  
s_CC_notification_data_t  
Struct used to pass operational data to TSE module.  
  
typedef struct s_CC_notification_data_t  
s_CC_notification_data_t  
Struct used to pass operational data to TSE module.  
  
typedef uint8_t notification_event_state  
  
typedef struct cc_notification_t  
_cc_notification_t
```

Functions

```
JOB_STATUS CC_Notification_TriggerAndTransmit(uint8_t index, uint8_t notificationEvent, uint8_t *pEvPar, uint8_t evParLen,  
void(*pCallback)(TRANSMISSION_RESULT *pTransmissionResult), bool tse)  
Triggers notification event and sends it to dedicated Agi profile.  
  
void notification_init()  
Inits notification module Do not call it directly.  
  
void notification_reset()  
Resets notification module Do not call it directly.
```

e_cmd_handler_re turn_code_t	handleAppNotificationSet (notification_type_t notificationType, NOTIFICATION_STATUS notificationStatus, uint8_t endpoint) Application specific Notification Set cmd handler.
uint8_t	cc_notification_supported_notifications_bitmask (uint8_t *pBitMaskArray, uint8_t bBitMaskLen, uint8_t endpoint) Fills in Bit Mask with supported Notification Types.
uint8_t	cc_notification_supported_events_bitmask (notification_type_t type, uint8_t endpoint, uint8_t *pBitMaskArray) Fills in Bit Mask with supported events for given notification.
bool	notification_report_get_data (notification_type_t *type, uint8_t endpoint, uint8_t *event, NOTIFICATION_STATUS *status) Prepares data for Notification Report command as response to Notification Get.
JOB_STATUS	CmdClassNotificationReport (uint8_t notification_index, uint8_t *pEvPar, uint8_t evParLen, void(*pCallback)(TRANSMISSION_RESULT *pTransmissionResult)) Transmits a Notification Report.
void	CC.Notification_report_stx (zaf_tx_options_t *tx_options, void *pData) Send report via lifeLine when a change happens.

Macros

#define	NOTIFICATION_BITMASK_ARRAY_LENGTH 32 the maximum length of the notification bitmask array
#define	NOTIFICATION_BITMASK_ARRAY_LENGTH 32 the maximum length of the notification bitmask array

Enumeration Documentation

NOTIFICATION_EVENT

NOTIFICATION_EVENT

General purpose notification event.

Enumerator	
NOTIFICATION_EVENT_NO_EVENT	
NOTIFICATION_EVENT_UNKNOWN_EVENT	
NOTIFICATION_EVENT_NO_EVENT	
NOTIFICATION_EVENT_UNKNOWN_EVENT	

notification_type_t

notification_type_t

Notification type (8 bit).

Enumerator	
NOTIFICATION_TYPE_NONE	
NOTIFICATION_TYPE_SMOKE_ALARM	
NOTIFICATION_TYPE_CO_ALARM	
NOTIFICATION_TYPE_CO2_ALARM	

NOTIFICATION_TYPE_HEAT_ALARM	
NOTIFICATION_TYPE_WATER_ALARM	
NOTIFICATION_TYPE_ACCESS_CONTROL	
NOTIFICATION_TYPE_HOME_SECURITY	
NOTIFICATION_TYPE_POWER_MANAGEMENT	
NOTIFICATION_TYPE_SYSTEM	
NOTIFICATION_TYPE_EMERGENCY_ALARM	
NOTIFICATION_TYPE_CLOCK	
NOTIFICATION_TYPE_MULTIDEVICE	

NOTIFICATION_EVENT

NOTIFICATION_EVENT

General purpose notification event.

Enumerator	
NOTIFICATION_EVENT_NO_EVENT	
NOTIFICATION_EVENT_UNKNOWN_EVENT	
NOTIFICATION_EVENT_NO_EVENT	
NOTIFICATION_EVENT_UNKNOWN_EVENT	

Typedef Documentation

s_CC_notification_data_t

typedef struct s_CC_notification_data_t_ s_CC_notification_data_t

Struct used to pass operational data to TSE module.

s_CC_notification_data_t

typedef struct s_CC_notification_data_t_ s_CC_notification_data_t

Struct used to pass operational data to TSE module.

notification_event_state

typedef uint8_t notification_event_state

cc_notification_t

typedef struct _cc_notification_t cc_notification_t

Function Documentation

CC_Notification_TriggerAndTransmit

```
JOB_STATUS CC_Notification_TriggerAndTransmit (uint8_t index, uint8_t notificationEvent, uint8_t * pEvPar, uint8_t evParLen, void(*)(TRANSMISSION\_RESULT *pTransmissionResult) pCallback, bool tse)
```

Triggers notification event and sends it to dedicated Agi profile.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	notification index.
uint8_t	[in]	notificationEvent	Event triggered.
uint8_t *	[in]	pEvPar	point to event parameters.
uint8_t	[in]	evParLen	length of event parameters.
void(*)(TRANSMISSION_RESULT *pTransmissionResult)	[out]	pCallback	callback function returning state on job
bool	[in]	tse	True or false, depending if TSE should be triggered, or not

Returns

- JOB_STATUS

notification_init

```
void notification_init ()
```

Init's notification module Do not call it directly.

Used only by CC_Notification.c

notification_reset

```
void notification_reset ()
```

Resets notification module Do not call it directly.

Used only by CC_Notification.c

handleAppNotificationSet

```
e_cmd_handler_return_code_t handleAppNotificationSet (notification\_type\_t notificationType, NOTIFICATION\_STATUS notificationStatus, uint8_t endpoint)
```

Application specific Notification Set cmd handler.

Parameters

Type	Direction	Argument Name	Description
notification_type_t	[in]	notificationType	notification type
NOTIFICATION_STATUS	[in]	notificationStatus	notification status of type NOTIFICATION_STATUS
uint8_t	[in]	endpoint	is the destination endpoint

Returns

- command handler return code

cc_notification_supported_notifications_bitmask

```
uint8_t cc_notification_supported_notifications_bitmask (uint8_t * pBitMaskArray, uint8_t bBitMaskLen, uint8_t endpoint)
```

Fills in Bit Mask with supported Notification Types.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	pBitMaskArray	The Bit Mask fields describe the supported Notification Type(s) by the device
uint8_t	[in]	bBitMaskLen	The length of the Bit Mask fields
uint8_t	[in]	endpoint	is the destination endpoint

Used to prepare data for Notification Supported Report Command as response to Notification Supported Get Command.

Returns

- Number of Bit Masks fields (1-31) used, in bytes.

cc_notification_supported_events_bitmask

```
uint8_t cc_notification_supported_events_bitmask (notification\_type\_t type, uint8_t endpoint, uint8_t * pBitMaskArray)
```

Fills in Bit Mask with supported events for given notification.

Parameters

Type	Direction	Argument Name	Description
notification_type_t	[in]	type	Notification type
uint8_t	[in]	endpoint	Notification Endpoint
uint8_t *	[out]	pBitMaskArray	The Bit Mask of supported Events within the requested Notification Type.

Used to prepare data for Event Supported Report Command

Finds notification based on {type, endpoint} values and creates bitmask with supported events. **See Also**

- cc_notification_config.c

Returns

- Number of Bit Masks fields (1-31) used in bytes

notification_report_get_data

```
bool notification_report_get_data (notification_type_t * type, uint8_t endpoint, uint8_t * event, NOTIFICATION_STATUS * status)
```

Prepares data for Notification Report command as response to Notification Get.

Parameters

Type	Direction	Argument Name	Description
notification_type_t *	[inout]	type	Notification Type
uint8_t	[in]	endpoint	Notification endpoint
uint8_t *	[inout]	event	If specified, current state of notification
NOTIFICATION_STATUS *	[out]	status	Current Notification status - On/Off

Returns

- true if notification with {type,endpoint} exists, false if input data wasn't correct

CmdClassNotificationReport

```
JOB_STATUS CmdClassNotificationReport (uint8_t notification_index, uint8_t * pEvPar, uint8_t evParLen, void(*) (TRANSMISSION_RESULT *pTransmissionResult) pCallback)
```

Transmits a Notification Report.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	notification_index	index of notification to be reported in notifications array
uint8_t *	N/A	pEvPar	List of parameters to be reported
uint8_t	N/A	evParLen	Length of event parameters list
void(*) (TRANSMISSION_RESULT *pTransmissionResult)	N/A	pCallback	callback function returning state on job

See SDS11060.doc "Table of defined Notification Types & Events". **Returns**

- JOB_STATUS

CC_Notification_report_stx

```
void CC_Notification_report_stx (zaf_tx_options_t * tx_options, void * pData)
```

Send report via lifeLine when a change happens.

Parameters

Type	Direction	Argument Name	Description
zaf_tx_options_t *	N/A	tx_options	txOptions
void *	N/A	pData	Command payload for the report

Callback used by TSE module. Refer to [ZAF_TSE.h](#) for more details.

s_CC_notification_data_t_

Struct used to pass operational data to TSE module.

Public Attributes

RECEIVE_OPTION S_TYPE_EX	rxOptions rxOptions
uint8_t	index
uint8_t *	pEventParameters index of notification in notifications array
uint8_t	eventParamLength Pointer to event parameters.

Public Attribute Documentation

rxOptions

RECEIVE_OPTIONS_TYPE_EX s_CC_notification_data_t::rxOptions

rxOptions

index

uint8_t s_CC_notification_data_t::index

pEventParameters

uint8_t * s_CC_notification_data_t::pEventParameters

index of notification in notifications array

eventParamLength

uint8_t s_CC_notification_data_t::eventParamLength

Pointer to event parameters.

_cc_notification_t

Public Attributes

RECEIVE_OPTION S_TYPE_EX	rxOpt
const uint8_t	endpoint
const notification_type_ t	type
const agi_profile_t	agi_profile
const notification_event _state *	events
const uint8_t	event_count
notification_event _state	current_event
NOTIFICATION_S TATUS	status Current event state.
const notification_event _state	events

Public Attribute Documentation

rxOpt

```
RECEIVE_OPTIONS_TYPE_EX _cc_notification_t::rxOpt
```

endpoint

```
const uint8_t _cc_notification_t::endpoint
```

type

```
const notification_type_t _cc_notification_t::type
```

agi_profile

```
const agi_profile_t _cc_notification_t::agi_profile
```

events

```
const notification_event_state* _cc_notification_t::events
```

event_count

```
const uint8_t _cc_notification_t::event_count
```

current_event

```
notification_event_state _cc_notification_t::current_event
```

status

```
NOTIFICATION_STATUS _cc_notification_t::status
```

Current event state.

events

```
const notification_event_state _cc_notification_t::events[CC_NOTIFICATION_MAX_EVENTS_PER_NOTIFICATION]
```

Configuration of CC Notification

Configuration of CC Notification

Modules

`_cc_notification_t`

Enumerations

```
enum NOTIFICATION_STATUS {  
    NOTIFICATION_STATUS_UNSOLICIT_DEACTIVATED = 0x00  
    NOTIFICATION_STATUS_UNSOLICIT_ACTIVATED = 0xFF  
}  
Notification status.
```

Typedefs

```
typedef uint8_t notification_event_state  
  
typedef struct _cc_notification_t
```

Functions

```
cc_notification_t * cc_notification_get_config(void)  
Returns the address of the first element in the notification array.  
  
uint8_t cc_notification_get_config_length(void)  
Returns the number of elements in the notification array.  
  
uint8_t cc_notification_get_supported_events_length(void)  
Returns the total number of supported events for all notifications.  
  
cc_notification_t * cc_notification_get(uint8_t index)  
Returns the address of the notification located at the given index in the notification array.  
  
int8_t cc_notification_get_index_by_type_and_endpoint(notification_type_t type, uint8_t endpoint)  
Returns the index of a notification based on given type and endpoint.  
  
notification_type_t cc_notification_get_type(uint8_t index)  
Returns the notification type from the given index.  
  
notification_event_state cc_notification_get_current_event(uint8_t index)  
Returns the current event of a notification given by the index in the notification array.  
  
agi_profile_t cc_notification_get_agi_profile(uint8_t index)  
Returns the AGI profile of a notification given by the index in the notification array.  
  
uint8_t cc_notification_get_endpoint(uint8_t index)  
Returns endpoint of a notification given by the index in the notification array.
```

NOTIFICATION_STATUS	<code>cc_notification_get_status</code> (uint8_t index) Returns the status (on/off) of a notification given by the index in the notification array.
bool	<code>cc_notification_is_type_supported</code> (notification_type_t type) Returns whether a notification exists in the notification array based on a given notification type.
int8_t	<code>cc_notification_find_event</code> (uint8_t event, uint8_t index) Returns the index of a given event, if supported, for a given notification index.
uint8_t	<code>cc_notification_config_get_number_of_events</code> (uint8_t index) Returns the number of configured events for a given notification index.

Enumeration Documentation

NOTIFICATION_STATUS

NOTIFICATION_STATUS

Notification status.

Enumerator

NOTIFICATION_STATUS_UNSOLICIT_DEACTIVATED	
NOTIFICATION_STATUS_UNSOLICIT_ACTIVATED	

Typedef Documentation

notification_event_state

<code>typedef uint8_t notification_event_state</code>

cc_notification_t

<code>typedef struct _cc_notification_t cc_notification_t</code>

Function Documentation

cc_notification_get_config

<code>cc_notification_t * cc_notification_get_config (void)</code>

Returns the address of the first element in the notification array.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- First element in the notification array.

cc_notification_get_config_length

```
uint8_t cc_notification_get_config_length (void )
```

Returns the number of elements in the notification array.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Number of elements in the notification array.

cc_notification_get_supported_events_length

```
uint8_t cc_notification_get_supported_events_length (void )
```

Returns the total number of supported events for all notifications.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Total number of supported events.

cc_notification_get

```
cc_notification_t * cc_notification_get (uint8_t index)
```

Returns the address of the notification located at the given index in the notification array.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Index in the notification array.

Returns

- Address of notification.

cc_notification_get_index_by_type_and_endpoint

```
int8_t cc_notification_get_index_by_type_and_endpoint (notification_type_t type, uint8_t endpoint)
```

Returns the index of a notification based on given type and endpoint.

Parameters

Type	Direction	Argument Name	Description
notification_type_t	[in]	type	Notification type.
uint8_t	[in]	endpoint	Endpoint.

Returns

- Index of notification.

cc_notification_get_type

```
notification_type_t cc_notification_get_type (uint8_t index)
```

Returns the notification type from the given index.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Index in the notification array.

Returns

- Notification type.

cc_notification_get_current_event

```
notification_event_state cc_notification_get_current_event (uint8_t index)
```

Returns the current event of a notification given by the index in the notification array.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Index in the notification array.

Returns

- Current event.

cc_notification_get_agi_profile

```
agi_profile_t cc_notification_get_agi_profile (uint8_t index)
```

Returns the AGI profile of a notification given by the index in the notification array.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Index in the notification array.

Returns

- AGI profile.

cc_notification_get_endpoint

```
uint8_t cc_notification_get_endpoint (uint8_t index)
```

Returns endpoint of a notification given by the index in the notification array.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Index in the notification array.

Returns

- Endpoint.

cc_notification_get_status

```
NOTIFICATION_STATUS cc_notification_get_status (uint8_t index)
```

Returns the status (on/off) of a notification given by the index in the notification array.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	index	Index in the notification array.

Returns

- Returns [NOTIFICATION_STATUS_UNSOLICIT_DEACTIVATED](#) if the notification is disabled.
- Returns [NOTIFICATION_STATUS_UNSOLICIT_ACTIVATED](#) if the notification is enabled.

cc_notification_is_type_supported

```
bool cc_notification_is_type_supported (notification\_type\_t type)
```

Returns whether a notification exists in the notification array based on a given notification type.

Parameters

Type	Direction	Argument Name	Description
notification_type_t	[in]	type	Notification type.

This function is used to check whether incoming CC Notification commands must be discarded or not. **Returns**

- true if available,

cc_notification_find_event

```
int8_t cc_notification_find_event (uint8_t event, uint8_t index)
```

Returns the index of a given event, if supported, for a given notification index.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	event	Event to be found.
uint8_t	[in]	index	Notification index in notifications array

Returns

- Returns event index if found
- Returns -1 if the event is not supported for the given notification index.

cc_notification_config_get_number_of_events

```
uint8_t cc_notification_config_get_number_of_events (uint8_t index)
```

Returns the number of configured events for a given notification index.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	index	Notification index.

Returns

- Number of configured events.

Storage for CC Notification

Storage for CC Notification

Functions

- bool

`cc_notification_write`(void)
Writes CC Notification data.
- bool

`cc_notification_read`(void)
Reads CC Notification data.

Function Documentation

cc_notification_write

bool cc_notification_write (void)

Writes CC Notification data.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- true on success, false otherwise.

cc_notification_read

bool cc_notification_read (void)

Reads CC Notification data.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- true on success, false otherwise.

Supervision

Supervision

Description

CC Supervision is built into the Application Framework and handles Supervision communication on S2 encapsulated frames. Supervision is only supported for Set and Report commands.

Command Class Supervision is default configuration to not support more Supervision reports on a Supervision Get command. In Supervision Get command is 'more status updates' field set to CC_SUPERVISION_STATUS_UPDATES_NOT_SUPPORTED.

Default configuration

The default setting is that CC Supervision supports only one Supervision Report per Supervision Get command. In Supervision Get command, the 'more status updates' field is set to CC_SUPERVISION_STATUS_UPDATES_NOT_SUPPORTED.

Handle more Supervision Reports

Device has the possibility to inform destination node that there is an operation in progress. Example is Wall controller with a display showing that a device is working (CC_SUPERVISION_STATUS_WORKING) until target value is reached (CC_SUPERVISION_STATUS_SUCCESS). Please see [cc_supervision_report_recived_handler\(\)](#) for more details.

Control Supervision Reports

Device has the possibility to send more Supervision Reports to report ongoing Request job. Example application Doorlock Keypad reports when Doorlock operation is started and also sends a new report when it's finished. For this purpose, `cc_supervision_get_received_handler()` can be used. Please see [cc_supervision_get_received_handler\(\)](#) for more details.

Modules

[SUPERVISION_GET_RECEIVED_HANDLER_ARGS](#)

[_cc_supervision_report_event](#)

[Supervision Events](#)

Enumerations

```
enum    cc_supervision_more_status_updates_t {  
        CC_SUPERVISION_MORE_STATUS_UPDATES_THIS_IS_LAST  
        CC_SUPERVISION_MORE_STATUS_UPDATES_REPORTS_TO_FOLLOW  
    }
```

Enum type is used in CC SUPERVISION_REPORT to allow a receiving node to advertise application status updates in future Supervision Report Commands.

```
enum \_SUPERVISION\_STATUS\_{
    CC_SUPERVISION_STATUS_NOT_SUPPORTED
    CC_SUPERVISION_STATUS_WORKING
    CC_SUPERVISION_STATUS_FAIL
    CC_SUPERVISION_STATUS_CANCEL
    CC_SUPERVISION_STATUS_NOT_FOUND
    CC_SUPERVISION_STATUS_SUCCESS = 0xFF
}
Status of Supervision.
```

Typedefs

```
typedef enum \_SUPERVISION\_S
TATUS\_
```

[cc_supervision_status_t](#)
Status of Supervision.

```
typedef struct \_cc\_supervision\_r
eport\_event
```

[cc_supervision_report_event](#)
Structure used by `supervision_event_handler()` to process upcoming events Registered for CC Supervision.

Functions

```
bool CmdClassSupervisionReportSend(zaf_tx_options_t *tx_options, uint8_t properties, cc_supervision_status_t
status, uint8_t duration)
CmdClassSupervisionReportSend.

void CommandClassSupervisionGetAdd(ZW_SUPERVISION_GET_FRAME *pPayload)
Creates payload for Supervision Get command by calling CommandClassSupervisionGetWrite and updates supervision
session ID.

void CommandClassSupervisionGetWrite(ZW_SUPERVISION_GET_FRAME *pbuf)
Generates payload for Supervision Get command.

void CommandClassSupervisionGetSetPayloadLength(ZW_SUPERVISION_GET_FRAME *pbuf, uint8_t payLoadlen)
Sets the payload length in a Supervision Get command.

uint8_t CommandClassSupervisionGetGetPayloadLength(ZW_SUPERVISION_GET_FRAME *pbuf)
Gets the length of encapsulated payload from the frame.

void cc\_supervision\_get\_received\_handler(SUPERVISION_GET_RECEIVED_HANDLER_ARGS *pArgs)
Invoked upon reception of a Supervision Get This function is weakly defined in CC_Supervision.c and can be defined in
the application if desired.

void cc\_supervision\_report\_recived\_handler(cc_supervision_status_t status, uint8_t duration)
Invoked upon reception of a Supervision Report This function is weakly defined in CC_Supervision.c and can be
defined in the application if desired.

bool cc\_supervision\_get\_default\_status\_updates\_enabled()
Returns whether transmission of status updates is enabled or disabled by default.
```

Macros

```
#define CC_SUPERVISION_EXTRACT_SESSION_ID (properties)
extract session id from properties bits [0..5]

#define CC_SUPERVISION_ADD_SESSION_ID (properties)
add session id from properties bits [0..5]
```

```
#define CC_SUPERVISION_EXTRACT_MORE_STATUS_UPDATE (status)
extract more status update field bit 7, CC SUPERVISION_REPORT

#define CC_SUPERVISION_ADD_MORE_STATUS_UPDATE (status)
status values: {0,1}.

#define CC_SUPERVISION_EXTRACT_STATUS_UPDATE (status)
extract more status update field bit 7, CC SUPERVISION_GET

#define CC_SUPERVISION_ADD_STATUS_UPDATE (status)
status values: {0,1}.
```

Enumeration Documentation

cc_supervision_more_status_updates_t

```
cc_supervision_more_status_updates_t
```

Enum type is used in CC SUPERVISION_REPORT to allow a receiving node to advertise application status updates in future Supervision Report Commands.

Enumerator	
CC_SUPERVISION_MORE_STATUS_UPDATES_THIS_IS_LAST	CC_SUPERVISION_MORE_STATUS_UPDATES_THIS_IS_LAST.
CC_SUPERVISION_MORE_STATUS_UPDATES_REPORTS_TO_FOLLOW	CC_SUPERVISION_MORE_STATUS_UPDATES_REPORTS_TO_FOLLOW.

_SUPERVISION_STATUS_

```
_SUPERVISION_STATUS_
```

Status of Supervision.

Enumerator	
CC_SUPERVISION_STATUS_NOT_SUPPORTED	SUPERVISION_STATUS_NOT_SUPPORTED.
CC_SUPERVISION_STATUS_WORKING	SUPERVISION_STATUS_WORKING.
CC_SUPERVISION_STATUS_FAIL	SUPERVISION_STATUS_FAIL.
CC_SUPERVISION_STATUS_CANCEL	CC Supervision shouldn't do anything with this frame, report will be sent from somewhere else.
CC_SUPERVISION_STATUS_NOT_FOUND	CC handler was not found in CC handler map.
CC_SUPERVISION_STATUS_SUCCESS	SUPERVISION_STATUS_SUCCESS.

Typedef Documentation

cc_supervision_status_t

```
typedef enum _SUPERVISION_STATUS_ cc_supervision_status_t
```

Status of Supervision.

cc_supervision_report_event

```
typedef struct _cc_supervision_report_event cc_supervision_report_event
```

Structure used by `supervision_event_handler()` to process upcoming events Registered for CC Supervision.

Function Documentation

CmdClassSupervisionReportSend

```
bool CmdClassSupervisionReportSend (zaf_tx_options_t * tx_options, uint8_t properties, cc_supervision_status_t status, uint8_t duration)
```

CmdClassSupervisionReportSend.

Parameters

Type	Direction	Argument Name	Description
zaf_tx_options_t *	[in]	tx_options	Transmit options of type zaf_tx_options_t
uint8_t	[in]	properties	includes: bit[0..5] Session ID bit[6] unused and bit[7] more status updates. This field is used to advertise if more Supervision Reports follow for the actual Session ID. [0] last report, [1] more report follow.
cc_supervision_status_t	[in]	status	Supervision status.
uint8_t	[in]	duration	The duration must comply with the following: 0x00 0 seconds. (Already at the Target Value.) 0x01-0x7F 1 second (0x01) to 127 seconds (0x7F) in 1 second resolution. 0x80-0xFD 1 minute (0x80) to 126 minutes (0xFD) in 1 minute resolution. 0xFE Unknown duration 0xFF Reserved

Returns

- true if the frame was enqueued else false

CommandClassSupervisionGetAdd

```
void CommandClassSupervisionGetAdd (ZW_SUPERVISION_GET_FRAME * pPayload)
```

Creates payload for Supervision Get command by calling [CommandClassSupervisionGetWrite](#) and updates supervision session ID.

Parameters

Type	Direction	Argument Name	Description
ZW_SUPERVISION_GET_FRAME *	[inout]	pPayload	pointer to supervision get payload

Used by Framework and should not be used be application.

CommandClassSupervisionGetWrite

```
void CommandClassSupervisionGetWrite (ZW_SUPERVISION_GET_FRAME * pbuf)
```

Generates payload for Supervision Get command.

Parameters

Type	Direction	Argument Name	Description
ZW_SUPERVISION_GET_FRAME *	[inout]	pbuf	Pointer to supervision get payload

Used by Framework and should not be used be application.

CommandClassSupervisionGetSetPayloadLength

```
void CommandClassSupervisionGetSetPayloadLength (ZW_SUPERVISION_GET_FRAME * pbuf, uint8_t payLoadlen)
```

Sets the payload length in a Supervision Get command.

Parameters

Type	Direction	Argument Name	Description
ZW_SUPERVISION_GET_FRAME *	[in]	pbuf	Pointer to supervision get payload
uint8_t	[in]	payLoadlen	Payload length to be set.

Used by Framework when sending supervision-encapsulated frames

CommandClassSupervisionGetGetPayloadLength

```
uint8_t CommandClassSupervisionGetGetPayloadLength (ZW_SUPERVISION_GET_FRAME * pbuf)
```

Gets the length of encapsulated payload from the frame.

Parameters

Type	Direction	Argument Name	Description
ZW_SUPERVISION_GET_FRAME *	N/A	pbuf	Frame that contains the encapsulated command

Used by framework to calculate the length of the payload in supervision-encapsulated frames.

Returns

- length of the payload

cc_supervision_get_received_handler

```
void cc_supervision_get_received_handler (SUPERVISION_GET_RECEIVED_HANDLER_ARGS * pArgs)
```

Invoked upon reception of a Supervision Get This function is weakly defined in CC_Supervision.c and can be defined in the application if desired.

Parameters

Type	Direction	Argument Name	Description
SUPERVISION_GET_RECEIVED_HANDLER_ARGS *	N/A	pArgs	parameters for the handler

cc_supervision_report_recived_handler

```
void cc_supervision_report_recived_handler (cc_supervision_status_t status, uint8_t duration)
```

Invoked upon reception of a Supervision Report This function is weakly defined in CC_Supervision.c and can be defined in the application if desired.

Parameters

Type	Direction	Argument Name	Description
cc_supervision_status_t	N/A	status	the required status
uint8_t	N/A	duration	the required duration

cc_supervision_get_default_status_updates_enabled

```
bool cc_supervision_get_default_status_updates_enabled ()
```

Returns whether transmission of status updates is enabled or disabled by default.

Used by the Supervion Command Class initialization

Returns

- true if enabled, false if disabled

SUPERVISION_GET_RECEIVED_HANDLER_ARGS

This struct is used for the arguments in cc_supervision_get_receive_handler function pointer in init_and_reset.

Public Attributes

uint8_t	cmdClass	Input.
uint8_t	cmd	Input.
uint8_t	properties1	Input: bit[0..5] session id, bit[7] status update disable/enable.
cc_supervision_status_t	status	Output.
uint8_t	duration	Output.
RECEIVE_OPTION_S_TYPE_EX *	rxOpt	Input.

Public Attribute Documentation

cmdClass

uint8_t SUPERVISION_GET_RECEIVED_HANDLER_ARGS::cmdClass

Input.

cmd

uint8_t SUPERVISION_GET_RECEIVED_HANDLER_ARGS::cmd

Input.

properties1

uint8_t SUPERVISION_GET_RECEIVED_HANDLER_ARGS::properties1

Input: bit[0..5] session id, bit[7] status update disable/enable.

Output: bit[0..5] session id, bit[7] more status updates of type cc_supervision_more_status_updates_t

status

```
cc_supervision_status_t SUPERVISION_GET_RECEIVED_HANDLER_ARGS::status
```

Output.

duration

```
uint8_t SUPERVISION_GET_RECEIVED_HANDLER_ARGS::duration
```

Output.

rxOpt

```
RECEIVE_OPTIONS_TYPE_EX* SUPERVISION_GET_RECEIVED_HANDLER_ARGS::rxOpt
```

Input.

_cc_supervision_report_event

Structure used by supervision_event_handler() to process upcoming events Registered for CC Supervision.

Public Attributes

zaf_tx_options_t	tx_options	
uint8_t	session_id	TX data to be used to send Supervision Report.
bool	status_update	Supervision session Id.
uint8_t	duration	Status Update. True if more reports will follow.

Public Attribute Documentation

tx_options

zaf_tx_options_t _cc_supervision_report_event::tx_options

session_id

uint8_t _cc_supervision_report_event::session_id

TX data to be used to send Supervision Report.

status_update

bool _cc_supervision_report_event::status_update

Supervision session Id.

duration

uint8_t _cc_supervision_report_event::duration

Status Update. True if more reports will follow.

Supervision Events

Supervision Events

Events supported by Supervision CC App or other CC can trigger these events by calling [zaf_event_distributor_enqueue_cc_event](#).

Macros

```
#define    CC_SUPERVISION_EVENT_SUCCESS 1

#define    CC_SUPERVISION_EVENT_FAIL 2

#define    CC_SUPERVISION_EVENT_WORKING 3
```

UserCode

UserCode

Modules

- [SUserCode](#)
- [s_CC_userCode_data_t](#)
- [UserCodeEvents](#)

Enumerations

```
enum user_id_status_t {  
    USER_ID_AVAILABLE = 0x00  
    USER_ID_OCCUPIED = 0x01  
    USER_ID_RESERVED = 0x02  
    USER_ID_NO_STATUS = 0xFE  
}
```

Typedefs

```
typedef USER_ID_STATUS  
user_id_status_t  
  
typedef struct SUserCode  
SUserCode  
  
typedef struct s_CC_userCode_data_t  
s_CC_userCode_data_t  
Struct used to pass operational data to TSE module.
```

Functions

```
void CC_UserCode_Migrate()  
  
bool CC_UserCode_Write(uint8_t identifier, SUserCode *userCodeData)  
  
bool CC_UserCode_Read(uint8_t identifier, SUserCode *userCodeData)  
  
bool CC_UserCode_EraseAllUserCodes(void)  
  
e_cmd_handler_return_code_t CC_UserCode_Set_handler(uint8_t identifier, USER_ID_STATUS id, uint8_t *pUserCode, uint8_t len, uint16_t  
modifierNodeId)  
The User Code Set Command used to set a User Code in the device.  
  
bool CC_UserCode_getId_handler(uint8_t identifier, USER_ID_STATUS *pId, uint8_t endpoint)  
Determines the status of a User ID.  
  
bool CC_UserCode_Report_handler(uint8_t identifier, uint8_t *pUserCode, size_t *pLen, uint8_t endpoint)  
The User Code Report Command can be used by e.g.  
  
uint8_t CC_UserCode_UsersNumberReport_handler(uint8_t endpoint)  
The Users Number Report Command used to report the maximum number of USER CODES the given node supports.
```

- void

[CC_UserCode_reset_data](#)(void)

Resets the data used by the command class.
- void

[CC_UserCode_set_usercode](#)(char *new_user_code)

Set the user code to a new value.
- JOB_STATUS

[CC_UserCode_SupportReport](#)(AGI_PROFILE *pProfile, uint8_t sourceEndpoint, uint8_t userIdentifier, uint8_t userIdStatus, uint8_t *pUserCode, uint8_t userCodeLen, void(*pCallback)(TRANSMISSION_RESULT *pTransmissionResult))

Send a Command Class User code support report.
- bool

[CC_UserCode_Validate](#)(uint8_t identifier, const uint8_t *pCode, uint8_t len)

Macros

- #define

USERCODE_MIN_LEN 4

Minimum length of a user code as defined in SDS12652.
- #define

USERCODE_MAX_LEN 10

Maximum length of a user code as defined in SDS12652.

Enumeration Documentation

user_id_status_t

```
user_id_status_t
```

User ID Status.

The User ID Status field indicates the state of the User Identifier. All other values not mentioned in below list are reserved for future implementation.

Hex	Description
00	Available (not set)
01	Occupied
02	Reserved by administrator
FE	Status not available

Enumerator	
USER_ID_AVAILABLE	Available (not set)
USER_ID_OCCUPIED	Occupied.
USER_ID_RESERVED	Reserved by administrator.
USER_ID_NO_STATUS	Status not available.

Typedef Documentation

USER_ID_STATUS

```
typedef user_id_status_t USER_ID_STATUS
```

SUserCode


```
typedef struct SUserCode SUserCode
```

s_CC_userCode_data_t

```
typedef struct s_CC_userCode_data_t_ s_CC_userCode_data_t
```

Struct used to pass operational data to TSE module.

Function Documentation

CC_UserCode_Migrate

```
void CC_UserCode_Migrate ()
```

CC_UserCode_Write

```
bool CC_UserCode_Write (uint8_t identifier, SUserCode * userCodeData)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	identifier	
SUserCode *	N/A	userCodeData	

CC_UserCode_Read

```
bool CC_UserCode_Read (uint8_t identifier, SUserCode * userCodeData)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	identifier	
SUserCode *	N/A	userCodeData	

CC_UserCode_EraseAllUserCodes

```
bool CC_UserCode_EraseAllUserCodes (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

CC_UserCode_Set_handler

```
e_cmd_handler_return_code_t CC_UserCode_Set_handler (uint8_t identifier, USER\_ID\_STATUS id, uint8_t * pUserCode,
uint8_t len, uint16_t modifierNodeId)
```

The User Code Set Command used to set a User Code in the device.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	identifier	User Identifier.
USER_ID_STATUS	[in]	id	user Id status.
uint8_t *	[in]	pUserCode	pointer to UserCode data.
uint8_t	[in]	len	UserCode data.
uint16_t	[in]	modifierNodeId	Node ID of the requesting node.

Returns

- command handler return code

CC_UserCode_getId_handler

```
bool CC_UserCode_getId_handler (uint8_t identifier, USER\_ID\_STATUS * pId, uint8_t endpoint)
```

Determines the status of a User ID.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	identifier	User Identifier.
USER_ID_STATUS *	[out]	pId	pointer to return Id.
uint8_t	[in]	endpoint	is the destination endpoint

Returns

- status valid boolean.

CC_UserCode_Report_handler

```
bool CC_UserCode_Report_handler (uint8_t identifier, uint8_t * pUserCode, size_t * pLen, uint8_t endpoint)
```

The User Code Report Command can be used by e.g.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	identifier	User Identifier.
uint8_t *	[out]	pUserCode	pointer to UserCode data.
size_t *	[out]	pLen	length UserCode data.
uint8_t	[in]	endpoint	is the destination endpoint

a door lock device to send a report either unsolicited or requested by the User Code Get Command. **Returns**

- status valid boolean.

CC_UserCode_UsersNumberReport_handler

```
uint8_t CC_UserCode_UsersNumberReport_handler (uint8_t endpoint)
```

The Users Number Report Command used to report the maximum number of USER CODES the given node supports.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	is the destination endpoint

The Users Number Report Command can be send requested by the Users Number Get Command. **Returns**

- maximum number of USER CODES.

CC_UserCode_reset_data

```
void CC_UserCode_reset_data (void )
```

Resets the data used by the command class.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

CC_UserCode_set_usercode

```
void CC_UserCode_set_usercode (char * new_user_code)
```

Set the user code to a new value.

Parameters

Type	Direction	Argument Name	Description
char *	[in]	new_user_code	The new user code, must be under the length of USERCODE_MAX_LEN

CC_UserCode_SupportReport

```
JOB_STATUS CC_UserCode_SupportReport (AGL\_PROFILE * pProfile, uint8_t sourceEndpoint, uint8_t userIdentifier, uint8_t  
userIdStatus, uint8_t * pUserCode, uint8_t userCodeLen, void(*)(TRANSMISSION\_RESULT *pTransmissionResult)  
pCallback)
```

Send a Command Class User code support report.

Parameters

Type	Direction	Argument Name	Description
AGI_PROFILE *	[in]	pProfile	pointer to AGI profile
uint8_t	[in]	sourceEndpoint	source endpoint
uint8_t	[in]	userIdentifier	user identifier
uint8_t	[in]	userIdStatus	user Id status
uint8_t *	[in]	pUserCode	user code
uint8_t	[in]	userCodeLen	length of user code
void(*)(TRANSMISSION_RESULT *pTransmissionResult)	[out]	pCallback	callback function returning status destination node receive job.

Returns

- status on the job.

CC_UserCode_Validate

```
bool CC_UserCode_Validate (uint8_t identifier, const uint8_t * pCode, uint8_t len)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	identifier	
const uint8_t *	N/A	pCode	
uint8_t	N/A	len	

SUserCode

Public Attributes

USER_ID_STATUS	user_id_status
uint8_t	userCode
uint8_t	userCodeLen

Public Attribute Documentation

user_id_status

USER_ID_STATUS SUserCode::user_id_status

userCode

uint8_t SUserCode::userCode[USERCODE_MAX_LEN]

userCodeLen

uint8_t SUserCode::userCodeLen

s_CC_userCode_data_t_

Struct used to pass operational data to TSE module.

Public Attributes

RECEIVE_OPTION	rxOptions
S_TYPE_EX	rxOptions
uint8_t	userIdentifier

Public Attribute Documentation

rxOptions

RECEIVE_OPTIONS_TYPE_EX s_CC_userCode_data_t::rxOptions

rxOptions

userIdentifier

uint8_t s_CC_userCode_data_t::userIdentifier

UserCodeEvents

UserCodeEvents

Modules

[cc_user_code_event_validate_data_t](#)

Macros

```
#define CC_USER_CODE_EVENT_VALIDATE 1
#define CC_USER_CODE_EVENT_VALIDATE_VALID 2
#define CC_USER_CODE_EVENT_VALIDATE_INVALID 3
```

cc_user_code_event_validate_data_t

Public Attributes

uint8_t	id
uint8_t *	data
uint8_t	length

Public Attribute Documentation

id

uint8_t cc_user_code_event_validate_data_t::id

data

uint8_t* cc_user_code_event_validate_data_t::data

length

uint8_t cc_user_code_event_validate_data_t::length

UserCredential

UserCredential

Modules

[u3c_user_t_](#)

[u3c_credential_metadata_t_](#)

[u3c_credential_t_](#)

[u3c_credential_identifier_t_](#)

[u3c_event_data_validate_t_](#)

[u3c_event_data_learn_read_done_t_](#)

[u3c_event_data_learn_start_t_](#)

[u3c_admin_code_metadata_t_](#)

Enumerations

```
enum    u3c_db_operation_result_ {  
        U3C_DB_OPERATION_RESULT_SUCCESS  
        U3C_DB_OPERATION_RESULT_ERROR  
        U3C_DB_OPERATION_RESULT_ERROR_IO  
        U3C_DB_OPERATION_RESULT_ERROR_DUPLICATE  
        U3C_DB_OPERATION_RESULT_FAIL_DNE  
        U3C_DB_OPERATION_RESULT_FAIL_FULL  
        U3C_DB_OPERATION_RESULT_FAIL_OCCUPIED  
        U3C_DB_OPERATION_RESULT_FAIL_REASSIGN  
        U3C_DB_OPERATION_RESULT_FAIL_IDENTICAL  
    }  
  
enum    u3c_modifier_type_ {  
        MODIFIER_TYPE_DNE = CREDENTIAL_REPORT_DNE  
        MODIFIER_TYPE_UNKNOWN  
        MODIFIER_TYPE_Z_WAVE  
        MODIFIER_TYPE_LOCALLY  
        MODIFIER_TYPE_MOBILE_APP_OR_OTHER_IOT_TECHNOLOGY  
    }  
  
enum    u3c_credential_type_ {  
        CREDENTIAL_TYPE_NONE  
        CREDENTIAL_TYPE_PIN_CODE = CREDENTIAL_REPORT_PIN_CODE  
        CREDENTIAL_TYPE_PASSWORD  
        CREDENTIAL_TYPE_RFID_CODE  
        CREDENTIAL_TYPE_BLE  
        CREDENTIAL_TYPE_NFC  
        CREDENTIAL_TYPE_UWB  
        CREDENTIAL_TYPE_EYE_BIOMETRIC  
        CREDENTIAL_TYPE_FACE_BIOMETRIC  
        CREDENTIAL_TYPE_FINGER_BIOMETRIC  
        CREDENTIAL_TYPE_HAND_BIOMETRIC  
        CREDENTIAL_TYPE_UNSPECIFIED_BIOMETRIC  
        CREDENTIAL_TYPE_NUMBER_OF_TYPES  
    }
```

```

}
```

```

enum    _u3c_credential_report_type_t {
    CREDENTIAL_REP_TYPE_ADDED = CREDENTIAL_REPORT_ADDED
    CREDENTIAL_REP_TYPE_MODIFIED = CREDENTIAL_REPORT_MODIFIED
    CREDENTIAL_REP_TYPE_DELETED = CREDENTIAL_REPORT_DELETED
    CREDENTIAL_REP_TYPE_UNCHANGED = CREDENTIAL_REPORT_UNCHANGED
    CREDENTIAL_REP_TYPE_RESPONSE_TO_GET = CREDENTIAL_REPORT_RESPONSE_TO_GET
    CREDENTIAL_REP_TYPE_ADD_AGAINST_OCCUPIED = CREDENTIAL_REPORT_ADD_AGAINST_OCCUPIED
    CREDENTIAL_REP_TYPE_MODIF_AGAINST_EMPTY = CREDENTIAL_REPORT_MODIFY_AGAINST_EMPTY
    CREDENTIAL_REP_TYPE_DUPLICATE = CREDENTIAL_REPORT_DUPLICATE
    CREDENTIAL_REP_TYPE_MANUFACTURER_SECURITY_RULES =
    CREDENTIAL_REPORT_MANUFACTURER_SECURITY_RULES
    CREDENTIAL_REP_TYPE_WRONG_UUID = CREDENTIAL_REPORT_WRONG_USER_UNIQUE_IDENTIFIER
    CREDENTIAL_REP_TYPE_DUPLICATE_ADMIN_PIN_CODE =
    CREDENTIAL_REPORT_DUPLICATE_ADMIN_PIN_CODE
}

enum    u3c_credential_learn_status_t {
    CL_STATUS_STARTED = CREDENTIAL_LEARN_REPORT_STARTED
    CL_STATUS_SUCCESS = CREDENTIAL_LEARN_REPORT_SUCCESS
    CL_STATUS_ALREADY_IN_PROGRESS = CREDENTIAL_LEARN_REPORT_ALREADY_IN_PROGRESS
    CL_STATUS_ENDED_NOT_DUE_TO_TIMEOUT =
    CREDENTIAL_LEARN_REPORT_ENDED_NOT_DUE_TO_TIMEOUT
    CL_STATUS_TIMEOUT = CREDENTIAL_LEARN_REPORT_TIMEOUT
    CL_STATUS_STEP_RETRY = CREDENTIAL_LEARN_REPORT_CREDENTIAL_LEARN_STEP_RETRY
    CL_STATUS_INVALID_ADD_OPERATION_TYPE =
    CREDENTIAL_LEARN_REPORT_INVALID_CREDENTIAL_LEARN_ADD_OPERATION_TYPE
    CL_STATUS_INVALID_MODIFY_OPERATION_TYPE =
    CREDENTIAL_LEARN_REPORT_INVALID_CREDENTIAL_LEARN_MODIFY_OPERATION_TYPE
}

enum    u3c_user_credential_association_report_status_t {
    U3C_UCAR_STATUS_SUCCESS = USER_CREDENTIAL_ASSOCIATION_REPORT_SUCCESS
    U3C_UCAR_STATUS_CREDENTIAL_TYPE_INVALID =
    USER_CREDENTIAL_ASSOCIATION_REPORT_CREDENTIAL_TYPE_INVALID
    U3C_UCAR_STATUS_CREDENTIAL_SLOT_INVALID =
    USER_CREDENTIAL_ASSOCIATION_REPORT_CREDENTIAL_SLOT_INVALID
    U3C_UCAR_STATUS_CREDENTIAL_SLOT_EMPTY =
    USER_CREDENTIAL_ASSOCIATION_REPORT_CREDENTIAL_SLOT_EMPTY
    U3C_UCAR_STATUS_DESTINATION_USER_UNIQUE_IDENTIFIER_INVALID =
    USER_CREDENTIAL_ASSOCIATION_REPORT_DESTINATION_USER_UNIQUE_IDENTIFIER_INVALID
    U3C_UCAR_STATUS_DESTINATION_USER_UNIQUE_IDENTIFIER_NONEXISTENT =
    USER_CREDENTIAL_ASSOCIATION_REPORT_DESTINATION_USER_UNIQUE_IDENTIFIER_NONEXISTENT
}

enum    u3c_user_report_type_t {
    USER_REP_TYPE_ADDED = USER_REPORT_ADDED
    USER_REP_TYPE_MODIFIED = USER_REPORT_MODIFIED
    USER_REP_TYPE_DELETED = USER_REPORT_DELETED
    USER_REP_TYPE_UNCHANGED = USER_REPORT_UNCHANGED
    USER_REP_TYPE_RESPONSE_TO_GET = USER_REPORT_RESPONSE_TO_GET
    USER_REP_TYPE_ADD_AGAINST_OCCUPIED = USER_REPORT_ADD_AGAINST_OCCUPIED
    USER_REP_TYPE_MODIF_AGAINST_EMPTY = USER_REPORT_MODIFY_AGAINST_EMPTY
    USER_REP_TYPE_Z_EXP_MIN_INVALID = USER_REPORT_ZERO_EXPIRING_MINUTES_INVALID
}

```

```

enum    u3c_user_type_ {
        USER_TYPE_GENERAL = 0x00
        USER_TYPE_PROGRAMMING = 0x03
        USER_TYPE_NON_ACCESS = 0x04
        USER_TYPE_DURESS = 0x05
        USER_TYPE_DISPOSABLE = 0x06
        USER_TYPE_EXPIRING = 0x07
        USER_TYPE_REMOTE_ONLY = 0x09
        USER_TYPE_END
    }

enum    u3c_credential_rule_ {
        CREDENTIAL_RULE_SINGLE = 0x01
        CREDENTIAL_RULE_DUAL
        CREDENTIAL_RULE_TRIPLE
    }

enum    u3c_user_name_encoding_ {
        USER_NAME_ENCODING_STANDARD_ASCII
        USER_NAME_ENCODING_STANDARD_AND_OEM_EXTENDED_ASCII
        USER_NAME_ENCODING_UNICODE_UTF_16
    }

enum    _u3c_admin_code_operation_result_ {
        ADMIN_CODE_OPERATION_RESULT_MODIFIED = 0x01
        ADMIN_CODE_OPERATION_RESULT_UNMODIFIED = 0x03
        ADMIN_CODE_OPERATION_RESULT_GET_RESP = 0x04
        ADMIN_CODE_OPERATION_RESULT_FAIL_DUPLICATE_CRED = 0x07
        ADMIN_CODE_OPERATION_RESULT_FAIL_MANUF_RULE = 0x08
        ADMIN_CODE_OPERATION_RESULT_ERROR_AC_NOT_SUPPORTED = 0x0D
        ADMIN_CODE_OPERATION_RESULT_ERROR_ACD_NOT_SUPPORTED = 0x0E
        ADMIN_CODE_OPERATION_RESULT_ERROR_NODE = 0x0F
        ADMIN_CODE_OPERATION_RESULT_INTERNAL_DUPLICATE_AC = 0x13
        ADMIN_CODE_OPERATION_RESULT_INTERNAL_NONE = 0x14
        ADMIN_CODE_OPERATION_RESULT_INTERNAL_INVALID_LENGTH = 0x1F
        ADMIN_CODE_OPERATION_RESULT_INTERNAL_INVALID_CHAR = 0x2F
    }
    This operation result can optionally be injected into an Admin Code Report to pass information back to the controlling
    node, in lieu of another report type.

enum    u3c_operation_type_t {
        U3C_OPERATION_TYPE_ADD = 0
        U3C_OPERATION_TYPE_MODIFY = 1
        U3C_OPERATION_TYPE_DELETE = 2
    }

```

```
enum u3c_event_ {
    CC_USER_CREDENTIAL_EVENT_VALIDATE = 1
    CC_USER_CREDENTIAL_EVENT_VALIDATE_VALID = 2
    CC_USER_CREDENTIAL_EVENT_VALIDATE_INVALID = 3
    CC_USER_CREDENTIAL_EVENT_LEARN_START_REQUEST = 4
    CC_USER_CREDENTIAL_EVENT_LEARN_START = 5
    CC_USER_CREDENTIAL_EVENT_LEARN_STEP_START = 6
    CC_USER_CREDENTIAL_EVENT_LEARN_READ_DONE = 7
    CC_USER_CREDENTIAL_EVENT_LEARN_SUCCESS = 8
    CC_USER_CREDENTIAL_EVENT_LEARN_FAILED = 9
    CC_USER_CREDENTIAL_EVENT_LEARN_STEP_RETRY = 10
    CC_USER_CREDENTIAL_EVENT_LEARN_TIMEOUT = 11
    CC_USER_CREDENTIAL_EVENT_LEARN_CANCEL_REMOTE = 12
    CC_USER_CREDENTIAL_EVENT_LEARN_CANCEL_LOCAL = 13
}
```

Typedefs

```
typedef enum u3c_db_operation_result
u3c_db_operation_result_t
```

```
typedef enum u3c_modifier_type
u3c_modifier_type_t
```

```
typedef enum u3c_credential_type
u3c_credential_type_t
```

```
typedef enum u3c_credential_report_type_t
u3c_credential_report_type_t
```

```
typedef enum u3c_credential_learn_status_t
u3c_credential_learn_status_t
```

```
typedef enum u3c_user_report_type_t
u3c_user_report_type_t
```

```
typedef enum u3c_user_type
u3c_user_type_t
```

```
typedef enum u3c_credential_rule
u3c_credential_rule_t
```

```
typedef enum u3c_user_name_encoding
u3c_user_name_encoding_t
```

```
typedef enum u3c_admin_code_operation_result
u3c_admin_code_operation_result_t
```

This operation result can optionally be injected into an Admin Code Report to pass information back to the controlling node, in lieu of another report type.

```
typedef struct u3c_user_t
u3c_user_t
```

```
typedef struct u3c_credential_metadata_t
u3c_credential_metadata_t
```

typedef struct u3c_credential_t	u3c_credential_t
typedef struct u3c_credential_id entifier_t	u3c_credential_identifier_t
typedef struct u3c_event_data_v alidate_t	u3c_event_data_validate_t Payload for CC_USER_CREDENTIAL_EVENT_VALIDATE, received from the application.
typedef struct u3c_event_data_l earn_read_done_t -	u3c_event_data_learn_read_done_t Payload for CC_USER_CREDENTIAL_EVENT_LEARN_READ_DONE, passed to the application.
typedef struct u3c_event_data_l earn_start_t	u3c_event_data_learn_start_t Credential Learn information passed to application layer.
typedef struct u3c_admin_code_ metadata_t	u3c_admin_code_metadata_t
typedef enum u3c_event_ -	u3c_event_type

Functions

void	CC_UserCredential_factory_reset(void) Resets the User Credential Command Class database to a blank state, removing all existing users, credentials and the admin code.
void	CC_UserCredential_init_database(void) Initializes the User Credential Command Class database to a consistent state.
u3c_db_operation _result	CC_UserCredential_get_user(uint16_t unique_identifier, u3c_user_t *user, uint8_t *name) Copies the data for a given User from the database to an arbitrary location in memory.
uint16_t	CC_UserCredential_get_next_user(uint16_t unique_identifier) Finds the first or next available User Unique Identifier.
u3c_db_operation _result	CC_UserCredential_add_user(u3c_user_t *user, uint8_t *name)
u3c_db_operation _result	CC_UserCredential_modify_user(u3c_user_t *user, uint8_t *name) Modifies a user in the database.
u3c_db_operation _result	CC_UserCredential_delete_user(uint16_t user_unique_identifier)
u3c_db_operation _result	CC_UserCredential_get_credential(uint16_t user_unique_identifier, u3c_credential_type credential_type, uint16_t credential_slot, u3c_credential_metadata_t *credential_metadata, uint8_t *credential_data) Copies the data for a given Credential from the database to an arbitrary location in memory.
bool	CC_UserCredential_get_next_credential(uint16_t user_unique_identifier, u3c_credential_type credential_type, uint16_t credential_slot, u3c_credential_type *next_credential_type, uint16_t *next_credential_slot) Finds the first or next available Credential entry for a given User or Credential Type (set either one of these parameters to 0 to disable filtering by that property).
u3c_db_operation _result	CC_UserCredential_add_credential(u3c_credential_t *credential) Adds a new credential to the database.

u3c_db_operation _result	<code>CC_UserCredential_modify_credential(u3c_credential_t *credential)</code>
u3c_db_operation _result	<code>CC_UserCredential_delete_credential(u3c_credential_type credential_type, uint16_t credential_slot)</code> Deletes a Credential.
u3c_db_operation _result	<code>CC_UserCredential_move_credential(u3c_credential_type credential_type, uint16_t source_credential_slot, uint16_t destination_user_uid, uint16_t destination_credential_slot)</code> Moves a credential from one slot to another and/or assigns a credential from one User Unique Identifier to another one.
u3c_db_operation _result	<code>CC_UserCredential_get_admin_code_info(u3c_admin_code_metadata_t *code)</code> Retrieves the admin code information from the node.
u3c_db_operation _result	<code>CC_UserCredential_set_admin_code(u3c_admin_code_metadata_t *code)</code> Sets the admin code on the node, if supported.
u3c_db_operation _result	<code>CC_UserCredential_add_user_and_report(u3c_user_t *p_user, uint8_t *p_name, RECEIVE_OPTIONS_TYPE_EX *p_rx_options)</code> Tries to add a new user to the database and reports the result.
u3c_db_operation _result	<code>CC_UserCredential_modify_user_and_report(u3c_user_t *p_user, uint8_t *p_name, RECEIVE_OPTIONS_TYPE_EX *p_rx_options)</code> Tries to modify a user in the database and reports the result.
u3c_db_operation _result	<code>CC_UserCredential_delete_user_and_report(uint16_t uuid, RECEIVE_OPTIONS_TYPE_EX *p_rx_options)</code> Tries to delete a user from the database and reports the result.
u3c_db_operation _result	<code>CC_UserCredential_add_credential_and_report(u3c_credential_t *p_credential, RECEIVE_OPTIONS_TYPE_EX *p_rx_options)</code> Tries to add a new credential to the database and reports the result.
u3c_db_operation _result	<code>CC_UserCredential_modify_credential_and_report(u3c_credential_t *p_credential, RECEIVE_OPTIONS_TYPE_EX *p_rx_options)</code> Tries to modify a credential in the database and reports the result.
u3c_db_operation _result	<code>CC_UserCredential_delete_credential_and_report(uint16_t uuid, u3c_credential_type credential_type, uint16_t credential_slot, RECEIVE_OPTIONS_TYPE_EX *p_rx_options)</code> Tries to delete a credential from the database and reports the result.
u3c_db_operation _result	<code>CC_UserCredential_move_credential_and_report(u3c_credential_type credential_type, uint16_t credential_slot, uint16_t destination_uuid, RECEIVE_OPTIONS_TYPE_EX *p_rx_options)</code> Assigns a credential to a different user.
bool	<code>CC_UserCredential_send_association_report(u3c_credential_metadata_t const *const p_source_metadata, u3c_credential_metadata_t const *const p_destination_metadata, u3c_user_credential_association_report_status_t const status, RECEIVE_OPTIONS_TYPE_EX *const p_rx_options)</code> Sends a USER_CREDENTIAL_ASSOCIATION_REPORT frame.
void	<code>CC_UserCredential_delete_all_credentials_of_type(uint16_t uuid, u3c_credential_type filter_type)</code> Deletes all Credentials of a given type for a User.
bool	<code>CC_UserCredential_manufacturer_validate_credential(const u3c_credential_t *const credential)</code> Validates a Credential according to the manufacturer's security rules.
bool	<code>CC_UserCredential_manufacturer_validate_admin_pin_code(u3c_admin_code_metadata_t *const code)</code> Validates the following conditions and requirements: CC:0083.01.1A.11.011 - Code follows all manufacturer security rules.
void	<code>CC_UserCredential_set_default_name(uint8_t *pName, u3c_user_t *pUser)</code> Generates a default User Name value with either ASCII or UTF-16 encoding, according to CC:0083.01.05.12.052.

Macros

```
#define U3C_CREDENTIAL_TYPE_PIN_CODE_MIN_LENGTH_REQUIREMENT 4

#define U3C_CREDENTIAL_TYPE_PIN_CODE_MAX_LENGTH_REQUIREMENT 10

#define ASCII_AC_MAX 0x7F

#define AC_MIN_LENGTH 4

#define AC_MAX_LENGTH 10

#define U3C_BUFFER_SIZE_USER_NAME CC_USER_CREDENTIAL_MAX_LENGTH_USER_NAME

#define DEFAULT_NAME_MAX_SIZE (sizeof("User-65535") - 1) * 2
Longest possible default user name: "User-65535", times 2 for UTF-16.

#define CREDENTIAL_SET_FRAME_LENGTH_WITHOUT_CREDENTIAL_LENGTH
(offsetof(ZW_CREDENTIAL_SET_1BYTE_FRAME, credentialLength))

#define CREDENTIAL_SET_FRAME_LENGTH_WITHOUT_CREDENTIAL_DATA
(offsetof(ZW_CREDENTIAL_SET_1BYTE_FRAME, credentialData1))
```

Enumeration Documentation

u3c_db_operation_result_

u3c_db_operation_result_	
Enumerator	
U3C_DB_OPERATION_RESULT_SUCCESS	The operation completed succesfully.
U3C_DB_OPERATION_RESULT_ERROR	An error occurred.
U3C_DB_OPERATION_RESULT_ERROR_IO	An I/O error occurred.
U3C_DB_OPERATION_RESULT_ERROR_DUPLICATE	Duplicate Entry in table.
U3C_DB_OPERATION_RESULT_FAIL_DNE	The object does not exist.
U3C_DB_OPERATION_RESULT_FAIL_FULL	There is no space left for the object.
U3C_DB_OPERATION_RESULT_FAIL_OCCUPIED	The object already exists.
U3C_DB_OPERATION_RESULT_FAIL_REASSIGN	The credential is assigned to a different user.
U3C_DB_OPERATION_RESULT_FAIL_IDENTICAL	The new data is identical to the data already stored locally.

u3c_modifier_type_

u3c_modifier_type_	
Enumerator	
MODIFIER_TYPE_DNE	
MODIFIER_TYPE_UNKNOWN	
MODIFIER_TYPE_Z_WAVE	
MODIFIER_TYPE_LOCALLY	
MODIFIER_TYPE_MOBILE_APP_OR_OTHER_IOT_TECHNOLOGY	

u3c_credential_type_

u3c_credential_type_

Enumerator

CREDENTIAL_TYPE_NONE	
CREDENTIAL_TYPE_PIN_CODE	
CREDENTIAL_TYPE_PASSWORD	
CREDENTIAL_TYPE_RFID_CODE	
CREDENTIAL_TYPE_BLE	
CREDENTIAL_TYPE_NFC	
CREDENTIAL_TYPE_UWB	
CREDENTIAL_TYPE_EYE_BIOMETRIC	
CREDENTIAL_TYPE_FACE_BIOMETRIC	
CREDENTIAL_TYPE_FINGER_BIOMETRIC	
CREDENTIAL_TYPE_HAND_BIOMETRIC	
CREDENTIAL_TYPE_UNSPECIFIED_BIOMETRIC	
CREDENTIAL_TYPE_NUMBER_OF_TYPES	

_u3c_credential_report_type_t_

_u3c_credential_report_type_t_

Enumerator

CREDENTIAL_REP_TYPE_ADDED	
CREDENTIAL_REP_TYPE_MODIFIED	
CREDENTIAL_REP_TYPE_DELETED	
CREDENTIAL_REP_TYPE_UNCHANGED	
CREDENTIAL_REP_TYPE_RESPONSE_TO_GET	
CREDENTIAL_REP_TYPE_ADD_AGAINST_OCCUPIED	
CREDENTIAL_REP_TYPE_MODIF_AGAINST_EMPTY	
CREDENTIAL_REP_TYPE_DUPLICATE	
CREDENTIAL_REP_TYPE_MANUFACTURER_SECURITY_RULES	
CREDENTIAL_REP_TYPE_WRONG_UUID	
CREDENTIAL_REP_TYPE_DUPLICATE_ADMIN_PIN_CODE	

u3c_credential_learn_status_t_

u3c_credential_learn_status_t_

Enumerator

CL_STATUS_STARTED	
CL_STATUS_SUCCESS	
CL_STATUS_ALREADY_IN_PROGRESS	
CL_STATUS_ENDED_NOT_DUE_TO_TIMEOUT	
CL_STATUS_TIMEOUT	
CL_STATUS_STEP_RETRY	

CL_STATUS_INVALID_ADD_OPERATION_TYPE	
CL_STATUS_INVALID_MODIFY_OPERATION_TYPE	

u3c_user_credential_association_report_status_t

u3c_user_credential_association_report_status_t	
Enumerator	
U3C_UCAR_STATUS_SUCCESS	
U3C_UCAR_STATUS_CREDENTIAL_TYPE_INVALID	
U3C_UCAR_STATUS_CREDENTIAL_SLOT_INVALID	
U3C_UCAR_STATUS_CREDENTIAL_SLOT_EMPTY	
U3C_UCAR_STATUS_DESTINATION_USER_UNIQUE_IDENTIFIER_INVALID	
U3C_UCAR_STATUS_DESTINATION_USER_UNIQUE_IDENTIFIER_NONEXISTENT	

u3c_user_report_type_t

u3c_user_report_type_t	
Enumerator	
USER_REP_TYPE_ADDED	
USER_REP_TYPE_MODIFIED	
USER_REP_TYPE_DELETED	
USER_REP_TYPE_UNCHANGED	
USER_REP_TYPE_RESPONSE_TO_GET	
USER_REP_TYPE_ADD_AGAINST_OCCUPIED	
USER_REP_TYPE_MODIF_AGAINST_EMPTY	
USER_REP_TYPE_Z_EXP_MIN_INVALID	

u3c_user_type_

u3c_user_type_	
Enumerator	
USER_TYPE_GENERAL	
USER_TYPE_PROGRAMMING	
USER_TYPE_NON_ACCESS	
USER_TYPE_DURESS	
USER_TYPE_DISPOSABLE	
USER_TYPE_EXPIRING	
USER_TYPE_REMOTE_ONLY	
USER_TYPE_END	

u3c_credential_rule_

u3c_credential_rule_	

Enumerator

CREDENTIAL_RULE_SINGLE	
CREDENTIAL_RULE_DUAL	
CREDENTIAL_RULE_TRIPLE	

u3c_user_name_encoding_

u3c_user_name_encoding_

Enumerator

USER_NAME_ENCODING_STANDARD_ASCII	
USER_NAME_ENCODING_STANDARD_AND_OEM_EXTENDED_ASCII	
USER_NAME_ENCODING_UNICODE_UTF_16	

_u3c_admin_code_operation_result_

_u3c_admin_code_operation_result_

This operation result can optionally be injected into an Admin Code Report to pass information back to the controlling node, in lieu of another report type.

Reports MUST be sent for Get operations, but MAY be sent for Set operations.

See specification for more requirement details.

Enumerator

ADMIN_CODE_OPERATION_RESULT_MODIFIED	Admin Code was modified (Set, External).
ADMIN_CODE_OPERATION_RESULT_UNMODIFIED	Admin code was not modified as current code was identical.
ADMIN_CODE_OPERATION_RESULT_GET_RESP	Default response, or response to successful Get.
ADMIN_CODE_OPERATION_RESULT_FAIL_DUPLICATE_CRED	Admin Code was not modified (Set) due to duplicate existing PIN code credential.
ADMIN_CODE_OPERATION_RESULT_FAIL_MANUF_RULE	Admin Code was not modified (Set) due to a manufacturer security rule.
ADMIN_CODE_OPERATION_RESULT_ERROR_AC_NOT_SUPPORTED	Failed due to Admin Code not being supported (Get/Set).
ADMIN_CODE_OPERATION_RESULT_ERROR_ACD_NOT_SUPPORTED	Failed due to Admin Code Deactivation not being supported (Set).
ADMIN_CODE_OPERATION_RESULT_ERROR_NODE	Failed due to unspecified node error.
ADMIN_CODE_OPERATION_RESULT_INTERNAL_DUPLICATE_AC	Internal validation result; Code is the same as the current admin code.
ADMIN_CODE_OPERATION_RESULT_INTERNAL_NONE	Internal validation result; No error.
ADMIN_CODE_OPERATION_RESULT_INTERNAL_INVALID_LENGTH	Internal validation result; Code has an invalid length.
ADMIN_CODE_OPERATION_RESULT_INTERNAL_INVALID_CHAR	Internal validation result; Code has one or more invalid characters.

u3c_operation_type_t

u3c_operation_type_t

Enumerator

U3C_OPERATION_TYPE_ADD	
U3C_OPERATION_TYPE_MODIFY	
U3C_OPERATION_TYPE_DELETE	

u3c_event_

u3c_event_

Enumerator

CC_USER_CREDENTIAL_EVENT_VALIDATE	Application is requesting a Credential to be validated.
CC_USER_CREDENTIAL_EVENT_VALIDATE_VALID	Credential found and User has sufficient rights.
CC_USER_CREDENTIAL_EVENT_VALIDATE_INVALID	Credential not found or User has insufficient rights.
CC_USER_CREDENTIAL_EVENT_LEARN_START_REQUEST	Application is requesting to start Credential Learn.
CC_USER_CREDENTIAL_EVENT_LEARN_START	Credential Learn process started.
CC_USER_CREDENTIAL_EVENT_LEARN_STEP_START	Application progressed to new Credential Learn step.
CC_USER_CREDENTIAL_EVENT_LEARN_READ_DONE	Credential data is ready to be entered into the database.
CC_USER_CREDENTIAL_EVENT_LEARN_SUCCESS	Credential has been accepted into the database.
CC_USER_CREDENTIAL_EVENT_LEARN_FAILED	
CC_USER_CREDENTIAL_EVENT_LEARN_STEP_RETRY	Local Credential read failed, trying to read again.
CC_USER_CREDENTIAL_EVENT_LEARN_TIMEOUT	Credential Learn step timed out.
CC_USER_CREDENTIAL_EVENT_LEARN_CANCEL_REMOTE	Remote node cancelled Credential Learn process.
CC_USER_CREDENTIAL_EVENT_LEARN_CANCEL_LOCAL	Application cancelled Credential Learn process.

Typedef Documentation

u3c_db_operation_result

typedef enum u3c_db_operation_result_ u3c_db_operation_result

u3c_modifier_type

typedef enum u3c_modifier_type_ u3c_modifier_type

u3c_credential_type

typedef enum u3c_credential_type_ u3c_credential_type

u3c_credential_report_type_t

```
typedef enum _u3c_credential_report_type_t_ u3c_credential_report_type_t
```

u3c_credential_learn_status_t

```
typedef enum u3c_credential_learn_status_t_ u3c_credential_learn_status_t
```

u3c_user_report_type_t

```
typedef enum u3c_user_report_type_t_ u3c_user_report_type_t
```

u3c_user_type

```
typedef enum u3c_user_type_ u3c_user_type
```

u3c_credential_rule

```
typedef enum u3c_credential_rule_ u3c_credential_rule
```

u3c_user_name_encoding

```
typedef enum u3c_user_name_encoding_ u3c_user_name_encoding
```

u3c_admin_code_operation_result

```
typedef enum _u3c_admin_code_operation_result_ u3c_admin_code_operation_result
```

This operation result can optionally be injected into an Admin Code Report to pass information back to the controlling node, in lieu of another report type.

Reports MUST be sent for Get operations, but MAY be sent for Set operations.

See specification for more requirement details.

u3c_user_t

```
typedef struct u3c_user_t_ u3c_user_t
```

u3c_credential_metadata_t

```
typedef struct u3c_credential_metadata_t_ u3c_credential_metadata_t
```

u3c_credential_t

```
typedef struct u3c_credential_t_ u3c_credential_t
```

u3c_credential_identifier_t

```
typedef struct u3c_credential_identifier_t_ u3c_credential_identifier_t
```

u3c_event_data_validate_t

```
typedef struct u3c_event_data_validate_t_ u3c_event_data_validate_t
```

Payload for CC_USER_CREDENTIAL_EVENT_VALIDATE, received from the application.

u3c_event_data_learn_read_done_t

```
typedef struct u3c_event_data_learn_read_done_t_ u3c_event_data_learn_read_done_t
```

Payload for CC_USER_CREDENTIAL_EVENT_LEARN_READ_DONE, passed to the application.

u3c_event_data_learn_start_t

```
typedef struct u3c_event_data_learn_start_t_ u3c_event_data_learn_start_t
```

Credential Learn information passed to application layer.

u3c_admin_code_metadata_t

```
typedef struct u3c_admin_code_metadata_t_ u3c_admin_code_metadata_t
```

u3c_event_type

```
typedef enum u3c_event_ u3c_event_type
```

Function Documentation

CC_UserCredential_factory_reset

```
void CC_UserCredential_factory_reset (void )
```

Resets the User Credential Command Class database to a blank state, removing all existing users, credentials and the admin code.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

CC_UserCredential_init_database

```
void CC_UserCredential_init_database (void )
```

Initializes the User Credential Command Class database to a consistent state.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is called automatically by the Z-Wave Application Framework when the application starts.

CC_UserCredential_get_user

```
u3c_db_operation_result CC_UserCredential_get_user (uint16_t unique_identifier, u3c\_user\_t * user, uint8_t * name)
```

Copies the data for a given User from the database to an arbitrary location in memory.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	unique_identifier	Unique Identifier of the requested User
u3c_user_t *	[out]	user	The memory location where the User data will be copied, or NULL if not requested
uint8_t *	[out]	name	The memory location where the User name will be copied, or NULL if not requested. If not NULL, user cannot be NULL either.

Returns

- the result of the operation

CC_UserCredential_get_next_user

```
uint16_t CC_UserCredential_get_next_user (uint16_t unique_identifier)
```

Finds the first or next available User Unique Identifier.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	unique_identifier	Unique Identifier of the current User, or 0 if the first available User Unique Identifier is requested

Returns

- The Unique Identifier of the next User if available, otherwise 0

CC_UserCredential_add_user

u3c_db_operation_result CC_UserCredential_add_user (u3c_user_t * user, uint8_t * name)

Parameters

Type	Direction	Argument Name	Description
u3c_user_t *	N/A	user	
uint8_t *	N/A	name	

CC_UserCredential_modify_user

u3c_db_operation_result CC_UserCredential_modify_user (u3c_user_t * user, uint8_t * name)

Modifies a user in the database.

Parameters

Type	Direction	Argument Name	Description
u3c_user_t *	[in]	user	Pointer to the user's new metadata
uint8_t *	[in]	name	Pointer to the user's new name (set to NULL to keep old name)

CC_UserCredential_delete_user

u3c_db_operation_result CC_UserCredential_delete_user (uint16_t user_unique_identifier)

Parameters

Type	Direction	Argument Name	Description
uint16_t	N/A	user_unique_identifier	

CC_UserCredential_get_credential

u3c_db_operation_result CC_UserCredential_get_credential (uint16_t user_unique_identifier, u3c_credential_type credential_type, uint16_t credential_slot, u3c_credential_metadata_t * credential_metadata, uint8_t * credential_data)

Copies the data for a given Credential from the database to an arbitrary location in memory.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	user_unique_identifier	Unique Identifier of the Credential's associated User (0 matches any UUID)
u3c_credential_type	[in]	credential_type	Type of the requested Credential
uint16_t	[in]	credential_slot	Slot of the requested Credential
u3c_credential_metadata_t *	[out]	credential_metadata	The memory location where the Credential metadata will be copied, or NULL if not requested
uint8_t *	[out]	credential_data	The memory location where the Credential data will be copied, or NULL if not requested

Returns

- the result of the operation

CC_UserCredential_get_next_credential

```
bool CC_UserCredential_get_next_credential (uint16_t user_unique_identifier, u3c_credential_type credential_type, uint16_t credential_slot, u3c_credential_type * next_credential_type, uint16_t * next_credential_slot)
```

Finds the first or next available Credential entry for a given User or Credential Type (set either one of these parameters to 0 to disable filtering by that property).

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	user_unique_identifier	Unique Identifier of the Credential's associated User (0 matches any UUID)
u3c_credential_type	[in]	credential_type	Type of the current Credential (0 matches any Credential Type)
uint16_t	[in]	credential_slot	Slot of the current Credential (0 to get the first available Credential)
u3c_credential_type *	[out]	next_credential_type	Type of the next available Credential, if available
uint16_t *	[out]	next_credential_slot	Slot of the next available Credential, if available

Set credential_slot to 0 to get the first available Credential matching the filter criteria.

Returns

- true if a subsequent Credential matching the criteria is found

CC_UserCredential_add_credential

```
u3c_db_operation_result CC_UserCredential_add_credential (u3c_credential_t * credential)
```

Adds a new credential to the database.

Parameters

Type	Direction	Argument Name	Description
u3c_credential_t *	[in]	credential	Pointer to the credential to be added

Returns

- The result of the operation

CC_UserCredential_modify_credential

u3c_db_operation_result CC_UserCredential_modify_credential (u3c_credential_t * credential)

Parameters

Type	Direction	Argument Name	Description
u3c_credential_t *	N/A	credential	

CC_UserCredential_delete_credential

u3c_db_operation_result CC_UserCredential_delete_credential (u3c_credential_type credential_type, uint16_t credential_slot, credential_slot)

Deletes a Credential.

Parameters

Type	Direction	Argument Name	Description
u3c_credential_type	[in]	credential_type	Unique Identifier of the Credential's associated User (0 matches any UUID)
uint16_t	[in]	credential_slot	Type of the current Credential
	[in]	credential_slot	Slot of the current Credential

Returns

- the result of the operation

CC_UserCredential_move_credential

u3c_db_operation_result CC_UserCredential_move_credential (u3c_credential_type credential_type, uint16_t source_credential_slot, uint16_t destination_user_uid, uint16_t destination_credential_slot)

Moves a credential from one slot to another and/or assigns a credential from one User Unique Identifier to another one.

Parameters

Type	Direction	Argument Name	Description
u3c_credential_type	[in]	credential_type	Type of the credential
uint16_t	[in]	source_credential_slot	Slot of the credential to be moved
uint16_t	[in]	destination_user_uid	The new desired UUID to assign the credential to
uint16_t	[in]	destination_credential_slot	The new desired slot number for the credential

CC_UserCredential_get_admin_code_info

```
u3c_db_operation_result CC_UserCredential_get_admin_code_info (u3c_admin_code_metadata_t * code)
```

Retrieves the admin code information from the node.

Parameters

Type	Direction	Argument Name	Description
u3c_admin_code_metadata_t *	[out]	code	Admin Code information. If Admin Code is not supported, the struct will be fully zeroed out upon return.

Can be used to ensure node supports the functionality.

Returns

- U3C_DB_OPERATION_SUCCESS if successful
- U3C_DB_OPERATION_ERROR otherwise

CC_UserCredential_set_admin_code

```
u3c_db_operation_result CC_UserCredential_set_admin_code (u3c_admin_code_metadata_t * code)
```

Sets the admin code on the node, if supported.

Parameters

Type	Direction	Argument Name	Description
u3c_admin_code_metadata_t *	[in]	code	Admin Code information to apply to node

Returns

- U3C_DB_OPERATION_ERROR_DUPLICATE if Admin Code is a duplicate of an existing PIN Code or Admin Code
- U3C_OPERATION_SUCCESS if successful
- U3C_OPERATION_ERROR otherwise

CC_UserCredential_add_user_and_report

```
u3c_db_operation_result CC_UserCredential_add_user_and_report (u3c_user_t * p_user, uint8_t * p_name, RECEIVE_OPTIONS_TYPE_EX * p_rx_options)
```

Tries to add a new user to the database and reports the result.

Parameters

Type	Direction	Argument Name	Description
u3c_user_t *	[in]	p_user	Pointer to the new user's metadata
uint8_t *	[in]	p_name	Pointer to the new user's name
RECEIVE_OPTIONS_TYPE_EX *	[in]	p_rx_options	Pointer to the details of the received frame

Returns

- The result of the operation

CC_UserCredential_modify_user_and_report

```
u3c_db_operation_result CC_UserCredential_modify_user_and_report (u3c_user_t * p_user, uint8_t * p_name,
RECEIVE_OPTIONS_TYPE_EX * p_rx_options)
```

Tries to modify a user in the database and reports the result.

Parameters

Type	Direction	Argument Name	Description
u3c_user_t *	[in]	p_user	Pointer to the user's new metadata
uint8_t *	[in]	p_name	Pointer to the new user's new name
RECEIVE_OPTIONS_TYPE_EX *	[in]	p_rx_options	Pointer to the details of the received frame

Returns

- The result of the operation

CC_UserCredential_delete_user_and_report

```
u3c_db_operation_result CC_UserCredential_delete_user_and_report (uint16_t uuid, RECEIVE_OPTIONS_TYPE_EX *
p_rx_options)
```

Tries to delete a user from the database and reports the result.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	uuid	The target user's unique identifier or 0 to delete all users.
RECEIVE_OPTIONS_TYPE_EX *	[in]	p_rx_options	Pointer to the details of the received frame

Returns

- The result of the operation

CC_UserCredential_add_credential_and_report

```
u3c_db_operation_result CC_UserCredential_add_credential_and_report (u3c_credential_t * p_credential,
RECEIVE_OPTIONS_TYPE_EX * p_rx_options)
```

Tries to add a new credential to the database and reports the result.

Parameters

Type	Direction	Argument Name	Description
u3c_credential_t *	[in]	p_credential	Pointer to the new credential
RECEIVE_OPTIONS_TYPE_EX *	[in]	p_rx_options	Pointer to the details of the received frame

Returns

- The result of the operation

CC_UserCredential_modify_credential_and_report

```
u3c_db_operation_result CC_UserCredential_modify_credential_and_report (u3c_credential_t * p_credential,
RECEIVE_OPTIONS_TYPE_EX * p_rx_options)
```

Tries to modify a credential in the database and reports the result.

Parameters

Type	Direction	Argument Name	Description
u3c_credential_t *	[in]	p_credential	Pointer to the credential with the new desired data
RECEIVE_OPTIONS_TYPE_EX *	[in]	p_rx_options	Pointer to the details of the received frame

Returns

- The result of the operation

CC_UserCredential_delete_credential_and_report

```
u3c_db_operation_result CC_UserCredential_delete_credential_and_report (uint16_t uuid, u3c_credential_type
credential_type, uint16_t credential_slot, RECEIVE_OPTIONS_TYPE_EX * p_rx_options)
```

Tries to delete a credential from the database and reports the result.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	uuid	The UUID that the credentials are assigned to (only used for bulk operations), 0 to disable filtering by user
u3c_credential_type	[in]	credential_type	Type of the target credential
uint16_t	[in]	credential_slot	Slot of the target credential, or 0 for all credentials
RECEIVE_OPTIONS_TYPE_EX *	[in]	p_rx_options	Pointer to the details of the received frame

Returns

- The result of the operation

CC_UserCredential_move_credential_and_report

```
u3c_db_operation_result CC_UserCredential_move_credential_and_report (u3c_credential_type credential_type, uint16_t
credential_slot, uint16_t destination_uuid, RECEIVE_OPTIONS_TYPE_EX * p_rx_options)
```

Assigns a credential to a different user.

Parameters

Type	Direction	Argument Name	Description
u3c_credential_type	[in]	credential_type	Type of the credential
uint16_t	[in]	credential_slot	Slot of the credential
uint16_t	[in]	destination_uuid	New desired UUID to assign the credential to

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE_EX *	[in]	p_rx_options	Pointer to the details of the received frame

Returns

- The result of the operation

CC_UserCredential_send_association_report

```
bool CC_UserCredential_send_association_report (u3c_credential_metadata_t const *const p_source_metadata,
u3c_credential_metadata_t const *const p_destination_metadata, u3c_user_credential_association_report_status_t const
status, RECEIVE\_OPTIONS\_TYPE\_EX *const p_rx_options, status, p_rx_options)
```

Sends a USER_CREDENTIAL_ASSOCIATION_REPORT frame.

Parameters

Type	Direction	Argument Name	Description
u3c_credential_metadata_t const *const	[out]	p_source_metadata	Address of frame memory to fill.
u3c_credential_metadata_t const *const	[inout]	p_destination_metadata	Frame buffer length. Will be overwritten with the filled frame length.
u3c_user_credential_association_report_status_t const	[in]	status	Source credential metadata.
RECEIVE_OPTIONS_TYPE_EX *const	[in]	p_rx_options	Destination credential metadata.
	[in]	status	Status of association set action.
	[in]	p_rx_options	Pointer to the details of the received frame.

Returns

- true if the frame was enqueued successfully.

CC_UserCredential_delete_all_credentials_of_type

```
void CC_UserCredential_delete_all_credentials_of_type (uint16_t uuid, u3c\_credential\_type filter_type)
```

Deletes all Credentials of a given type for a User.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	uuid	User Unique Identifier
u3c_credential_type	[in]	filter_type	The type of Credentials to delete, set this to CREDENTIAL_TYPE_NONE to delete every Credential

CC_UserCredential_manufacturer_validate_credential

```
bool CC_UserCredential_manufacturer_validate_credential (const u3c\_credential\_t *const credential)
```

Validates a Credential according to the manufacturer's security rules.

Parameters

Type	Direction	Argument Name	Description
const u3c_credential_t *const	N/A	credential	

Returns

- True if the Credential conforms to the security rules

CC_UserCredential_manufacturer_validate_admin_pin_code

```
bool CC_UserCredential_manufacturer_validate_admin_pin_code (u3c\_admin\_code\_metadata\_t *const code)
```

Validates the following conditions and requirements: CC:0083.01.1A.11.011 - Code follows all manufacturer security rules.

Parameters

Type	Direction	Argument Name	Description
u3c_admin_code_metadata_t *const	N/A	code	Pointer to structure containing Admin Code information.

Should not be called if AC is not supported.

Returns

- True if all checks pass. If checks do not pass, returns false, and result code populated in code struct.

Note

- result code of a successful check should be ignored, but in practice ensure it is set to ADMIN_CODE_OPERATION_RESULT_NONE in this case.

CC_UserCredential_set_default_name

```
void CC_UserCredential_set_default_name (uint8_t * pName, u3c\_user\_t * pUser)
```

Generates a default User Name value with either ASCII or UTF-16 encoding, according to CC:0083.01.05.12.052.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	pName	Pointer to the User Name
u3c_user_t *	[in]	pUser	Pointer to the User object

u3c_user_t_

Public Attributes

uint16_t	unique_identifier
uint16_t	modifier_node_id
uint16_t	expiring_timeout_minutes
uint8_t	name_length
u3c_user_type	type
u3c_modifier_type	modifier_type
u3c_credential_rule	credential_rule
u3c_user_name_encoding	name_encoding
bool	active

Public Attribute Documentation

unique_identifier

```
uint16_t u3c_user_t::unique_identifier
```

modifier_node_id

```
uint16_t u3c_user_t::modifier_node_id
```

expiring_timeout_minutes

```
uint16_t u3c_user_t::expiring_timeout_minutes
```

name_length

```
uint8_t u3c_user_t::name_length
```

type

u3c_user_type u3c_user_t::type

modifier_type

u3c_modifier_type u3c_user_t::modifier_type

credential_rule

u3c_credential_rule u3c_user_t::credential_rule

name_encoding

u3c_user_name_encoding u3c_user_t::name_encoding

active

bool u3c_user_t::active

u3c_credential_metadata_t_

Public Attributes

uint16_t	uuid
uint16_t	slot
uint16_t	modifier_node_id
uint8_t	length
u3c_modifier_type	modifier_type
u3c_credential_type	type

Public Attribute Documentation

uuid

uint16_t u3c_credential_metadata_t::uuid

slot

uint16_t u3c_credential_metadata_t::slot

modifier_node_id

uint16_t u3c_credential_metadata_t::modifier_node_id

length

uint8_t u3c_credential_metadata_t::length

modifier_type

u3c_modifier_type u3c_credential_metadata_t::modifier_type

type

u3c_credential_type u3c_credential_metadata_t::type

u3c_credential_t_

Public Attributes

u3c_credential_m	metadata
etadata_t	
uint8_t *	data

Public Attribute Documentation

metadata

u3c_credential_metadata_t u3c_credential_t::metadata

data

uint8_t* u3c_credential_t::data

u3c_credential_identifier_t_

Public Attributes

uint16_t	uuid
u3c_credential_type	type
uint16_t	slot

Public Attribute Documentation

uuid

```
uint16_t u3c_credential_identifier_t::uuid
```

type

```
u3c_credential_type u3c_credential_identifier_t::type
```

slot

```
uint16_t u3c_credential_identifier_t::slot
```

u3c_event_data_validate_t_

Payload for CC_USER_CREDENTIAL_EVENT_VALIDATE, received from the application.

Public Attributes

u3c_credential_t *	credential
bool	is_unlocked

Public Attribute Documentation

credential

u3c_credential_t* u3c_event_data_validate_t::credential

is_unlocked

bool u3c_event_data_validate_t::is_unlocked

u3c_event_data_learn_read_done_t_

Payload for CC_USER_CREDENTIAL_EVENT_LEARN_READ_DONE, passed to the application.

Public Attributes

```
uint8_t * data
uint8_t data_length
```

Public Attribute Documentation

data

```
uint8_t* u3c_event_data_learn_read_done_t::data
```

data_length

```
uint8_t u3c_event_data_learn_read_done_t::data_length
```

u3c_event_data_learn_start_t_

Credential Learn information passed to application layer.

Public Attributes

u3c_credential_id	target
entifier_t	
u3c_operation_ty	operation_type
pe_t	
uint16_t	source_id
uint8_t	timeout_seconds

Public Attribute Documentation

target

u3c_credential_identifier_t u3c_event_data_learn_start_t::target

operation_type

u3c_operation_type_t u3c_event_data_learn_start_t::operation_type

source_id

uint16_t u3c_event_data_learn_start_t::source_id

timeout_seconds

uint8_t u3c_event_data_learn_start_t::timeout_seconds

u3c_admin_code_metadata_t_

Public Attributes

u3c_admin_code_	result
operation_result	
uint8_t	code_length
uint8_t	code_data

Public Attribute Documentation

result

```
u3c_admin_code_operation_result u3c_admin_code_metadata_t::result
```

code_length

```
uint8_t u3c_admin_code_metadata_t::code_length
```

code_data

```
uint8_t u3c_admin_code_metadata_t::code_data[U3C_CREDENTIAL_TYPE_PIN_CODE_MAX_LENGTH_REQUIREMENT]
```


WakeUp

WakeUp

Modules

[SWakeupCcData](#)

Typedefs

typedef struct [SWakeupCcData](#)

Functions

void	CC_WakeUp_notification_tx (void(*pCallback)(uint8_t txStatus, TX_STATUS_TYPE *pExtendedTxStatus)) Transmits a Wake Up Notification command and handles callback.
uint32_t	cc_wake_up_config_get_default_sleep_time_sec (void) Used by the WakeUp Command Class configuration Returns with the default sleep time in second between Wake Up Notifications.
uint32_t	cc_wake_up_config_get_minimum_sleep_time_sec (void) Used by the WakeUp Command Class configuration Returns with the minimum sleep time in second between Wake Up Notifications.
uint32_t	cc_wake_up_config_get_maximum_sleep_time_sec (void) Used by the WakeUp Command Class configuration Returns with the maximum sleep time in second between Wake Up Notifications.
uint32_t	cc_wake_up_config_get_sleep_step_time_sec (void) Used by the WakeUp Command Class configuration Returns with the sleep step time in second.

Macros

#define ZAF_FILE_SIZE_WAKEUPCCDATA (sizeof(SWakeupCcData))

Typedef Documentation

SWakeupCcData

```
typedef struct SWakeupCcData SWakeupCcData
```

Function Documentation

CC_WakeUp_notification_tx

```
void CC_WakeUp_notification_tx (void(*) (uint8_t txStatus, TX_STATUS_TYPE *pExtendedTxStatus) pCallback)
```

Transmits a Wake Up Notification command and handles callback.

Parameters

Type	Direction	Argument Name	Description
void(*) (uint8_t txStatus, TX_STATUS_TYPE *pExtendedTxStatus)	[in]	pCallback	Pointer to function which will be called on Wake Up Notification ACK/NACK.

cc_wake_up_config_get_default_sleep_time_sec

```
uint32_t cc_wake_up_config_get_default_sleep_time_sec (void )
```

Used by the WakeUp Command Class configuration Returns with the default sleep time in second between Wake Up Notifications.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Default sleep time in second between Wake Up Notifications

cc_wake_up_config_get_minimum_sleep_time_sec

```
uint32_t cc_wake_up_config_get_minimum_sleep_time_sec (void )
```

Used by the WakeUp Command Class configuration Returns with the minimum sleep time in second between Wake Up Notifications.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Minimum sleep time in second

cc_wake_up_config_get_maximum_sleep_time_sec

```
uint32_t cc_wake_up_config_get_maximum_sleep_time_sec (void )
```

Used by the WakeUp Command Class configuration Returns with the maximum sleep time in second between Wake Up Notifications.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Maximum sleep time in second

cc_wake_up_config_get_sleep_step_time_sec

```
uint32_t cc_wake_up_config_get_sleep_step_time_sec (void )
```

Used by the WakeUp Command Class configuration Returns with the sleep step time in second.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- sleep step time in second

SWakeupCcData

Public Attributes

uint8_t [MasterNodeId](#)

uint32_t [SleepPeriod](#)

Public Attribute Documentation

MasterNodeId

uint8_t SWakeupCcData::MasterNodeId

SleepPeriod

uint32_t SWakeupCcData::SleepPeriod

ZWavePlusInfo

ZWavePlusInfo

Modules

[zw_plus_info_config_icons_t](#)

Functions

- [zw_plus_info_config_icons_t *](#)

[cc_zwave_plus_info_config_get_root_icons\(void\)](#)
Returns the Installer and User Icon Type configuration of the root device.
- [uint8_t](#)

[cc_zwave_plus_info_config_get_endpoint_count\(void\)](#)
Returns the number of endpoints defined in the application command class configuration.
- [zw_plus_info_config_icons_t *](#)

[cc_zwave_plus_info_config_get_endpoint_entry\(uint8_t endpoint\)](#)
Returns the Installer and User Icon Type configuration of the given 'endpoint'.

Function Documentation

cc_zwave_plus_info_config_get_root_icons

```
zw_plus_info_config_icons_t * cc_zwave_plus_info_config_get_root_icons (void )
```

Returns the Installer and User Icon Type configuration of the root device.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Pointer to struct contains the icon types.

cc_zwave_plus_info_config_get_endpoint_count

```
uint8_t cc_zwave_plus_info_config_get_endpoint_count (void )
```

Returns the number of endpoints defined in the application command class configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

In case of no zw_cc_zwaveplusinfo is given in the config, returns with 0.

Returns

- the number of endpoints.

cc_zwave_plus_info_config_get_endpoint_entry

```
zw_plus_info_config_icons_t * cc_zwave_plus_info_config_get_endpoint_entry (uint8_t endpoint)
```

Returns the Installer and User Icon Type configuration of the given 'endpoint'.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	endpoint	the index of the requested endpoint. First endpoint index starting by 1.

Returns

- Pointer to struct contains the icon types for the given endpoint.

zw_plus_info_config_icons_t

Public Attributes

uint16_t	installer_icon_type
uint16_t	user_icon_type
uint8_t	endpoint

Public Attribute Documentation

installer_icon_type

uint16_t zw_plus_info_config_icons_t::installer_icon_type

user_icon_type

uint16_t zw_plus_info_config_icons_t::user_icon_type

endpoint

uint8_t zw_plus_info_config_icons_t::endpoint

ZAF

ZAF

Modules

[ZAF Interface](#)

[ZAF Actuator Module](#)

[Application Timer](#)

[Command Classes List Generator](#)

[Event Manager](#)

[MISC](#)

[NVM](#)

[True Status Engine](#)

[ApplicationEvent](#)

[Association Data Store](#)

[CC_Invoker](#)

[CMD_Publisher](#)

[Command Class Utils](#)

[FILE_ID](#)

[Network](#)

[Retention Register](#)

[ZAF_Transport](#)

[TYPES](#)

[PRODUCT_ID_ENUM](#)

[ZAF_ProtocolConfig](#)

ZAF Interface

ZAF Interface

Enumerations

```
enum EInclusionMode\_t {
    EINCLUSIONMODE_NOT_SET = 0
    EINCLUSIONMODE_ZWAVE_CLS
    EINCLUSIONMODE_ZWAVE_LR
}
```

Used to indicate node inclusion type that related to it working over Long Range or Z-Wave mode.

Typedefs

```
typedef void(*) zaf\_wake\_up\_callback\_t(void)
```

Functions

```
void ZAF\_Init(TaskHandle_t AppTaskHandle, SApplicationHandles *pAppHandles)
Initializes the Z-Wave Application Framework.
```

```
void ZAF\_Reset(void)
Reset ZAF and command classes that have registered a reset function with a REGISTER_CC macro.
```

```
TaskHandle_t ZAF\_getAppTaskHandle(void)
Returns the task handle set by ZAF\_Init().
```

```
void zaf\_stay\_awake(void)
Makes the node stay awake for either 2 or 10 seconds.
```

```
void zaf\_learn\_mode\_finished(void)
Function called when the EVENT\_SYSTEM\_LEARNMODE\_FINISHED is received.
```

```
void ZAF\_setAppHandle(SApplicationHandles *pAppHandle)
Pass an application handle to the ZAF framework with pointers to application resources.
```

```
void ZAF\_SetCPHandle(CP_Handle_t handle)
Set the CP context.
```

```
SApplicationHandles * ZAF\_getAppHandle(void)
Get m_pAppHandles object.
```

```
const SProtocolInfo * ZAF\_getProtocolInfo(void)
Get Protocol Info object.
```

```
SQueueNotifying * ZAF\_getZwTxQueue(void)
Get Notifying queue object.
```

```
SQueueNotifying * ZAF\_getZwCommandQueue(void)
Get Notifying queue object used for ZW Command Queue.
```

```
const zpal_radio_network_stats_t * ZAF\_getNetworkStatistics(void)
Get Network Statistics object.
```

uint8_t	ZAF_GetSecurityKeys(void) Returns the granted security keys.
CP_Handle_t	ZAF_getCPHandle(void) Get Command Publisher handle Any other module can request the address of the CP context to use the API.
bool	isFLiRS(const SAppNodeInfo_t *pAppNodeInfo) Check if device is FLiRS based on device option mask.
EInclusionState_t	ZAF_GetInclusionState(void) Returns the current inclusion state of the node.
node_id_t	ZAF_GetNodeID(void) Returns the current ID of the node.
uint32_t	ZAF_GetHomeID(void) Returns the current home ID.
node_id_t	ZAF_GetSucNodeid(void) Returns the current Suc node ID.
EInclusionMode_t	ZAF_GetInclusionMode(void) Returns the current inclusion mode of the node, which indicates whether the node is acting as a Z-Wave or Z-Wave LR node.
bool	isRfRegionValid(zpal_radio_region_t region) Checks if region is supported by Z-Wave stack.
bool	ZAF_isLongRangeRegion(zpal_radio_region_t eRegion) Check if eRegion is a Long Range region.
void	zaf_set_stay_awake_callback(zaf_wake_up_callback_t callback) Registers a wake up function to keep the node awake.
zaf_wake_up_callback_t	zaf_get_stay_awake_callback(void) Returns the wake up callback.

Enumeration Documentation

EInclusionMode_t

EInclusionMode_t

Used to indicate node inclusion type that related to it working over Long Range or Z-Wave mode.

This information will be used to distinguish between these two mode in order to indicate the needed operations when they differ between Z-Wave and Long Range.

Enumerator	
EINCLUSIONMODE_NOT_SET	
EINCLUSIONMODE_ZWAVE_CLS	!< Does not indicate inclusion state, but that it is undefined!
EINCLUSIONMODE_ZWAVE_LR	!< This node was included as a Z-Wave Classic device.

Typedef Documentation

zaf_wake_up_callback_t

```
typedef void(* zaf_wake_up_callback_t) (void) (void)
```

Function Documentation

ZAF_Init

```
void ZAF_Init (TaskHandle_t AppTaskHandle, SApplicationHandles * pAppHandles)
```

Initializes the Z-Wave Application Framework.

Parameters

Type	Direction	Argument Name	Description
TaskHandle_t	N/A	AppTaskHandle	Task handle
SApplicationHandles *	N/A	pAppHandles	pointer to Application Handle in application

This function must be invoked by end devices and portable controllers, but not Serial API.

It initializes the following stuff:

- Application timer
- Board
- Application file system
- Power lock for FLIRS
- True Status
- Command Publisher
- All command classes that have registered an init function.
- Generates the three lists of command classes used for Node Information Frame, etc.

ZAF_Reset

```
void ZAF_Reset (void )
```

Reset ZAF and command classes that have registered a reset function with a REGISTER_CC macro.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZAF_getAppTaskHandle

```
TaskHandle_t ZAF_getAppTaskHandle (void )
```

Returns the task handle set by [ZAF_Init\(\)](#).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Task handle.

zaf_stay_awake

```
void zaf_stay_awake (void )
```

Makes the node stay awake for either 2 or 10 seconds.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

For an LS (listening sleeping) node, the node will stay awake for 2 seconds. If the node supports CC Wake Up, it will stay awake for 10 seconds, if CC Wake Up is active.

For RS (reporting sleeping), the node will stay awake for 10 seconds, if CC Wake Up is active.

zaf_learn_mode_finished

```
void zaf_learn_mode_finished (void )
```

Function called when the [EVENT_SYSTEM_LEARNMODE_FINISHED](#) is received.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function is weakly defined and can be overwritten by one command class. Applications should not overwrite this function instead they should handle the event. CC WakeUp overwrites it.

ZAF_setAppHandle

```
void ZAF_setAppHandle (SApplicationHandles * pAppHandle)
```

Pass an application handle to the ZAF framework with pointers to application resources.

Parameters

Type	Direction	Argument Name	Description
SApplicationHandles *	N/A	pAppHandle	pointer to Application Handle in application

Can be later used by CC-s.

ZAF_SetCPHandle

```
void ZAF_SetCPHandle (CP\_Handle\_t handle)
```

Set the CP context.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	to set

The application should use this an initialization, so the ZAF_Common interface can redistribute the pContext address subsequently

ZAF_getAppHandle

```
SApplicationHandles * ZAF_getAppHandle (void )
```

Get m_pAppHandles object.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- m_pAppHandles

ZAF_getProtocolInfo

```
const SProtocolInfo * ZAF_getProtocolInfo (void )
```

Get Protocol Info object.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- pProtocolInfo member of m_pAppHandles

ZAF_getZwTxQueue

```
SQueueNotifying * ZAF_getZwTxQueue (void )
```

Get Notifying queue object.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

It is stored in m_pAppHandles

Returns

- pZwTxQueue member of m_pAppHandles

ZAF_getZwCommandQueue

SQueueNotifying * ZAF_getZwCommandQueue (void)

Get Notifying queue object used for ZW Command Queue.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- pZwCommandQueue member of m_pAppHandles

ZAF_getNetworkStatistics

const zpal_radio_network_stats_t * ZAF_getNetworkStatistics (void)

Get Network Statistics object.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- pNetworkStatistics member of m_pAppHandles

ZAF_GetSecurityKeys

uint8_t ZAF_GetSecurityKeys (void)

Returns the granted security keys.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Refer to [ZW_security_api.h](#) for the security key masks. **Returns**

- 8 bit mask where a set bit indicates that a security key is granted.

ZAF_getCPHandle

CP_Handle_t ZAF_getCPHandle (void)

Get Command Publisher handle Any other module can request the address of the CP context to use the API.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Pointer to Command Publisher context

isFLiRS

```
bool isFLiRS (const SAppNodeInfo_t * pAppNodeInfo)
```

Check if device is FLiRS based on device option mask.

Parameters

Type	Direction	Argument Name	Description
const SAppNodeInfo_t *	N/A	pAppNodeInfo	

Returns

- True or false, based on pAppNodeInfo

ZAF_GetInclusionState

```
ElInclusionState_t ZAF_GetInclusionState (void )
```

Returns the current inclusion state of the node.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- ElInclusionState_t

ZAF_GetNodeID

```
node_id_t ZAF_GetNodeID (void )
```

Returns the current ID of the node.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

If the node is not added to a network, the node ID is zero. Returns

- Node ID

ZAF_GetHomeID

uint32_t ZAF_GetHomeID (void)

Returns the current home ID.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Home ID

ZAF_GetSucNodeId

node_id_t ZAF_GetSucNodeId (void)

Returns the current Suc node ID.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Suc Node ID

ZAF_GetInclusionMode

EInclusionMode_t ZAF_GetInclusionMode (void)

Returns the current inclusion mode of the node, which indicates whether the node is acting as a Z-Wave or Z-Wave LR node.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- EInclusionMode_t

isRfRegionValid

bool isRfRegionValid (zpal_radio_region_t region)

Checks if region is supported by Z-Wave stack.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_region_t	N/A	region	Region to check

Returns

- True if region is valid, false if it isn't.

ZAF_isLongRangeRegion

```
bool ZAF_isLongRangeRegion (zpal_radio_region_t eRegion)
```

Check if eRegion is a Long Range region.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_region_t	N/A	eRegion	Region to check

Warning: the function does not check that the region is supported by the zpal. **Returns**

- True if the region support Long Range, False otherwise.

zaf_set_stay_awake_callback

```
void zaf_set_stay_awake_callback (zaf_wake_up_callback_t callback)
```

Registers a wake up function to keep the node awake.

Parameters

Type	Direction	Argument Name	Description
zaf_wake_up_callback_t	N/A	callback	Function to invoke to keep the node awake.

The function will be invoked on every reception of a frame to keep the node awake.

zaf_get_stay_awake_callback

```
zaf_wake_up_callback_t zaf_get_stay_awake_callback (void )
```

Returns the wake up callback.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Wake up callback

ZAF Actuator Module

ZAF Actuator Module

Modules

actuator

Enumerations

```
enum _eActuatorState {
    EACTUATOR_NOT_CHANGING
    EACTUATOR_CHANGING
    EACTUATOR_FAILED
}
```

Return type for Actuator functions that can trigger changes that require some time to be done.

Typedefs

```
typedef void(*) zaf_actuator_callback_t(struct actuator *pActuator)

typedef struct s_Actuator
actuator The structure of Actuator module.

typedef enum eActuatorState
_eActuatorState Return type for Actuator functions that can trigger changes that require some time to be done.
```

Functions

```
void ZAF_Actuator_Init(s_Actuator *pActuator, uint8_t minValue, uint8_t maxValue, uint16_t refreshRate, uint8_t
durationDefault, zaf_actuator_callback_t cc_callback)
Initializes Actuator module.

eActuatorState ZAF_Actuator_Set(s_Actuator *pActuator, uint8_t value, uint8_t duration)
Sets target value to value in actuator pointed by pActuator, during duration period.

eActuatorState ZAF_Actuator_StartChange(s_Actuator *pActuator, bool ignoreStartLevel, bool upDown, uint8_t startLevel,
uint8_t duration)
Starts the change on actuator pointed by pActuator .

eActuatorState ZAF_Actuator_StopChange(s_Actuator *pActuator)
Stops ongoing change on actuator pointed by pActuator .

uint8_t ZAF_Actuator_GetCurrentValue(s_Actuator *pActuator)
Returns current value.

uint8_t ZAF_Actuator_GetTargetValue(s_Actuator *pActuator)
Returns target value.

uint8_t ZAF_Actuator_GetDurationRemaining(s_Actuator *pActuator)
Returns estimated time until target value is reached.
```

- uint8_t

[ZAF_Actuator_GetLastOnValue](#)(s_Actuator *pActuator)
Gets last ON value.
- uint32_t

[getDurationInMs](#)(uint8_t duration)
Converts coded duration into milliseconds format, refer to CC:0000.00.00.11.015.
- uint8_t

[ZAF_Actuator_GetMax](#)(s_Actuator *pActuator)
Returns value defined as maximum for `pActuator`.

Enumeration Documentation

_eActuatorState

_eActuatorState

Return type for Actuator functions that can trigger changes that require some time to be done.

Enumerator	
EACTUATOR_NOT_CHANGING	EACTUATOR_NOT_CHANGING - Already in final state.
EACTUATOR_CHANGING	EACTUATOR_CHANGING - Not in final state.
EACTUATOR_FAILED	EACTUATOR_FAILED - Requested change rejected.

Typedef Documentation

zaf_actuator_callback_t

typedef void(* zaf_actuator_callback_t) (struct actuator *pActuator) (struct actuator *pActuator)

s_Actuator

typedef struct actuator s_Actuator

The structure of Actuator module.

eActuatorState

typedef enum _eActuatorState eActuatorState

Return type for Actuator functions that can trigger changes that require some time to be done.

Function Documentation

ZAF_Actuator_Init

void ZAF_Actuator_Init ([s_Actuator](#) * pActuator, uint8_t minValue, uint8_t maxValue, uint16_t refreshRate, uint8_t durationDefault, [zaf_actuator_callback_t](#) cc_callback)

Initializes Actuator module.

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	N/A	pActuator	Pointer to Actuator to be initialized
uint8_t	N/A	minValue	Minimum possible value
uint8_t	N/A	maxValue	Maximum possible value
uint16_t	N/A	refreshRate	Minimal refresh rate in milliseconds. Tells how often the application should refresh it's state during timed changes. Actuator might adjust it to higher value, when number of steps between current and target value is low.
uint8_t	N/A	durationDefault	Factory default duration
zaf_actuator_callback_t	N/A	cc_callback	Pointer to function in 'user' command class

ZAF_Actuator_Set

```
eActuatorState ZAF_Actuator_Set (s\_Actuator * pActuator, uint8_t value, uint8_t duration)
```

Sets target value to `value` in actuator pointed by `pActuator`, during `duration` period.

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	[inout]	pActuator	Pointer to Actuator that will be set to new value
uint8_t	[in]	value	New target value
uint8_t	[in]	duration	Time period that change should take. Actual time spent depends on number of steps between current and target value and refreshRate. Requested and actual time may differ slightly.

Returns

- EACTUATOR_NOT_CHANGING if already on target value EACTUATOR_CHANGING if the change has successfully started EACTUATOR_FAILED if request was invalid

ZAF_Actuator_StartChange

```
eActuatorState ZAF_Actuator_StartChange (s\_Actuator * pActuator, bool ignoreStartLevel, bool upDown, uint8_t startLevel, uint8_t duration)
```

Starts the change on actuator pointed by `pActuator`.

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	[inout]	pActuator	Pointer to object whose value will be updated
bool	[in]	ignoreStartLevel	Should startLevel be ignored. If yes, change starts from current level.
bool	[in]	upDown	Direction of change. FALSE for UP, TRUE for DOWN.

Type	Direction	Argument Name	Description
uint8_t	[in]	startLevel	If not ignored, color will be first set to this level and then change will start
uint8_t	[in]	duration	Time period that change should take. Actual time spent depends on number of steps between current and target value and refreshRate. Requested and actual time may differ slightly.

Returns

- EACTUATOR_NOT_CHANGING if already on target value EACTUATOR_CHANGING if the change has successfully started

ZAF_Actuator_StopChange

eActuatorState ZAF_Actuator_StopChange (s_Actuator * pActuator)

Stops ongoing change on actuator pointed by pActuator .

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	N/A	pActuator	Pointer to object whose change should be stopped

Returns

- EACTUATOR_CHANGING if there was an ongoing change that was stopped, or else EACTUATOR_NOT_CHANGING

ZAF_Actuator_GetCurrentValue

uint8_t ZAF_Actuator_GetCurrentValue (s_Actuator * pActuator)

Returns current value.

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	N/A	pActuator	Pointer to Actuator struct for which current value was requested

Returns

- Current value

ZAF_Actuator_GetTargetValue

uint8_t ZAF_Actuator_GetTargetValue (s_Actuator * pActuator)

Returns target value.

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	N/A	pActuator	Pointer to Actuator struct for which target value was requested

Returns

- Target value

ZAF_Actuator_GetDurationRemaining

```
uint8_t ZAF_Actuator_GetDurationRemaining (s_Actuator * pActuator)
```

Returns estimated time until target value is reached.

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	N/A	pActuator	Pointer to Actuator struct for which remaining duration was requested

Returns

- Remaining duration in format described by requirement CC:0000.00.00.11.015 or 0 if already at target value

ZAF_Actuator_GetLastOnValue

```
uint8_t ZAF_Actuator_GetLastOnValue (s_Actuator * pActuator)
```

Gets last ON value.

Parameters

Type	Direction	Argument Name	Description
s_Actuator *	N/A	pActuator	Pointer to Actuator struct for which last On value was requested

If lastOnValue was never set, it returns zero. Then the caller has to use default ON value. **Returns**

- lastOnValue

getDurationInMs

```
uint32_t getDurationInMs (uint8_t duration)
```

Converts coded duration into milliseconds format, refer to CC:0000.00.00.11.015.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	duration	Coded duration

Returns

- duration in milliseconds

ZAF_Actuator_GetMax

```
uint8_t ZAF_Actuator_GetMax (s_Actuator * pActuator)
```

Returns value defined as maximum for `pActuator` .

Parameters

Type	Direction	Argument Name	Description
<code>s_Actuator *</code>	N/A	<code>pActuator</code>	Pointer to actuator

Returns

- Value of max

actuator

The structure of Actuator module.

Public Attributes

uint32_t	defaultRefreshRate	Step/rate counted in milliseconds - how often app refresh its state when transitions from current to target value.
uint32_t	refreshRate	Adjusted refreshRate if needed, greater than defaultRefreshRate.
uint8_t	min	Minimum value that object can have.
uint8_t	max	Maximum value that object can have.
SSwTimer	timer	Timer used while transitioning from one state to another.
uint16_t	valueCurrent	Current value of object.
uint16_t	valueTarget	End value.
zaf_actuator_callb ack_t	cc_cb	Pointer to function called when CC needs to take an action.
uint16_t	singleStepValue	Internal value used to increase/decrease valueCurrent during timed changes.
bool	directionUp	Internal value, determines should valueCurrent be increasing or decreasing.
uint8_t	lastOnValue	Last ON value before setting to OFF (when applicable)
uint8_t	durationDefault	Factory default duration, used in timed changes, when no other duration was given.

Public Attribute Documentation

defaultRefreshRate

uint32_t actuator::defaultRefreshRate

Step/rate counted in milliseconds - how often app refresh its state when transitions from current to target value.

Used as timeout in timer

refreshRate

uint32_t actuator::refreshRate

Adjusted refreshRate if needed, greater than defaultRefreshRate.

min

```
uint8_t actuator::min
```

Minimum value that object can have.

max

```
uint8_t actuator::max
```

Maximum value that object can have.

timer

```
SSwTimer actuator::timer
```

Timer used while transitioning from one state to another.

valueCurrent

```
uint16_t actuator::valueCurrent
```

Current value of object.

Internally stored as 10 times higher to get more accurate change. **Warnings**

- Always use [ZAF_Actuator_GetCurrentValue\(\)](#) to get current value

valueTarget

```
uint16_t actuator::valueTarget
```

End value.

Internally stored as 10 times higher to match current value format **Warnings**

- Always use [ZAF_Actuator_GetTargetValue\(\)](#) to get target value

cc_cb

```
zaf_actuator_callback_t actuator::cc_cb
```

Pointer to function called when CC needs to take an action.

singleStepValue

```
uint16_t actuator::singleStepValue
```

Internal value used to increase/decrease valueCurrent during timed changes.

directionUp

```
bool actuator::directionUp
```

Internal value, determines should valueCurrent be increasing or decreasing.

lastOnValue

```
uint8_t actuator::lastOnValue
```

Last ON value before setting to OFF (when applicable)

durationDefault

```
uint8_t actuator::durationDefault
```

Factory default duration, used in timed changes, when no other duration was given.

Application Timer

Application Timer

Modules

[SAppTimer](#)

Typedefs

typedef struct
[SAppTimer](#)
AppTimer object.

Functions

void [AppTimerInit](#)(uint8_t iTaskNotificationBitNumber, void *ReceiverTask)
Initialize AppTimer.

void [AppTimerSetReceiverTask](#)(void *ReceiverTask)
Configures the Receiver task of callbacks.

bool [AppTimerRegister](#)(SSwTimer *pTimer, bool bAutoReload, void(*pCallback)(SSwTimer *pTimer))
Register a SSwTimer to a TimerLiaison.

bool [AppTimerDeepSleepPersistentRegister](#)(SSwTimer *pTimer, bool bAutoReload, void(*pCallback)(SSwTimer *pTimer))
Wrapper for AppTimerRegister to register timers that should be re-loaded after deep sleep hibernate.

[ESwTimerStatus](#) [AppTimerDeepSleepPersistentStart](#)(SSwTimer *pTimer, uint32_t iTimeout)
Wrapper for [TimerStart\(\)](#) that ensures the timer value is saved so it can be re-loaded after deep sleep hibernate.

[ESwTimerStatus](#) [AppTimerDeepSleepPersistentRestart](#)(SSwTimer *pTimer)
Wrapper for [TimerRestart\(\)](#) that ensures the timer status is saved so it can be re-loaded after deep sleep hibernate.

[ESwTimerStatus](#) [AppTimerDeepSleepPersistentStop](#)(SSwTimer *pTimer)
Wrapper for [TimerStop\(\)](#) that ensures the timer status is saved so it does not get re-started after deep sleep hibernate.

void [AppTimerNotificationHandler](#)(void)
Must be called from application task when the task notification bit assigned to AppTimer is set.

void [AppTimerDeepSleepPersistentResetStorage](#)(void)
Clear storage used for persisting application timers during deep sleep hibernate.

void [AppTimerDeepSleepPersistentSaveAll](#)(void)
Save SSwTimers that should survive deep sleep Hibernate.

void [AppTimerDeepSleepPersistentLoadAll](#)(zpal_reset_reason_t resetReason)
Load SSwTimers after deep sleep hibernate wakeup.

uint32_t [AppTimerDeepSleepGetFirstRetentionRegister](#)(void)
Get index of first retention register used for saving Deep Sleep persistent app timers.

uint32_t [AppTimerDeepSleepGetLastRetentionRegister](#)(void)
Get index of last retention register used for saving Deep Sleep persistent app timers.

```
void AppTimerStopAll(void)
    Stops all the running applications timers.
```

Macros

```
#define MAX_NUM_APP_TIMERS 12
    Max number of application timers.

#define MAX_NUM_PERSISTENT_APP_TIMERS 6

#define APP_TIMER_RETENTION_REGISTER_RESERVED_COUNT (MAX_NUM_PERSISTENT_APP_TIMERS + 2)
```

Typedef Documentation

SAppTimer

typedef struct SAppTimer SAppTimer

AppTimer object.

Function Documentation

AppTimerInit

void AppTimerInit (uint8_t iTaskNotificationBitNumber, void * ReceiverTask)

Initialize AppTimer.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	iTaskNotificationBitNumber	Number defines which bit to use when notifying receiver task of pending timer event (range 0 - 31)
void *	[in]	ReceiverTask	Handle to the Application task

AppTimerSetReceiverTask

void AppTimerSetReceiverTask (void * ReceiverTask)

Configures the Receiver task of callbacks.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	ReceiverTask	Handle to the Application task

Provided as the AppTimer may be needed for registering SwTimers before the receiver task is created, meaning AppTimerInit is called prior to the Receiver task handle being available. Though not the intention, the receiver task can be changed at run time.

AppTimerRegister

```
bool AppTimerRegister (SSwTimer * pTimer, bool bAutoReload, void(*) (SSwTimer *pTimer) pCallback)
```

Register a SwTimer to a TimerLiaison.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the SwTimer object to register
bool	[in]	bAutoReload	Enable timer auto reload on timeout. Configuration of AutoReload cannot be changed after registration.
void(*) (SSwTimer *pTimer)	[in]	pCallback	Callback method of type void Callback(SSwTimer* pTimer). Argument may be NULL, in which case no callback is performed.

Initial SwTimer configuration is also performed.

Method creates a static FreeRTOS timer from the SwTimer object passed as argument, and registers it as requiring callbacks in the Receiver task. If calling method AppTimerRegister on the same SwTimer object multiple times, all but the first call will be ignored.

AppTimerDeepSleepPersistentRegister

```
bool AppTimerDeepSleepPersistentRegister (SSwTimer * pTimer, bool bAutoReload, void(*) (SSwTimer *pTimer) pCallback)
```

Wrapper for AppTimerRegister to register timers that should be re-loaded after deep sleep hibernate.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	N/A	pTimer	
bool	N/A	bAutoReload	
void(*) (SSwTimer *pTimer)	N/A	pCallback	

In addition to having AppTimerRegister() register the timer with a TimerLiaison this function also marks the timer to survive deep sleep hibernate.

For parameter descriptions See Also

- AppTimerRegister.

AppTimerDeepSleepPersistentStart

```
ESwTimerStatus AppTimerDeepSleepPersistentStart (SSwTimer * pTimer, uint32_t iTimeout)
```

Wrapper for TimerStart() that ensures the timer value is saved so it can be re-loaded after deep sleep hibernate.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	N/A	pTimer	

Type	Direction	Argument Name	Description
uint32_t	N/A	iTimeout	

For parameter descriptions **See Also**

- [TimerStart](#).

AppTimerDeepSleepPersistentRestart

ESwTimerStatus AppTimerDeepSleepPersistentRestart ([SSwTimer](#) * pTimer)

Wrapper for [TimerRestart\(\)](#) that ensures the timer status is saved so it can be re-loaded after deep sleep hibernate.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	N/A	pTimer	

For parameter descriptions **See Also**

- [TimerRestart](#).

AppTimerDeepSleepPersistentStop

ESwTimerStatus AppTimerDeepSleepPersistentStop ([SSwTimer](#) * pTimer)

Wrapper for [TimerStop\(\)](#) that ensures the timer status is saved so it does not get re-started after deep sleep hibernate.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	N/A	pTimer	

For parameter descriptions **See Also**

- [TimerStop](#).

AppTimerNotificationHandler

void AppTimerNotificationHandler (void)

Must be called from application task when the task notification bit assigned to AppTimer is set.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Method will perform pending callbacks. There is no side effects from calling method when assigned task notification bit was not set.

AppTimerDeepSleepPersistentResetStorage

```
void AppTimerDeepSleepPersistentResetStorage (void )
```

Clear storage used for persisting application timers during deep sleep hibernate.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

AppTimerDeepSleepPersistentSaveAll

```
void AppTimerDeepSleepPersistentSaveAll (void )
```

Save SSwTimers that should survive deep sleep Hibernate.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The number of remaining milliseconds until timeout are saved to the RTCC retention registers for all application timers registered to be persisted during Deep Sleep.

AppTimerDeepSleepPersistentLoadAll

```
void AppTimerDeepSleepPersistentLoadAll (zpal_reset_reason_t resetReason)
```

Load SSwTimers after deep sleep hibernate wakeup.

Parameters

Type	Direction	Argument Name	Description
zpal_reset_reason_t	N/A	resetReason	Reason why the device woke up

Activates callback registered for the timer causing the wakeup

Must be called AFTER [AppTimerInit\(\)](#) and [AppTimerDeepSleepPersistentRegister\(\)](#)

AppTimerDeepSleepGetFirstRetentionRegister

```
uint32_t AppTimerDeepSleepGetFirstRetentionRegister (void )
```

Get index of first retention register used for saving Deep Sleep persistent app timers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The retention registers are manipulated with the `zpal_retention_register_xx()` API.

AppTimerDeepSleepGetLastRetentionRegister

```
uint32_t AppTimerDeepSleepGetLastRetentionRegister (void )
```

Get index of last retention register used for saving Deep Sleep persistent app timers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

NB: The number of retention registers used depends on how many times [AppTimerDeepSleepPersistentRegister\(\)](#) have been called since system startup. It is not safe to use the returned value to locate "free" retention registers unless it is guarantees that all calls to [AppTimerDeepSleepPersistentRegister\(\)](#) has been performed.

The retention registers are manipulated with the `zpal_retention_register_xx()` API.

AppTimerStopAll

```
void AppTimerStopAll (void )
```

Stops all the running applications timers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

SAppTimer

AppTimer object.

Public Attributes

SSwTimerLiaison	TimerLiaison TimerLiaison object for Application task.
SSwTimer *	aTimerPointerArray Array for TimerLiaison - for keeping registered timers.
bool	DeepSleepPersistent Is timer persistent (persistent timers will be save/reloaded to/from retention registers during Deep Sleep hibernate)
void(*)	pDeepSleepCallback Holds the SSwTimer callback for Deep Sleep persistent timers.

Public Attribute Documentation

TimerLiaison

```
SSwTimerLiaison SAppTimer::TimerLiaison
```

TimerLiaison object for Application task.

aTimerPointerArray

```
SSwTimer* SAppTimer::aTimerPointerArray[MAX_NUM_APP_TIMERS]
```

Array for TimerLiaison - for keeping registered timers.

DeepSleepPersistent

```
bool SAppTimer::DeepSleepPersistent[MAX_NUM_APP_TIMERS]
```

Is timer persistent (persistent timers will be save/reloaded to/from retention registers during Deep Sleep hibernate)

pDeepSleepCallback

```
void(*) SAppTimer::pDeepSleepCallback[MAX_NUM_APP_TIMERS])(SSwTimer *pTimer)
```

Holds the [SSwTimer](#) callback for Deep Sleep persistent timers.

It will be called by AppTimerDeepSleepCallbackWrapper()

Command Classes List Generator

Functions

- void

[zafi_cc_list_generator_get_lists](#)(uint8_t endpoint, zaf_cc_list_t **unsecure_included_ccs, zaf_cc_list_t **secure_included_unsecure_ccs, zaf_cc_list_t **secure_included_secure_ccs)

Gets all lists based on an endpoint.
- void

[zafi_cc_list_generator_generate](#)(void)

Generates 3 CC lists for the root device and for each supported endpoint.
- void

[zafi_cc_list_generator_destroy](#)(void)

Frees the lists generate by [zafi_cc_list_generator_generate](#).

Function Documentation

zafi_cc_list_generator_get_lists

```
void zafi_cc_list_generator_get_lists (uint8_t endpoint, zaf\_cc\_list\_t ** unsecure_included_ccs, zaf\_cc\_list\_t ** secure_included_unsecure_ccs, zaf\_cc\_list\_t ** secure_included_secure_ccs)
```

Gets all lists based on an endpoint.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	endpoint	0 for root device, 1-N where N is max number of endpoints, other values are invalid.
zaf_cc_list_t **	[out]	unsecure_included_ccs	A pointer to the list of unsecure included Command Classes
zaf_cc_list_t **	[out]	secure_included_unsecure_ccs	A pointer to the list of secure included unsecure Command Classes
zaf_cc_list_t **	[out]	secure_included_secure_ccs	A pointer to the list of secure included secure Command Classes

zafi_cc_list_generator_generate

```
void zafi_cc_list_generator_generate (void )
```

Generates 3 CC lists for the root device and for each supported endpoint.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Remarks

- Must be invoked by [ZAF_Init\(\)](#).

zafi_cc_list_generator_destroy

```
void zafi_cc_list_generator_destroy (void )
```

Frees the lists generate by [zafi_cc_list_generator_generate](#).

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Remarks

- MUST NOT be invoked from the application because the CC lists are used during the lifetime of the device.

Event Manager

Event Manager

Enumerations

```
enum \_EVENT\_SYSTEM\_ {
    EVENT_SYSTEM_RESET = DEFINE_EVENT_SYSTEM_NBR
    EVENT_SYSTEM_LEARNMODE_START
    EVENT_SYSTEM_LEARNMODE_FINISHED
    EVENT_SYSTEM_LEARNMODE_STOP
    EVENT_SYSTEM_LEARNMODE_TOGGLE
    EVENT_SYSTEM_WATCHDOG_RESET
    EVENT_SYSTEM_OTA_START
    EVENT_SYSTEM_LEARNMODE_DSK_START
    EVENT_SYSTEM_FLUSHMEM_READY
    EVENT_SYSTEM_SMARTSTART_IN_PROGRESS
    EVENT_SYSTEM_EMPTY
}
```

System events.

Typedefs

```
typedef enum EVENT\_SYSTEM
\_EVENT\_SYSTEM
-
```

System events.

Functions

```
void ZAF\_eventSchedulerInit(void(*pApplicationStateMachine)(uint8_t))
    Initializes event scheduler.

bool ZCB\_eventSchedulerEventAdd(uint8_t event)
    Adds a given event to the event queue.

bool ZCB\_eventScheduler(void)
    Processes events.

bool ZAF\_jobEnqueue(uint8_t event)
    Adds a given event to the job queue.

bool ZAF\_jobDequeue(uint8_t *pEvent)
    Deque job queue.

uint8_t ZAF\_jobQueueCount(void)
    Get number of events on queue.
```

Macros

```
#define DEFINE\_EVENT\_WAKEUP\_NBR 0x01
    Event groups for application state event machine.

#define DEFINE\_EVENT\_KEYPAD\_NBR 0x10
```

```
#define DEFINE_EVENT_IR_NBR 0x20

#define DEFINE_EVENT_SYSTEM_NBR 0x30

#define DEFINE_EVENT_KEY_NBR 0x40
Event group for keys 0x40 - 0x6F.

#define DEFINE_EVENT_APP_NBR 0x70
Event group for zaf 0x80 - 0xFF.

#define EventSchedulerInit ZAF_eventSchedulerInit

#define ZCB_EventSchedulerEventAdd ZCB_eventSchedulerEventAdd

#define ZCB_EventScheduler ZCB_eventScheduler

#define ZCB_EventEnqueue ZAF_jobEnqueue

#define ZCB_EventDequeue ZAF_jobDequeue
```

Enumeration Documentation

_EVENT_SYSTEM_

```
_EVENT_SYSTEM_
```

System events.

Enumerator	
EVENT_SYSTEM_RESET	
EVENT_SYSTEM_LEARNMODE_START	
EVENT_SYSTEM_LEARNMODE_FINISHED	
EVENT_SYSTEM_LEARNMODE_STOP	
EVENT_SYSTEM_LEARNMODE_TOGGLE	
EVENT_SYSTEM_WATCHDOG_RESET	
EVENT_SYSTEM_OTA_START	
EVENT_SYSTEM_LEARNMODE_DSK_START	
EVENT_SYSTEM_FLUSHMEM_READY	
EVENT_SYSTEM_SMARTSTART_IN_PROGRESS	
EVENT_SYSTEM_EMPTY	

Typedef Documentation

EVENT_SYSTEM

```
typedef enum _EVENT_SYSTEM_ EVENT_SYSTEM
```

System events.

Function Documentation

ZAF_eventSchedulerInit

```
void ZAF_eventSchedulerInit (void*)(uint8_t) pApplicationStateMachine)
```

Initializes event scheduler.

Parameters

Type	Direction	Argument Name	Description
void*)(uint8_t)	N/A	pApplicationStateMachine	

ZCB_eventSchedulerEventAdd

```
bool ZCB_eventSchedulerEventAdd (uint8_t event)
```

Adds a given event to the event queue.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	event	A given event.

Returns

- true if given event is added to queue, false if queue full.

ZCB_eventScheduler

```
bool ZCB_eventScheduler (void )
```

Processes events.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZAF_jobEnqueue

```
bool ZAF_jobEnqueue (uint8_t event)
```

Adds a given event to the job queue.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	event	A given event.

Returns

- true if given event is added to queue, false if queue full.

ZAF_jobDequeue

```
bool ZAF_jobDequeue (uint8_t * pEvent)
```

Deque job queue.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	pEvent	return event from the queue.

Returns

- false if queue is empty else true.

ZAF_jobQueueCount

```
uint8_t ZAF_jobQueueCount (void )
```

Get number of events on queue.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- number of events on queue

MISC

MISC

Macros

```
#define ZWAVE_PLUS_TX_OPTIONS (TRANSMIT_OPTION_ACK | TRANSMIT_OPTION_AUTO_ROUTE |  
TRANSMIT_OPTION_EXPLORE)  
This define holds a default set of transmit options.
```


NVM

NVM

Functions

void	zaf_nvm_app_set_default_configuration (void) Used by the application to execute actions after a set default configuration.
void	zaf_nvm_app_reset (void) Used by the application to execute actions after a application reset.
void	zaf_nvm_app_load_configuration (void) Used by the application to execute actions after a load configuration.
void	zaf_nvm_app_load_configuration_migration (uint32_t current_version, uint32_t saved_version) Used by the application to execute actions when the current version differs from the saved version.
void	zafi_nvm_app_set_default_configuration (void) Resets configuration to default values.
void	zafi_nvm_app_reset (void) Resets the NVM file system.
void	zafi_nvm_app_load_configuration (void) Loads the default configuration common for all applications.
bool	ZAF_nvm_init (void) Opens file system for ZAF.
bool	ZAF_nvm_erase (void) Erases application nvm.
zpal_status_t	ZAF_nvm_erase_object (zpal_nvm_object_key_t key) Erases an object from application nvm.
zpal_status_t	ZAF_nvm_read (zpal_nvm_object_key_t key, void *object, size_t object_size) Reads an object from application nvm.
zpal_status_t	ZAF_nvm_read_object_part (zpal_nvm_object_key_t key, void *object, size_t offset, size_t size) Reads part of an object from application nvm.
zpal_status_t	ZAF_nvm_write (zpal_nvm_object_key_t key, const void *object, size_t object_size) Writes an object to application nvm.
zpal_status_t	ZAF_nvm_get_object_size (zpal_nvm_object_key_t key, size_t *len) Get the object size identified with a given key from NVM.
bool	ZAF_nvm_app_init (void) Opens file system for application.
bool	ZAF_nvm_app_erase (void) Erases application nvm.
zpal_status_t	ZAF_nvm_app_erase_object (zpal_nvm_object_key_t key) Erases an object from application nvm.

- zpal_status_t

ZAF_nvm_app_read(zpal_nvm_object_key_t key, void *object, size_t object_size)
Reads an object from application nvm.
- zpal_status_t

ZAF_nvm_app_read_object_part(zpal_nvm_object_key_t key, void *object, size_t offset, size_t size)
Reads part of an object from application nvm.
- zpal_status_t

ZAF_nvm_app_write(zpal_nvm_object_key_t key, const void *object, size_t object_size)
Writes an object to application nvm.
- zpal_status_t

ZAF_nvm_app_get_object_size(zpal_nvm_object_key_t key, size_t *len)
Get the object size identified with a given key from NVM.

Function Documentation

zaf_nvm_app_set_default_configuration

```
void zaf_nvm_app_set_default_configuration (void )
```

Used by the application to execute actions after a set default configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zaf_nvm_app_reset

```
void zaf_nvm_app_reset (void )
```

Used by the application to execute actions after a application reset.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zaf_nvm_app_load_configuration

```
void zaf_nvm_app_load_configuration (void )
```

Used by the application to execute actions after a load configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zaf_nvm_app_load_configuration_migration

```
void zaf_nvm_app_load_configuration_migration (uint32_t current_version, uint32_t saved_version)
```

Used by the application to execute actions when the current version differs from the saved version.

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	current_version	Version returned by zpal_get_app_version
uint32_t	N/A	saved_version	Version stored in ZAF_FILE_ID_APP_VERSION

zafi_nvm_app_set_default_configuration

```
void zafi_nvm_app_set_default_configuration (void )
```

Resets configuration to default values.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Remarks

- Internal function only

Add application specific functions here to initialize configuration values stored in persistent memory. Will be called at any of the following events:

- Network Exclusion
- Network Secure Inclusion (after S2 bootstrapping complete)
- Device Reset Locally

zafi_nvm_app_reset

```
void zafi_nvm_app_reset (void )
```

Resets the NVM file system.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Remarks

- Internal function only

zafi_nvm_app_load_configuration

```
void zafi_nvm_app_load_configuration (void )
```

Loads the default configuration common for all applications.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Remarks

- If no settings are found, the application file system will be erased and default values will be written.
- Internal function. Don't invoke from application.

ZAF_nvm_init

```
bool ZAF_nvm_init (void )
```

Opens file system for ZAF.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- bool true if [zpal_nvm_init\(\)](#) succeeded else false.

ZAF_nvm_erase

```
bool ZAF_nvm_erase (void )
```

Erases application nvm.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

(Does nothing on 800s since it has 1 shared nvm for application and protocol.)

Returns

- true on successful erase

ZAF_nvm_erase_object

```
zpal_status_t ZAF_nvm_erase_object (zpal\_nvm\_object\_key\_t key)
```

Erases an object from application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.

Returns

- status of erase operation

ZAF_nvm_read

```
zpal_status_t ZAF_nvm_read (zpal_nvm_object_key_t key, void * object, size_t object_size)
```

Reads an object from application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
void *	[out]	object	Address of array where object can be written to.
size_t	[in]	object_size	Size of the stored object.

Returns

- status of read operation

ZAF_nvm_read_object_part

```
zpal_status_t ZAF_nvm_read_object_part (zpal_nvm_object_key_t key, void * object, size_t offset, size_t size)
```

Reads part of an object from application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
void *	[out]	object	Address of array where part of object can be written to.
size_t	[in]	offset	The offset in object where part shall be read from.
size_t	[in]	size	Size of the object part.

Returns

- status of read operation

ZAF_nvm_write

```
zpal_status_t ZAF_nvm_write (zpal_nvm_object_key_t key, const void * object, size_t object_size)
```

Writes an object to application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
const void *	[out]	object	Address of array of object that is written.
size_t	[in]	object_size	Size of the stored object.

Returns

- status of write operation

ZAF_nvm_get_object_size

```
zpal_status_t ZAF_nvm_get_object_size (zpal_nvm_object_key_t key, size_t * len)
```

Get the object size identified with a given key from NVM.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
size_t *	[out]	len	Object size.

Returns

- status of operation

ZAF_nvm_app_init

```
bool ZAF_nvm_app_init (void )
```

Opens file system for application.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- bool true if [zpal_nvm_init\(\)](#) succeeded else false.

ZAF_nvm_app_erase

```
bool ZAF_nvm_app_erase (void )
```

Erases application nvm.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

(Does nothing on 800s since it has 1 shared nvm for application and protocol.)

Returns

- true on successful erase

ZAF_nvm_app_erase_object

```
zpal_status_t ZAF_nvm_app_erase_object (zpal_nvm_object_key_t key)
```

Erases an object from application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.

Returns

- status of erase operation

ZAF_nvm_app_read

```
zpal_status_t ZAF_nvm_app_read (zpal\_nvm\_object\_key\_t key, void * object, size_t object_size)
```

Reads an object from application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
void *	[out]	object	Address of array where object can be written to.
size_t	[in]	object_size	Size of the stored object.

Returns

- status of read operation

ZAF_nvm_app_read_object_part

```
zpal_status_t ZAF_nvm_app_read_object_part (zpal\_nvm\_object\_key\_t key, void * object, size_t offset, size_t size)
```

Reads part of an object from application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
void *	[out]	object	Address of array where part of object can be written to.
size_t	[in]	offset	The offset in object where part shall be read from.
size_t	[in]	size	Size of the object part.

Returns

- status of read operation

ZAF_nvm_app_write

```
zpal_status_t ZAF_nvm_app_write (zpal\_nvm\_object\_key\_t key, const void * object, size_t object_size)
```

Writes an object to application nvm.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
const void *	[out]	object	Address of array of object that is written.
size_t	[in]	object_size	Size of the stored object.

Returns

- status of write operation

ZAF_nvm_app_get_object_size

```
zpal_status_t ZAF_nvm_app_get_object_size (zpal\_nvm\_object\_key\_t key, size_t * len)
```

Get the object size identified with a given key from NVM.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_object_key_t	[in]	key	Object key.
size_t *	[out]	len	Object size.

Returns

- status of operation

True Status Engine

True Status Engine

Modules

```
_s_zaf_tse_data_input_template_t_
```

Typedefs

```
typedef struct s_zaf_tse_data_input_template_t
_s_zaf_tse_data_input_template_t

typedef void(* zaf_tse_callback_t)(zaf_tx_options_t *txOptions, void *pData)
Callback function for True Status engine.
```

Functions

```
bool ZAF_TSE_Trigger(zaf_tse_callback_t pCallback, void *pData, bool overwrite_previous_trigger)
An API for the True Status engine activation.

bool ZAF_TSE_Init(void)
Initialization function for the TSE.

void ZAF_TSE_TXCallback(transmission_result_t *pTransmissionResult)
Must be passed as a callback to zaf_transport_tx() in functions used as TSE callbacks.
```

Typedef Documentation

s_zaf_tse_data_input_template_t

```
typedef struct _s_zaf_tse_data_input_template_t s_zaf_tse_data_input_template_t
```

zaf_tse_callback_t

```
typedef void(* zaf_tse_callback_t) (zaf_tx_options_t *txOptions, void *pData) (zaf_tx_options_t *txOptions, void *pData)
```

Callback function for True Status engine.

Parameters

Type	Direction	Argument Name	Description
	[out]	txOptions	Tx options of the current transmission
	[in]	pData	Data relevant to the caller of TSE

Must be implemented by the caller of TSE

Function Documentation

ZAF_TSE_Trigger

```
bool ZAF_TSE_Trigger (zaf_tse_callback_t pCallback, void * pData, bool overwrite_previous_trigger)
```

An API for the True Status engine activation.

Parameters

Type	Direction	Argument Name	Description
zaf_tse_callback_t	[in]	pCallback	Pointer to the function to callback. The callback function must be a function taking the following arguments: 1. TRANSMIT_OPTIONS_TYPE_SINGLE_EX rxOptionsEx 2. void* pData
void *	[in]	pData	Pointer to a data struct that will be passed in argument to the pCallback function. The pData pointed struct MUST first contain a RECEIVE_OPTIONS_TYPE_EX variable indicating properties about the received frame that triggered the change. Local changes must also include a RECEIVE_OPTIONS_TYPE_EX in the pData.
bool	[in]	overwrite_previous_trigger	Boolean parameter indicating if a previous trigger with the same pCallback and the same source Endpoint in the pData should be discarded or not. Set it to true for overwriting previous triggers and false to stack up all the trigger messages.

The True Status engine will queue up the status reporting request into a queue. The status report are triggered after ZAF_TSE_DELAY_TRIGGER milliseconds.

If the pCallback was already in the queue and overwrite_previous_trigger is set to true, pData will be updated and the timer waiting ZAF_TSE_DELAY_TRIGGER will be restarted.

Returns

- True if the pCallback / pData were queued in the engine. False if the queue is full and the pCallback was not queued.

ZAF_TSE_Init

```
bool ZAF_TSE_Init (void )
```

Initialization function for the TSE.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

The function initialize the necessary timers based on the ZAF_TSE_MAXIMUM_SIMULTANEOUS_TRIGGERS setting. The status report are triggered after ZAF_TSE_DELAY_TRIGGER milliseconds.

Returns

- True if initialization completed successfully. False if initialization could not be done without errors.

ZAF_TSE_TXCallback

```
void ZAF_TSE_TXCallback (transmission_result_t * pTransmissionResult)
```

Must be passed as a callback to zaf_transport_tx() in functions used as TSE callbacks.

Parameters

Type	Direction	Argument Name	Description
transmission_result_t *	N/A	pTransmissionResult	Pointer to transmission data. The argument is not used, but exists because zaf_transport_tx() takes an argument being a pointer to a function with this type.

One example is CC_BinarySwitch_report_stx().

_s_zaf_tse_data_input_template_t_

Public Attributes

RECEIVE_OPTION rxOptions
S_TYPE_EX

Public Attribute Documentation

rxOptions

RECEIVE_OPTIONS_TYPE_EX _s_zaf_tse_data_input_template_t::rxOptions

ApplicationEvent

ApplicationEvent

Enumerations

```
enum EApplicationEvent {
    EAPPLICATIONEVENT_TIMER = 0
    EAPPLICATIONEVENT_ZWRX
    EAPPLICATIONEVENT_ZWCOMMANDSTATUS
    EAPPLICATIONEVENT_APP
    EAPPLICATIONEVENT_CC
    EAPPLICATIONEVENT_ZWRX = 0
    EAPPLICATIONEVENT_ZWCOMMANDSTATUS
    EAPPLICATIONEVENT_STATECHANGE
    EAPPLICATIONEVENT_SERIALDATARX
    EAPPLICATIONEVENT_SERIALTIMEOUT
    EAPPLICATIONEVENT_TIMER
}
```

Prioritized events that can wakeup protocol thread.

Typedefs

```
typedef enum EApplicationEvent
EApplicationEvent Prioritized events that can wakeup protocol thread.
```

Enumeration Documentation

EApplicationEvent

EApplicationEvent

Prioritized events that can wakeup protocol thread.

Enumerator	
EAPPLICATIONEVENT_TIMER	
EAPPLICATIONEVENT_ZWRX	
EAPPLICATIONEVENT_ZWCOMMANDSTATUS	
EAPPLICATIONEVENT_APP	
EAPPLICATIONEVENT_CC	
EAPPLICATIONEVENT_ZWRX	
EAPPLICATIONEVENT_ZWCOMMANDSTATUS	
EAPPLICATIONEVENT_STATECHANGE	
EAPPLICATIONEVENT_SERIALDATARX	
EAPPLICATIONEVENT_SERIALTIMEOUT	
EAPPLICATIONEVENT_TIMER	

Typedef Documentation

EApplicationEvent

```
typedef enum EApplicationEvent EApplicationEvent
```

Prioritized events that can wakeup protocol thread.

Association Data Store

Association Data Store

Modules

[s_ZAF_ADS_SecurityHighestClass_t](#)

Enumerations

```
enum e\_ZAF\_ADS\_SecurityHighestClassState\_t {  
    SECURITY_HIGHEST_CLASS_NOT_DISCOVERED  
    SECURITY_HIGHEST_CLASS_DISCOVERED_NOT_GRANTED_TO_ME  
    SECURITY_HIGHEST_CLASS_DISCOVERED  
}  
Defines the states of a highest security class discovery.  
  
enum e\_ZAF\_ADS\_CC\_Capability\_t {  
    CC_CAPABILITY_NOT_DISCOVERED  
    CC_CAPABILITY_SUPPORTED  
    CC_CAPABILITY_NOT_SUPPORTED  
}  
Defines the states of a command class discovery.  
  
enum e\_ZAF\_ADS\_CommandClass\_t {  
    ZAF_ADS_CC_SUPERVISION  
    ZAF_ADS_CC_MULTI_COMMAND  
    ZAF_ADS_MAX_CC  
}  
Defines the list of command classes for which the support can be stored.
```

Typedefs

```
typedef void * ZAF\_ADS\_Handle\_t  
Type to use when defining a handle for the ADS functions.
```

Functions

ZAF_ADS_Handle_t	ZAF_ADS_Init (void *pStorage, uint8_t MaxNodeCount) Initializes the Association Data Store.
bool	ZAF_ADS_Delete (ZAF_ADS_Handle_t handle, uint8_t NodeID) Removes all information related to a NodeID in the data store.
e_ZAF_ADS_SecurityHighestClassState_t	ZAF_ADS_GetNodeHighestSecurityClass (ZAF_ADS_Handle_t handle, uint8_t NodeID, security_key_t *pHighestClass) Returns the highest security class for a given NodeID.
bool	ZAF_ADS_SetNodeHighestSecurityClass (ZAF_ADS_Handle_t handle, uint8_t NodeID, security_key_t HighestClass) Sets the highest security class of a node.
e_ZAF_ADS_CC_Capability_t	ZAF_ADS_IsNodeSupportingCC (ZAF_ADS_Handle_t handle, uint8_t NodeID, e_ZAF_ADS_CommandClass_t

commandClass)

Returns whether a node supports a given CC or not.

bool

[ZAF_ADS_SetNodeIsSupportingCC](#)(ZAF_ADS_Handle_t handle, uint8_t NodeID, e_ZAF_ADS_CommandClass_t commandClass, e_ZAF_ADS_CC_Capability_t CC_Capability)

Sets whether a given node supports a given CC.

Macros

#define

ZAF_ADS_HEADER_SIZE

4

#define

ZAF_ADS_ELEMENT_SIZE

20

#define

ZAF_ADS_STORAGE_SIZE

(num_elements)

#define

ZAF_ADS_STORAGE

(name, num_elements)

Allocates storage at a word boundary (or any other boundary if needed)

Enumeration Documentation

e_ZAF_ADS_SecurityHighestClassState_t

e_ZAF_ADS_SecurityHighestClassState_t

Defines the states of a highest security class discovery.

Enumerator	
SECURITY_HIGHEST_CLASS_NOT_DISCOVERED	Not discovered yet.
SECURITY_HIGHEST_CLASS_DISCOVERED_NOT_GRANTED_TO_ME	The security class has been discovered but was not granted to me.
SECURITY_HIGHEST_CLASS_DISCOVERED	The Security class has been discovered and is granted to me.

e_ZAF_ADS_CC_Capability_t

e_ZAF_ADS_CC_Capability_t

Defines the states of a command class discovery.

Enumerator	
CC_CAPABILITY_NOT_DISCOVERED	Support of the CC has not yet been discovered.
CC_CAPABILITY_SUPPORTED	The CC is supported.
CC_CAPABILITY_NOT_SUPPORTED	The CC is NOT supported.

e_ZAF_ADS_CommandClass_t

e_ZAF_ADS_CommandClass_t

Defines the list of command classes for which the support can be stored.

Enumerator

ZAF_ADS_CC_SUPERVISION	Command Class Supervision.
ZAF_ADS_CC_MULTI_COMMAND	Command Class Multi Command.
ZAF_ADS_MAX_CC	Must be the last element in this enum.

Typedef Documentation

ZAF_ADS_Handle_t

```
typedef void* ZAF_ADS_Handle_t
```

Type to use when defining a handle for the ADS functions.

The handle is assigned by calling ZAF_ADS_Init.

Function Documentation

ZAF_ADS_Init

```
ZAF_ADS_Handle_t ZAF_ADS_Init (void * pStorage, uint8_t MaxNodeCount)
```

Initializes the Association Data Store.

Parameters

Type	Direction	Argument Name	Description
void *	N/A	pStorage	Pointer to allocated storage by ZAF_ADS_STORAGE. The name given in ZAF_ADS_STORAGE is what must be supplied as this argument.
uint8_t	N/A	MaxNodeCount	MaxNodeCount should be set to the max number of nodes that can be associated in controlling association groups. A controlling group is one that can transmit set or get commands. If an application has one controlling group that sends Basic Set, and the group can contain 5 associations, the MaxNodeCount should be set to 5.

This function must be invoked before all the other ADS functions.

This value must match the number of elements given in ZAF_ADS_STORAGE. **Returns**

- Returns a handle to the allocated Association Data Store.

ZAF_ADS_Delete

```
bool ZAF_ADS_Delete (ZAF\_ADS\_Handle\_t handle, uint8_t NodeID)
```

Removes all information related to a NodeID in the data store.

Parameters

Type	Direction	Argument Name	Description
ZAF_ADS_Handle_t	N/A	handle	ADS handle returned from ZAF_ADS_Init.
uint8_t	N/A	NodeID	Node ID to remove.

Returns

- Returns true if the node was removed, and false otherwise.

ZAF_ADS_GetNodeHighestSecurityClass

```
e_ZAF_ADS_SecurityHighestClassState_t ZAF_ADS_GetNodeHighestSecurityClass (ZAF_ADS_Handle_t handle, uint8_t NodeID, security_key_t * pHighestClass)
```

Returns the highest security class for a given NodeID.

Parameters

Type	Direction	Argument Name	Description
ZAF_ADS_Handle_t	N/A	handle	ADS handle returned from ZAF_ADS_Init.
uint8_t	N/A	NodeID	Node ID of which the highest security class must be read.
security_key_t *	N/A	pHighestClass	Pointer to variable where the highest security class must be written.

Returns

- Returns the state of the discovery for the given node ID.

ZAF_ADS_SetNodeHighestSecurityClass

```
bool ZAF_ADS_SetNodeHighestSecurityClass (ZAF_ADS_Handle_t handle, uint8_t NodeID, security_key_t HighestClass)
```

Sets the highest security class of a node.

Parameters

Type	Direction	Argument Name	Description
ZAF_ADS_Handle_t	N/A	handle	ADS handle returned from ZAF_ADS_Init.
uint8_t	N/A	NodeID	Node ID of which the highest security class must be set.
security_key_t	N/A	HighestClass	Security class to be set as the highest.

Returns

- Returns true if the class was set and false if there were no free slots in the ADS (increase the MaxNodeCount for the storage space used with ZAF_ADS_Init)

ZAF_ADS_IsNodeSupportingCC

```
e_ZAF_ADS_CC_Capability_t ZAF_ADS_IsNodeSupportingCC (ZAF_ADS_Handle_t handle, uint8_t NodeID, e_ZAF_ADS_CommandClass_t commandClass)
```

Returns whether a node supports a given CC or not.

Parameters

Type	Direction	Argument Name	Description
ZAF_ADS_Handle_t	N/A	handle	ADS handle returned from ZAF_ADS_Init.

Type	Direction	Argument Name	Description
uint8_t	N/A	NodeID	Node ID of which the check is made.
e_ZAF_ADS_CommandClass_t	N/A	commandClass	Command class to check support for.

Returns

- Returns the state of the CC support discovery.

ZAF_ADS_SetNodeIsSupportingCC

```
bool ZAF_ADS_SetNodeIsSupportingCC (ZAF\_ADS\_Handle\_t handle, uint8_t NodeID, e\_ZAF\_ADS\_CommandClass\_t commandClass, e\_ZAF\_ADS\_CC\_Capability\_t CC_Capability)
```

Sets whether a given node supports a given CC.

Parameters

Type	Direction	Argument Name	Description
ZAF_ADS_Handle_t	N/A	handle	ADS handle returned from ZAF_ADS_Init.
uint8_t	N/A	NodeID	Node ID of which the given CC must be set.
e_ZAF_ADS_CommandClass_t	N/A	commandClass	CC that is discovered.
e_ZAF_ADS_CC_Capability_t	N/A	CC_Capability	State of the support.

Returns

- Returns true if the state was set and false if there were no free slots in the ADS (increase the MaxNodeCount for the storage space used with ZAF_ADS_Init)

s_ZAF_ADS_SecurityHighestClass_t

Pairs a security with the discovery state of it.

Public Attributes

<code>security_key_t</code>	<code>securityKey</code>
<code>e_ZAF_ADS_SecurityHighestClassState_t</code>	<code>state</code>

Public Attribute Documentation

securityKey

`security_key_t s_ZAF_ADS_SecurityHighestClass_t::securityKey`

state

`e_ZAF_ADS_SecurityHighestClassState_t s_ZAF_ADS_SecurityHighestClass_t::state`

CC_Invoker

CC_Invoker

Typedefs

```
typedef void *      zaf\_cc\_context\_t

typedef bool(*      zaf\_cc\_invoker\_callback\_t)(CC_handler_map_latest_t const *const p_cc_entry, zaf_cc_context_t context)
    Callback type for ZAF\_CC\_foreach\(\).

typedef bool(*      zaf\_cc\_config\_invoker\_callback\_t)(zaf_cc_config_entry_latest_t const *const p_cc_config_entry, void
    *context)
    Callback type for ZAF\_CC\_config\_foreach\(\).
```

Functions

```
received\_frame\_s  ZAF\_CC\_invoke\_specific(CC_handler_map_latest_t const *const p_cc_entry, cc_handler_input_t *input,
    tatus\_t          cc\_handler\_output\_t *output)
    Invokes the handler with the correct arguments for a given command class entry.

received\_frame\_s  invoke\_cc\_handler(cc_handler_input_t *input, cc_handler_output_t *output)
    tatus\_t          Invokes a handler based on the command class in the given frame.

void              ZAF\_CC\_init\_specific(uint8_t cmdClass)
    Initializes a specific command class.

void              ZAF\_CC\_reset\_specific(uint8_t cmdClass)
    Resets specific command class.

void              ZAF\_CC\_foreach(zaf_cc_invoker_callback_t callback, zaf_cc_context_t context)
    Invokes callback for each registered CC.

size_t            ZAF\_CC\_handler\_map\_size(void)
    Returns the size of the handler map.

void              ZAF\_CC\_config\_foreach(zaf_cc_config_invoker_callback_t callback, void *context)
    Invokes callback for each registered CC configuration.

size_t            ZAF\_CC\_config\_entry\_size(void)
    Returns the size of the config entry.
```

Typedef Documentation

zaf_cc_context_t

```
typedef void* zaf_cc_context_t
```

zaf_cc_invoker_callback_t

```
typedef bool(* zaf_cc_invoker_callback_t) (CC_handler_map_latest_t const *const p_cc_entry, zaf_cc_context_t context) )
```

```
(CC_handler_map_latest_t const *const p_cc_entry, zaf_cc_context_t context)
```

Callback type for [ZAF_CC_foreach\(\)](#).

The CC handler map must always be the latest available version.

zaf_cc_config_invoker_callback_t

```
typedef bool(* zaf_cc_config_invoker_callback_t) (zaf_cc_config_entry_latest_t const *const p_cc_config_entry, void *context) )(zaf_cc_config_entry_latest_t const *const p_cc_config_entry, void *context)
```

Callback type for [ZAF_CC_config_foreach\(\)](#).

The CC handler map must always be the latest available version.

Function Documentation

ZAF_CC_invoke_specific

```
received_frame_status_t ZAF_CC_invoke_specific (CC_handler_map_latest_t const *const p_cc_entry, cc_handler_input_t * input, cc_handler_output_t * output)
```

Invokes the handler with the correct arguments for a given command class entry.

Parameters

Type	Direction	Argument Name	Description
CC_handler_map_latest_t const *const	[in]	p_cc_entry	Pointer to command class entry.
cc_handler_input_t *	[in]	input	Input data. See cc_handler_input_t for details
cc_handler_output_t *	[out]	output	Output data. See cc_handler_output_t for details

Command class handlers can be implemented in different versions having different parameters. This function will invoke the handler with the correct set of arguments based on the handler version.

Returns

- Returns `RECEIVED_FRAME_STATUS_CC_NOT_FOUND` if the handler version is not supported. Otherwise, the return value from the handler is passed through.

invoke_cc_handler

```
received_frame_status_t invoke_cc_handler (cc_handler_input_t * input, cc_handler_output_t * output)
```

Invokes a handler based on the command class in the given frame.

Parameters

Type	Direction	Argument Name	Description
cc_handler_input_t *	[in]	input	Input data. See cc_handler_input_t for details
cc_handler_output_t *	[out]	output	Output data. See cc_handler_output_t for details

This function is effectively a wrapper of [ZAF_CC_invoke_specific\(\)](#) since it finds the corresponding CC entry and invokes [ZAF_CC_invoke_specific\(\)](#).

Returns

- Returns RECEIVED_FRAME_STATUS_CC_NOT_FOUND if the CC entry is not found or of the handler version is not supported. Otherwise, the return value from the handler is passed through.

ZAF_CC_init_specific

```
void ZAF_CC_init_specific (uint8_t cmdClass)
```

Initializes a specific command class.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	cmdClass	The CC to initialize

ZAF_CC_reset_specific

```
void ZAF_CC_reset_specific (uint8_t cmdClass)
```

Resets specific command class.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	cmdClass	The CC to reset

ZAF_CC_foreach

```
void ZAF_CC_foreach (zaf_cc_invoker_callback_t callback, zaf_cc_context_t context)
```

Invokes callback for each registered CC.

Parameters

Type	Direction	Argument Name	Description
zaf_cc_invoker_callback_t	N/A	callback	Callback function to invoke.
zaf_cc_context_t	N/A	context	Context to pass on to the callback function.

Will stop if the callback returns true.

ZAF_CC_handler_map_size

```
size_t ZAF_CC_handler_map_size (void )
```

Returns the size of the handler map.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- size_t Handler map size

ZAF_CC_config_foreach

```
void ZAF_CC_config_foreach (zaf_cc_config_invoker_callback_t callback, void * context)
```

Invokes callback for each registered CC configuration.

Parameters

Type	Direction	Argument Name	Description
zaf_cc_config_invoker_callback_t	N/A	callback	Callback function to invoke.
void *	N/A	context	Context to pass on to the callback function.

Will stop if the callback returns true.

ZAF_CC_config_entry_size

```
size_t ZAF_CC_config_entry_size (void )
```

Returns the size of the config entry.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- size_t Config entry size

CMD_Publisher

CMD_Publisher

Typedefs

- typedef void *

[CP_Handle_t](#)

Handle to the Cmd Publisher "object".
- typedef void(*

[zaf_cp_subscriberFunction_t](#)

(void *pSubscriberContext, void *pRxPackage)
Callback function triggered by ZAF_CP_CommandPublish, when corresponding subscriber gets notified.

Functions

- [CP_Handle_t](#)

[ZAF_CP_Init](#)

(void *pStorage, uint8_t numSubscribers)
Initializes Command publisher module.
- bool

[ZAF_CP_SubscribeToAll](#)

(CP_Handle_t handle, void *pSubscriberContext, zaf_cp_subscriberFunction_t pFunction)
Subscribe a function with pHandle to all command classes.
- bool

[ZAF_CP_SubscribeToCC](#)

(CP_Handle_t handle, void *pSubscriberContext, zaf_cp_subscriberFunction_t pFunction, uint16_t CmdClass)
Subscribe a function with pHandle to an entire command class.
- bool

[ZAF_CP_SubscribeToCmd](#)

(CP_Handle_t handle, void *pSubscriberContext, zaf_cp_subscriberFunction_t pFunction, uint16_t CmdClass, uint8_t Cmd)
Subscribe a function with pHandle to a specific command of the given command class.
- bool

[ZAF_CP_UnsubscribeToAll](#)

(CP_Handle_t handle, void *pSubscriberContext, zaf_cp_subscriberFunction_t pFunction)
Unsubscribe from previously subscribed handle.
- bool

[ZAF_CP_UnsubscribeToCC](#)

(CP_Handle_t handle, void *pSubscriberContext, zaf_cp_subscriberFunction_t pFunction, uint16_t CmdClass)
Unsubscribe from previously subscribed command class.
- bool

[ZAF_CP_UnsubscribeToCmd](#)

(CP_Handle_t handle, void *pSubscriberContext, zaf_cp_subscriberFunction_t pFunction, uint16_t CmdClass, uint8_t Cmd)
Unsubscribe from previously subscribed command.
- void

[ZAF_CP_CommandPublish](#)

(CP_Handle_t handle, void *pRxPackage)
Informs all relevant subscribers about incoming frame.

Macros

- #define

ZAF_CP_HEADER_SIZE

4
- #define

ZAF_CP_ELEMENT_SIZE

16
- #define

ZAF_CP_STORAGE_SIZE

(num_elements)
- #define

ZAF_CP_STORAGE

(name, num_elements)

Typedef Documentation

CP_Handle_t

```
typedef void* CP_Handle_t
```

Handle to the Cmd Publisher "object".

zaf_cp_subscriberFunction_t

```
typedef void(* zaf_cp_subscriberFunction_t) (void *pSubscriberContext, void *pRxPackage) (void *pSubscriberContext, void *pRxPackage)
```

Callback function triggered by ZAF_CP_CommandPublish, when corresponding subscriber gets notified.

Parameters

Type	Direction	Argument Name	Description
	N/A	pSubscriberContext	Subscribers context to be passed to subscriber
	N/A	pRxPackage	Received Frame

Function Documentation

ZAF_CP_Init

```
CP_Handle_t ZAF_CP_Init (void * pStorage, uint8_t numSubscribers)
```

Initializes Command publisher module.

Parameters

Type	Direction	Argument Name	Description
void *	N/A	pStorage	Stores the subscribers list. Should be allocated with ZAF_CP_STORAGE
uint8_t	N/A	numSubscribers	Number of subscribers

Note

- ZAF_CP_STORAGE must be called before this

Returns

- handle to the initialized content

ZAF_CP_SubscribeToAll

```
bool ZAF_CP_SubscribeToAll (CP_Handle_t handle, void * pSubscriberContext, zaf_cp_subscriberFunction_t pFunction)
```

Subscribe a function with pHandle to all command classes.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	Handle to Cmd Publisher module
void *	N/A	pSubscriberContext	Context of the subscriber
zaf_cp_subscriberFunction_t	N/A	pFunction	Subscribers callback function

Returns

- true if new subscription went OK, false if not

ZAF_CP_SubscribeToCC

```
bool ZAF_CP_SubscribeToCC (CP_Handle_t handle, void * pSubscriberContext, zaf_cp_subscriberFunction_t pFunction,
uint16_t CmdClass)
```

Subscribe a function with pHandle to an entire command class.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	Handle to Cmd Publisher module
void *	N/A	pSubscriberContext	Context of the subscriber
zaf_cp_subscriberFunction_t	N/A	pFunction	Subscribers callback function
uint16_t	N/A	CmdClass	Command Class to subscribe to

Returns

- true if new subscription went OK, false if not

ZAF_CP_SubscribeToCmd

```
bool ZAF_CP_SubscribeToCmd (CP_Handle_t handle, void * pSubscriberContext, zaf_cp_subscriberFunction_t pFunction,
uint16_t CmdClass, uint8_t Cmd)
```

Subscribe a function with pHandle to a specific command of the given command class.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	Handle to Cmd Publisher module
void *	N/A	pSubscriberContext	Context of the subscriber
zaf_cp_subscriberFunction_t	N/A	pFunction	Subscribers callback function
uint16_t	N/A	CmdClass	Command Class to subscribe to
uint8_t	N/A	Cmd	Specific command to subscribe to

Returns

- true if new subscription went OK, false if not

ZAF_CP_UnsubscribeToAll

```
bool ZAF_CP_UnsubscribeToAll (CP_Handle_t handle, void * pSubscriberContext, zaf_cp_subscriberFunction_t pFunction)
```

Unsubscribe from previously subscribed handle.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	Handle to Cmd Publisher module
void *	N/A	pSubscriberContext	Context of the subscriber
zaf_cp_subscriberFunction_t	N/A	pFunction	Subscribers callback function

Returns

- true if unsubscribing went OK, false if not

ZAF_CP_UnsubscribeToCC

```
bool ZAF_CP_UnsubscribeToCC (CP_Handle_t handle, void * pSubscriberContext, zaf_cp_subscriberFunction_t pFunction, uint16_t CmdClass)
```

Unsubscribe from previously subscribed command class.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	Handle to Cmd Publisher module
void *	N/A	pSubscriberContext	Context of the subscriber
zaf_cp_subscriberFunction_t	N/A	pFunction	Subscribers callback function
uint16_t	N/A	CmdClass	Command Class to unsubscribe from

Returns

- true if unsubscribing went OK, false if not

ZAF_CP_UnsubscribeToCmd

```
bool ZAF_CP_UnsubscribeToCmd (CP_Handle_t handle, void * pSubscriberContext, zaf_cp_subscriberFunction_t pFunction, uint16_t CmdClass, uint8_t Cmd)
```

Unsubscribe from previously subscribed command.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	Handle to Cmd Publisher module
void *	N/A	pSubscriberContext	Context of the subscriber
zaf_cp_subscriberFunction_t	N/A	pFunction	Subscribers callback function
uint16_t	N/A	CmdClass	Command Class to unsubscribe from
uint8_t	N/A	Cmd	Specific command to unsubscribe from

Returns

- true if unsubscribing went OK, false if not

ZAF_CP_CommandPublish

```
void ZAF_CP_CommandPublish (CP_Handle_t handle, void * pRxPackage)
```

Informs all relevant subscribers about incoming frame.

Parameters

Type	Direction	Argument Name	Description
CP_Handle_t	N/A	handle	Handle to Cmd Publisher module
void *	N/A	pRxPackage	Received frame. It will be passed to subscribers

Based on incoming frame, it will decide which type of subscribers should be notified It should be called when the receive queue is emptied

Command Class Utils

Command Class Utils

Functions

```
bool CmdClassSupported(security_key_t eKey, uint8_t commandClass, uint8_t command, uint8_t *pSecurelist,
uint8_t securelistLen, uint8_t *pNonSecurelist, uint8_t nonSecurelistLen)
Returns whether a given command is supported based on command class lists and different rules.
```

Function Documentation

CmdClassSupported

```
bool CmdClassSupported (security_key_t eKey, uint8_t commandClass, uint8_t command, uint8_t * pSecurelist, uint8_t
securelistLen, uint8_t * pNonSecurelist, uint8_t nonSecurelistLen)
```

Returns whether a given command is supported based on command class lists and different rules.

Parameters

Type	Direction	Argument Name	Description
security_key_t	[in]	eKey	The security key that the frame was received with.
uint8_t	[in]	commandClass	The CC of the frame.
uint8_t	[in]	command	The command of the frame.
uint8_t *	[in]	pSecurelist	Pointer to the secure CC list.
uint8_t	[in]	securelistLen	The length of the secure CC list.
uint8_t *	[in]	pNonSecurelist	Pointer to the non-secure CC list.
uint8_t	[in]	nonSecurelistLen	Length of the non-secure CC list.

Returns

- Returns true if the command is supported and false otherwise.

FILE_ID

FILE_ID

ZAF File IDs.

- MUST NOT be changed.

Macros

```
#define ZAF_FILE_ID_APP_VERSION (0)

#define ZAF_FILE_ID_ASSOCIATIONINFO (1)

#define ZAF_FILE_ID_USERCODE (2)

#define ZAF_FILE_ID_BATTERYDATA (3)

#define ZAF_FILE_ID_NOTIFICATIONDATA (4)

#define ZAF_FILE_ID_WAKEUPCCDATA (5)

#define ZAF_FILE_ID_CC_FIRMWARE_UPDATE (6)

#define ZAF_FILE_ID_CC_CONFIGURATION (7)

#define ZAF_FILE_ID_KEY_FOB_NODES (8)

#define ZAF_FILE_ID_CC_DOOR_LOCK (9)

#define ZAF_FILE_ID_BASIC_SET (10)

#define ZAF_FILE_ID_CENTRAL_SCENE_CONFIG (11)

#define ZAF_FILE_ID_APP_NAME (12)

#define ZAF_FILE_ID_CC_USER_CODE_HEADER (13)

#define ZAF_FILE_ID_CC_CONFIGURATION_BASE (100)

#define ZAF_FILE_ID_CC_CONFIGURATION_LAST (199)

#define ZAF_FILE_ID_CC_USER_CODE_BASE (200)

#define ZAF_FILE_ID_CC_USER_CODE_LAST (455)

#define ZAF_FILE_ID_CC_COLOR_SWITCH_BASE (456)

#define ZAF_FILE_ID_CC_COLOR_SWITCH_LAST (711)

#define ZAF_FILE_ID_CC_MULTILEVEL_SWITCH_BASE (712)

#define ZAF_FILE_ID_CC_MULTILEVEL_SWITCH_LAST (967)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_NUMBER_OF_USERS (968)
Data object and database descriptor table file ID numbers for CC_UserCredential.

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_NUMBER_OF_CREDENTIALS (969)
```

```
#define ZAF_FILE_ID_CC_USER_CREDENTIAL_USER_DESCRIPTOR_TABLE (970)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_USER_BASE (971)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_USER_LAST (1226)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_USER_NAME_BASE (1227)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_USER_NAME_LAST (1482)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_CREDENTIAL_DESCRIPTOR_TABLE (1483)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_CREDENTIAL_BASE (1484)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_CREDENTIAL_LAST (1739)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_CREDENTIAL_DATA_BASE (1740)

#define ZAF_FILE_ID_CC_USER_CREDENTIAL_CREDENTIAL_DATA_LAST (1995)

#define ZAF_FILE_ID_ADMIN_PIN_CODE (1996)
Admin PIN Code data.

#define ZAF_FILE_SIZE_APP_VERSION (sizeof(uint32_t))
```


Network

Network

Modules

[Network Learn Mode](#)

[Network Management](#)

Network Learn Mode

Network Learn Mode

Functions

```
void ZAF_setNetworkLearnMode(E_NETWORK_LEARN_MODE_ACTION bMode)
    Sets network learn mode to given value.
```

Function Documentation

ZAF_setNetworkLearnMode

```
void ZAF_setNetworkLearnMode (E_NETWORK_LEARN_MODE_ACTION bMode)
```

Sets network learn mode to given value.

Parameters

Type	Direction	Argument Name	Description
E_NETWORK_LEARN_MODE_ACTION	N/A	bMode	Network mode to be set. Refer to E_NETWORK_LEARN_MODE_ACTION

This function is called when node should be added or removed from the network, or to enable Smart Start.

Network Management

Network Management

Functions

- void

[ZAF_SetMaxInclusionRequestIntervals](#)(uint32_t intervals)

Command to set the maximum inclusion request interval.
- void

[ZAF_SendINIF](#)(void(*pCallback)(uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus))

Broadcasts an Included Node Information Frame (INIF).

Function Documentation

ZAF_SetMaxInclusionRequestIntervals

```
void ZAF_SetMaxInclusionRequestIntervals (uint32_t intervals)
```

Command to set the maximum inclusion request interval.

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	intervals	The maximum number of seconds in units of 128sec/tick in between SmartStart inclusion requests. Valid range 0 and 5-99. 0 is default value and corresponds to 512 sec. The range 5-99 corresponds to 640-12672sec.

A status message with type=EZWAVECOMMANDSTATUS_ZW_SET_MAX_INCL_REQ_INTERVALS is returned to the application command status queue when the command has completed.

ZAF_SendINIF

```
void ZAF_SendINIF (void(*) (uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus) pCallback)
```

Broadcasts an Included Node Information Frame (INIF).

Parameters

Type	Direction	Argument Name	Description
void(*) (uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus)	N/A	pCallback	Pointer to a callback function that will be invoked when transmission is successful or has failed.

Retention Register

Retention Register

Functions

- `zpal_status_t` `ZAF_retention_register_read`(uint32_t index, uint32_t *data)
Reads a 32-bit value from the specified retention register.
- `zpal_status_t` `ZAF_retention_register_write`(uint32_t index, uint32_t value)
Write a 32-bit value to the specified retention register.
- `size_t` `ZAF_retention_register_count`(void)
Get number of retention registers available for application.

Function Documentation

ZAF_retention_register_read

```
zpal_status_t ZAF_retention_register_read (uint32_t index, uint32_t * data)
```

Reads a 32-bit value from the specified retention register.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	index	Retention register number (zero-based).
uint32_t *	[out]	data	Pointer to a 32-bit variable where the value can be stored.

Returns

- `ZPAL_STATUS_OK` on success, `ZPAL_STATUS_INVALID_ARGUMENT` on invalid `index` or `data` and `ZPAL_STATUS_FAIL` otherwise.

ZAF_retention_register_write

```
zpal_status_t ZAF_retention_register_write (uint32_t index, uint32_t value)
```

Write a 32-bit value to the specified retention register.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	index	Retention register number (zero-based).
uint32_t	[in]	value	32-bit value to save in retention register.

Returns

- `ZPAL_STATUS_OK` on success, `ZPAL_STATUS_INVALID_ARGUMENT` on invalid `index` and `ZPAL_STATUS_FAIL` otherwise.

ZAF_retention_register_count

```
size_t ZAF_retention_register_count (void )
```

Get number of retention registers available for application.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- number of retention registers.

ZAF_Transport

ZAF_Transport

Modules

- [ZAF_Transport_Endpoint](#)
- [ZAF_Transport_Multicast](#)
- [ZAF_Transport_Security](#)

Functions

bool

[ZAF_transportSendDataAbort](#)(void)

Instructs Z-Wave protocol to Abort an Application initiated transmit if any in progress.

Function Documentation

ZAF_transportSendDataAbort

bool ZAF_transportSendDataAbort (void)

Instructs Z-Wave protocol to Abort an Application initiated transmit if any in progress.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZAF_Transport_Endpoint

ZAF_Transport_Endpoint

Modules

- [_MULTICHAN_NODE_INFO_](#)
- [_MULTICHAN_NODE_INFO_PACKED_](#)
- [destination_info_t](#)
- [MULTICHAN_NODE_ID_PACKED](#)
- [_TRANSMIT_OPTIONS_TYPE_EX_](#)
- [_TRANSMIT_OPTIONS_TYPE_SINGLE_EX_](#)

Enumerations

```
enum EZAF_EnqueueStatus_t {
    ZAF_ENQUEUE_STATUS_SUCCESS = 0
    ZAF_ENQUEUE_STATUS_TIMEOUT = 1
    ZAF_ENQUEUE_STATUS_BUFFER_OVERRUN = 2
}
Defines return values for ZAF_Transmit.

enum _ENDPOINT_ {
    ENDPOINT_ROOT = 0
    ENDPOINT_1
    ENDPOINT_2
    ENDPOINT_3
    ENDPOINT_4
    ENDPOINT_5
    ENDPOINT_6
    ENDPOINT_7
    ENDPOINT_8
    ENDPOINT_9
}
```

Typedefs

- [typedef struct _MULTICHAN_NODE_INFO_](#) [MULTICHAN_NODE_INFO](#)
MULTICHAN_NODE_INFO should include feature node informations as CRC, security etc.
- [typedef struct _MULTICHAN_NODE_INFO_PACKED_](#) [MULTICHAN_NODE_INFO_PACKED](#)
- [typedef destination_info_t](#) [MULTICHAN_NODE_ID](#)
- [typedef struct _TRANSMIT_OPTIONS_TYPE_EX_](#) [TRANSMIT_OPTIONS_TYPE_EX](#)

```
typedef struct
_TRANSMIT_OPTIONS_TYPE_SINGLE_EX

typedef enum
_ENDPOINT_
```

Functions

void	ZW_TransportEndpoint_Init (void) Initializes the ZAF Transport Layer.
zaf_cc_list_t *	GetEndpointcmdClassList (bool secList, uint8_t endpoint) Get endpoint non-secure or secure command class list.
EZAF_EnqueueStatus_t	ZAF_Transmit (uint8_t *pData, size_t dataLength, TRANSMIT_OPTIONS_TYPE_SINGLE_EX *pTxOptionsEx, ZAF_TX_Callback_t pCallback) Send data.
received_frame_status_t	Transport_ApplicationCommandHandlerEx (RECEIVE_OPTIONS_TYPE_EX *rxOpt, ZW_APPLICATION_TX_BUFFER *pCmd, uint8_t cmdLength) Invokes a command class handler based on the first byte in the frame.
void	RxToTxOptions (RECEIVE_OPTIONS_TYPE_EX *rxopt, TRANSMIT_OPTIONS_TYPE_SINGLE_EX **txopt) Helper function to switching endpoints for SendData.
bool	Check_not_legal_response_job (RECEIVE_OPTIONS_TYPE_EX *rxOpt) Validate destination endpoint use bit-addressing.
bool	is_multicast (RECEIVE_OPTIONS_TYPE_EX *rxOpt) This function indicates if the frame was received using multicast addressing The Z-Wave Multicast frame, the broadcast NodeID and the Multi Channel multi-End Point destination are all considered multicast addressing methods.
void	SetFlagSupervisionEncap (bool flag) Set supervision-encapsulated-frame flag.

Enumeration Documentation

EZAF_EnqueueStatus_t

```
EZAF_EnqueueStatus_t
```

Defines return values for ZAF_Transmit.

They match the enum values defined in EQueueNotifyingStatus (i.e., EQUEUENOTIFYING_STATUS_SUCCESS and EQUEUENOTIFYING_STATUS_TIMEOUT, respectively).

Enumerator	
ZAF_ENQUEUE_STATUS_SUCCESS	Item successfully put on queue.
ZAF_ENQUEUE_STATUS_TIMEOUT	Failed to put item on queue as it was full.
ZAF_ENQUEUE_STATUS_BUFFER_OVERRUN	Failed to put item on queue as it was to long to fit in tx buffer.

ENDPOINT

```
_ENDPOINT_
```


Enumerator

ENDPOINT_ROOT	
ENDPOINT_1	
ENDPOINT_2	
ENDPOINT_3	
ENDPOINT_4	
ENDPOINT_5	
ENDPOINT_6	
ENDPOINT_7	
ENDPOINT_8	
ENDPOINT_9	

Typedef Documentation

MULTICHAN_NODE_INFO

```
typedef struct _MULTICHAN_NODE_INFO_ MULTICHAN_NODE_INFO
```

MULTICHAN_NODE_INFO should include feature node informations as CRC, security etc.

MULTICHAN_NODE_INFO_PACKED

```
typedef struct _MULTICHAN_NODE_INFO_PACKED_ MULTICHAN_NODE_INFO_PACKED
```

MULTICHAN_NODE_ID

```
typedef destination_info_t MULTICHAN_NODE_ID
```

TRANSMIT_OPTIONS_TYPE_EX

```
typedef struct _TRANSMIT_OPTIONS_TYPE_EX_ TRANSMIT_OPTIONS_TYPE_EX
```

TRANSMIT_OPTIONS_TYPE_SINGLE_EX

```
typedef struct _TRANSMIT_OPTIONS_TYPE_SINGLE_EX_ TRANSMIT_OPTIONS_TYPE_SINGLE_EX
```

ENDPOINT

```
typedef enum _ENDPOINT_ ENDPOINT
```

Function Documentation

ZW_TransportEndpoint_Init

```
void ZW_TransportEndpoint_Init (void )
```

Initializes the ZAF Transport Layer.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

GetEndpointcmdClassList

```
zaf_cc_list_t * GetEndpointcmdClassList (bool secList, uint8_t endpoint)
```

Get endpoint non-secure or secure command class list.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	secList	true if secure list else false for non-secure list
uint8_t	[in]	endpoint	parameter

Returns

- Return endpoint command class list. Return NULL if list is not present!

ZAF_Transmit

```
EZAF_EnqueueStatus_t ZAF_Transmit (uint8_t * pData, size_t dataLength, TRANSMIT_OPTIONS_TYPE_SINGLE_EX * pTxOptionsEx, ZAF_TX_Callback_t pCallback)
```

Send data.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[in]	pData	IN Data buffer pointer.
size_t	[in]	dataLength	IN Data buffer length.
TRANSMIT_OPTIONS_TYPE_SINGLE_EX *	[in]	pTxOptionsEx	transmit options pointer.
ZAF_TX_Callback_t	[out]	pCallback	is a callback function-pointer returning result of the job. Please note that both bStatus TRANSMIT_COMPLETE_OK and TRANSMIT_COMPLETE_VERIFIED indicates successful transmission.

This function is used both as response to a command and for sending unsolicited frames Encapsulate data for endpoint support and call protocol ZW_SendDataEx.

Returns

- status of enum type EZAF_EnqueueStatus_t

Transport_ApplicationCommandHandlerEx

```
received_frame_status_t Transport_ApplicationCommandHandlerEx (RECEIVE_OPTIONS_TYPE_EX * rxOpt,
ZW_APPLICATION_TX_BUFFER * pCmd, uint8_t cmdLength)
```

Invokes a command class handler based on the first byte in the frame.

Parameters

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE_EX *	[in]	rxOpt	IN receive options of type RECEIVE_OPTIONS_TYPE_EX.
ZW_APPLICATION_TX_BUFFER *	[in]	pCmd	IN Payload from the received frame.
uint8_t	[in]	cmdLength	IN Number of command bytes including command.

Must be defined in the application.

DeprecatedIf all command classes use one of the REGISTER_CC macros, this function is not required to be implemented in the application. There exists a weak definition of it. ZAF_CC_Invoker will take care of invoking the right command class handler.

Returns

- status of extracting frame

RxToTxOptions

```
void RxToTxOptions (RECEIVE_OPTIONS_TYPE_EX * rxopt, TRANSMIT_OPTIONS_TYPE_SINGLE_EX ** txopt)
```

Helper function to switching endpoints for SendData.

Parameters

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE_EX *	[in]	rxopt	Receive options to convert.
TRANSMIT_OPTIONS_TYPE_SINGLE_EX **	[out]	txopt	Converted transmit options

Check_not_legal_response_job

```
bool Check_not_legal_response_job (RECEIVE_OPTIONS_TYPE_EX * rxOpt)
```

Validate destination endpoint use bit-addressing.

Parameters

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE_EX *	[in]	rxOpt	pointer of type RECEIVE_OPTIONS_TYPE_EX

This function must only be used to check jobs that return response (Get <-> Report). **Returns**

- Boolean true if use bit-addressing else false

is_multicast

```
bool is_multicast (RECEIVE_OPTIONS_TYPE_EX * rxOpt)
```

This function indicates if the frame was received using multicast addressing The Z-Wave Multicast frame, the broadcast NodeID and the Multi Channel multi-End Point destination are all considered multicast addressing methods.

Parameters

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE_EX *	[in]	rxOpt	pointer of type RECEIVE_OPTIONS_TYPE_EX

Returns

- Boolean true if use bit-addressing else false

SetFlagSupervisionEncap

```
void SetFlagSupervisionEncap (bool flag)
```

Set supervision-encapsulated-frame flag.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	flag	supervision-encapsulated-frame true if active and false if deactive.

The Command class call [Check_not_legal_response_job\(\)](#) Use the flag to discard current job if a Get<->Response job (CC:006C.01.00.13.002).

_MULTICHAN_NODE_INFO_

MULTICHAN_NODE_INFO should include feature node informations as CRC, security etc.

Public Attributes

uint8_t	BitMultiChannelEncap
	uint8_t 0, bit 0
security_key_t	security
	of type security_key_t

Public Attribute Documentation

BitMultiChannelEncap

uint8_t _MULTICHAN_NODE_INFO::BitMultiChannelEncap

uint8_t 0, bit 0

security

security_key_t _MULTICHAN_NODE_INFO::security

of type security_key_t

_MULTICHAN_NODE_INFO_PACKED_

Public Attributes

uint8_t	security	bit 0-3 of type security_key_t: 0-NON_KEY,1-S2_UNAUTHENTICATED, 2-S2_AUTHENTICATED, 3-S2_ACCESS, 4-S0 (security_key_t)
uint8_t	unused	bit 4
uint8_t	BitMultiChannelEncap	bit 5
uint8_t	unused2	bit 6
uint8_t	unused3	bit 7

Public Attribute Documentation

security

uint8_t _MULTICHAN_NODE_INFO_PACKED_::security

bit 0-3 of type security_key_t: 0-NON_KEY,1-S2_UNAUTHENTICATED, 2-S2_AUTHENTICATED, 3-S2_ACCESS, 4-S0 (security_key_t)

unused

uint8_t _MULTICHAN_NODE_INFO_PACKED_::unused

bit 4

BitMultiChannelEncap

uint8_t _MULTICHAN_NODE_INFO_PACKED_::BitMultiChannelEncap

bit 5

unused2

uint8_t _MULTICHAN_NODE_INFO_PACKED_::unused2

bit 6

unused3

uint8_t _MULTICHAN_NODE_INFO_PACKED_::unused3

bit 7

destination_info_t

Association node data field in a association group.

Public Attributes

MULTICHAN_DEST_NODE_ID	node
MULTICHAN_NODE_INFO	nodeInfo

Public Attribute Documentation

node

MULTICHAN_DEST_NODE_ID destination_info_t::node

nodeInfo

MULTICHAN_NODE_INFO destination_info_t::nodeInfo

MULTICHAN_NODE_ID_PACKED

Public Attributes

MULTICHAN_DEST_NODE_ID_8bit	node
MULTICHAN_NODE_INFO_PACKED	nodeInfoPacked

Public Attribute Documentation

node

MULTICHAN_DEST_NODE_ID_8bit MULTICHAN_NODE_ID_PACKED::node

nodeInfoPacked

MULTICHAN_NODE_INFO_PACKED MULTICHAN_NODE_ID_PACKED::nodeInfoPacked

_TRANSMIT_OPTIONS_TYPE_EX_

Public Attributes

uint8_t	S2_groupID
uint8_t	txOptions
uint8_t	sourceEndpoint
MULTICHAN_NO DE_ID *	pList
uint8_t	list_length

Public Attribute Documentation

S2_groupID

```
uint8_t _TRANSMIT_OPTIONS_TYPE_EX::S2_groupID
```

txOptions

```
uint8_t _TRANSMIT_OPTIONS_TYPE_EX::txOptions
```

sourceEndpoint

```
uint8_t _TRANSMIT_OPTIONS_TYPE_EX::sourceEndpoint
```

pList

```
MULTICHAN_NODE_ID* _TRANSMIT_OPTIONS_TYPE_EX::pList
```

list_length

```
uint8_t _TRANSMIT_OPTIONS_TYPE_EX::list_length
```

_TRANSMIT_OPTIONS_TYPE_SINGLE_EX_

Public Attributes

uint8_t	txOptions
uint8_t	txSecOptions Options for enabling specific Security scheme functionality.
uint8_t	sourceEndpoint
MULTICHAN_NO_DE_ID *	pDestNode

Public Attribute Documentation

txOptions

```
uint8_t _TRANSMIT_OPTIONS_TYPE_SINGLE_EX::txOptions
```

txSecOptions

```
uint8_t _TRANSMIT_OPTIONS_TYPE_SINGLE_EX::txSecOptions
```

Options for enabling specific Security scheme functionality.

sourceEndpoint

```
uint8_t _TRANSMIT_OPTIONS_TYPE_SINGLE_EX::sourceEndpoint
```

pDestNode

```
MULTICHAN_NODE_ID* _TRANSMIT_OPTIONS_TYPE_SINGLE_EX::pDestNode
```

ZAF_Transport_Multicast

ZAF_Transport_Multicast

Enumerations

```
enum ETRANSPORT_MULTICAST_STATUS {
    ETRANSPORTMULTICAST_FAILED = 0
    ETRANSPORTMULTICAST_ADDED_TO_QUEUE = 1
}
```

Functions

```
enum ETRANSPORT_MULTICAST_STATUS ZW_TransportMulticast_SendRequest(const uint8_t *const p_data, uint8_t data_length, uint8_t fSupervisionEnable, TRANSMIT_OPTIONS_TYPE_EX *p_nodelist, ZAF_TX_Callback_t p_callback)
    Initiates transmission of a frame and will handle multi channel and multicast transmissions.

void ZW_TransportMulticast_clearTimeout(void)

void ZW_TransportMulticast_init(void)
    Initializes the Tx Buffer.
```

Enumeration Documentation

ETRA...MULTICAST_STATUS

ETRA...MULTICAST_STATUS

Enumerator

ETRA...MULTICAST_FAILED	
ETRA...MULTICAST_ADDED_TO_QUEUE	

Function Documentation

ZW_TransportMulticast_SendRequest

enum ETRANSPORT_MULTICAST_STATUS ZW_TransportMulticast_SendRequest (const uint8_t *const p_data, uint8_t data_length, uint8_t fSupervisionEnable, TRANSMIT_OPTIONS_TYPE_EX * p_nodelist, ZAF_TX_Callback_t p_callback)

Initiates transmission of a frame and will handle multi channel and multicast transmissions.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *const	[in]	p_data	Pointer to data.
uint8_t	[in]	data_length	Length of data in bytes.
uint8_t	[in]	fSupervisionEnable	Enable Supervision encapsulation if set to true.

Type	Direction	Argument Name	Description
TRANSMIT_OPTIONS_TYPE_EX *	[in]	p_nodelist	Pointer to a list of nodes.
ZAF_TX_Callback_t	[in]	p_callback	Pointer to a callback function which is called for each transmission to a node.

If the node has associations to multiple endpoints of the same node, this function will transmit one multi channel encapsulated frame that addresses the associated endpoints.

If the node is included using S2 and has more than one non-endpoint association, this function will transmit an S2 multicast (broadcast) frame addressing those associations followed by singlecast follow-up frames.

If the node is included using S0 there will be no multicast since S0 does not support multicast.

If the node is non-securely included and has more than one non-endpoint association, this function will transmit a non-secure multicast followed by singlecast follow-up frames.

Returns

- Returns status of transmission.

ZW_TransportMulticast_clearTimeout

```
void ZW_TransportMulticast_clearTimeout (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZW_TransportMulticast_init

```
void ZW_TransportMulticast_init (void )
```

Initializes the Tx Buffer.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZAF_Transport_Security

ZAF_Transport_Security

Functions

- void

[Transport_ApplicationCommandHandler](#)(ZW_APPLICATION_TX_BUFFER *pCmd, uint8_t cmdLength, RECEIVE_OPTIONS_TYPE *rxOpt)
Called when the frame is received.
- void

[ApplicationCommandHandler](#)(void *pSubscriberContext, SZwaveReceivePackage *pRxPackage)
Callback function triggered by Cmd Publisher when the frame is received from protocol See [ZAF_CmdPublisher.h](#) for more details.
- uint8_t

[Transport_OnApplicationInitSW](#)(void)
Initializes the ZAF transport layer.
- uint8_t

[Transport_OnLearnCompleted](#)(node_id_t nodeId)
Register the new node id in the transport layer after learn mode completes.
- void

[ZAF_Transport_OnLearnCompleted](#)(void)
Updates the active NIF depending on learn mode result.
- bool

[TransportCmdClassSupported](#)(uint8_t commandClass, uint8_t command, enum SECURITY_KEY eKey)
Returns whether a given command is supported.
- enum
SECURITY_KEY

[GetHighestSecureLevel](#)(uint8_t protocolSecBits)
Get highest secure level.
- [zaf_cc_list_t](#) *

[GetCommandClassList](#)(bool included, security_key_t eKey, uint8_t endpoint)
Get command class list from device.

Macros

- #define

APPL_NODEPARAM_MAX 35
Max number of command classes a command class list can contain.

Function Documentation

Transport_ApplicationCommandHandler

```
void Transport_ApplicationCommandHandler (ZW_APPLICATION_TX_BUFFER * pCmd, uint8_t cmdLength, RECEIVE_OPTIONS_TYPE * rxOpt)
```

Called when the frame is received.

Parameters

Type	Direction	Argument Name	Description
ZW_APPLICATION_TX_BUFFER *	N/A	pCmd	Payload from the received frame, the command is the very first byte
uint8_t	N/A	cmdLength	Number of command bytes including the command

Type	Direction	Argument Name	Description
RECEIVE_OPTIONS_TYPE *	N/A	rxOpt	Struct to use (may be partially filled out if parsing encapsulated command)

ApplicationCommandHandler

```
void ApplicationCommandHandler (void * pSubscriberContext, SZwaveReceivePackage * pRxPackage)
```

Callback function triggered by Cmd Publisher when the frame is received from protocol See [ZAF_CmdPublisher.h](#) for more details.

Parameters

Type	Direction	Argument Name	Description
void *	N/A	pSubscriberContext	A handle to the Transport layer
SZwaveReceivePackage *	N/A	pRxPackage	Received package containing: payload from received frame, number of command bytes (including the command) and rxOpt struct

Transport_OnApplicationInitSW

```
uint8_t Transport_OnApplicationInitSW (void )
```

Initializes the ZAF transport layer.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- true if transport layer is initialized, false otherwise.

Transport_OnLearnCompleted

```
uint8_t Transport_OnLearnCompleted (node\_id\_t nodeID)
```

Register the new node id in the transport layer after learn mode completes.

Parameters

Type	Direction	Argument Name	Description
node_id_t	N/A	nodeID	The newly assigned node id.

This function must be called in LearnCompleted application function

Deprecated Use [ZAF_Transport_OnLearnCompleted\(\)](#).

Returns

- true on success.

ZAF_Transport_OnLearnCompleted

```
void ZAF_Transport_OnLearnCompleted (void )
```

Updates the active NIF depending on learn mode result.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Must be invoked when learn mode has completed (on inclusion/exclusion).

TransportCmdClassSupported

```
bool TransportCmdClassSupported (uint8_t commandClass, uint8_t command, enum SECURITY_KEY eKey)
```

Returns whether a given command is supported.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	commandClass	Command class to check for.
uint8_t	N/A	command	Command to check for.
enum SECURITY_KEY	N/A	eKey	Security key that the frame was received with.

Returns

- Returns true if the command is supported and false otherwise.

GetHighestSecureLevel

```
enum SECURITY_KEY GetHighestSecureLevel (uint8_t protocolSecBits)
```

Get highest secure level.

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	protocolSecBits	secure bits

Returns

- secure level

GetCommandClassList

```
zaf_cc_list_t * GetCommandClassList (bool included, security_key_t eKey, uint8_t endpoint)
```

Get command class list from device.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	included	boolean value for node is included
security_key_t	[in]	eKey	security key of type security_key_t
uint8_t	[in]	endpoint	0 - root 1 - X endpoint number

TYPES

TYPES

Modules

[_MULTICHAN_SOURCE_NODE_ID_](#)

[_MULTICHAN_DEST_NODE_ID_](#)

[MULTICHAN_DEST_NODE_ID_8bit](#)

[_RECEIVE_OPTIONS_TYPE_EX_](#)

[ZAF_TRANSPORT_TX_BUFFER](#)

[ccc_pair_t](#)

[agi_profile_t](#)

[transmission_result_t](#)

[_cc_handler_input_t](#)

[_cc_handler_output_t](#)

[CC_handler_map_t](#)

[CC_handler_map_v2_t](#)

[CC_handler_map_v3_t](#)

[CC_handler_map_v4_t](#)

[CC_handler_map_v5_t](#)

[zaf_cc_config_t](#)

[zaf_cc_config_entry_v1_t](#)

[zaf_cc_list_t](#)

Enumerations

```
enum    e_cmd_handler_return_code_t {
        E_CMD_HANDLER_RETURN_CODE_FAIL
        E_CMD_HANDLER_RETURN_CODE_HANDLED
        E_CMD_HANDLER_RETURN_CODE_WORKING
        E_CMD_HANDLER_RETURN_CODE_NOT_SUPPORTED
        E_CMD_HANDLER_RETURN_CODE_NO_CHANGE
    }

enum    received_frame_status_t {
        RECEIVED_FRAME_STATUS_NO_SUPPORT = 0x00
        RECEIVED_FRAME_STATUS_WORKING = 0x01
        RECEIVED_FRAME_STATUS_FAIL = 0x02
        RECEIVED_FRAME_STATUS_CANCEL = 0x03
        RECEIVED_FRAME_STATUS_CC_NOT_FOUND = 0x04
        RECEIVED_FRAME_STATUS_SUCCESS = 0xFF
    }
```

```
}
Status on incoming
frame.

enum zaf_job_status_t {
    JOB_STATUS_SUCCESS = 0
    JOB_STATUS_BUSY
    JOB_STATUS_NO_DESTINATIONS
}
Callback status used on framework API for request/response-job.

enum TRANSMISSION_RESULT_FINISH_STATUS {
    TRANSMISSION_RESULT_NOT_FINISHED
    TRANSMISSION_RESULT_FINISHED
    TRANSMISSION_RESULT_UNKNOWN
}
Indicates whether all transmissions are done.
```

Typedefs

```
typedef struct _MULTICHAN_SOURCE_NODE_ID_ MULTICHAN_SOURCE_NODE_ID

typedef struct _MULTICHAN_DEST_NODE_ID_ MULTICHAN_DEST_NODE_ID

typedef struct _RECEIVE_OPTIONS_TYPE_EX_ RECEIVE_OPTIONS_TYPE_EX
Properties of the received frame.

typedef cc_group_t cc_group_t
For backwards compatibility.

typedef cc_group_t CMD_CLASS_GRP

typedef agi_profile_t agi_profile_t
For backwards compatibility.

typedef job_status_t job_status_t
For backwards compatibility.

typedef JOB_STATUS job_status_t

typedef TRANSMISSION_RESULT transmission_result_t
For backwards compatibility.

typedef void(*) cc_handler_t(void)

typedef cc_handler_v1_t(receive_frame_status_t*)(RECEIVE_OPTIONS_TYPE_EX *, ZW_APPLICATION_TX_BUFFER *, uint8_t)

typedef cc_handler_v2_t(receive_frame_status_t*)(RECEIVE_OPTIONS_TYPE_EX *, ZW_APPLICATION_TX_BUFFER *, uint8_t, ZW_APPLICATION_TX_BUFFER *, uint8_t *)
```

typedef struct _cc_handler_input _t	cc_handler_input_t
typedef struct _cc_handler_outpu t_t	cc_handler_output_t
typedef received_frame_s tatus_t(*)	cc_handler_v3_t)(cc_handler_input_t *, cc_handler_output_t *)
typedef void(*)	basic_set_mapper_t)(ZW_APPLICATION_TX_BUFFER *p_frame) Basic Set mapper function type.
typedef void(*)	basic_get_mapper_t)(uint8_t endpoint, uint8_t *p_current_value, uint8_t *p_target_value, uint8_t *p_duration) Basic Get mapper function type.
typedef void(*)	cc_init_function_t)(void)
typedef void(*)	cc_reset_function_t)(void)
typedef void(*)	cc_migrate_function_t)(void)
typedef uint8_t(*)	lifeline_report_get_t)(cc_group_t *p_cc_pair) A lifeline report function must take an array of CC pairs as input and return the number of CC pairs being added to the array.
typedef CC_handler_map_v 5_t	CC_handler_map_latest_t Defines a type for the latest available CC handle type.
typedef void(*)	ZAF_TX_Callback_t)(transmission_result_t *pTxResult)
typedef void(*)	ZW_TX_Callback_t)(uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus) Callback function triggered after completed transmission.
typedef void(*)	ZW_TX_Multi_Callback_t)(uint8_t txStatus) Callback function triggered after ZW_SendDataMulti_Bridge completed transmission.
typedef void(*)	ZW_Void_Callback_t)(void) Generic Callback function type.
typedef zaf_cc_config_ent ry_v1_t	zaf_cc_config_entry_latest_t Eases the upgrade to future versions of the CC config struct.

Variables

const CC_handler_map_l atest_t	__start_zw_cc_handlers_v5 This is the first of the registered app handlers.
const CC_handler_map_l atest_t	__stop_zw_cc_handlers_v5 This marks the end of the handlers.

Macros

#define	HANDLER_SECTION "zw_cc_handlers_v5"

```
#define cc_handlers_start __start_zw_cc_handlers_v5

#define cc_handlers_stop __stop_zw_cc_handlers_v5

#define CC_API_HANDLER_V1 (1)
    This marks the version of the command class handlers used by ZAF_CC_Invoker.

#define CC_API_HANDLER_V2 (2)

#define CC_API_HANDLER_V3 (3)

#define REGISTER_CC_V6 (cc_api_handler_version, cc, version, handler, basic_set_mapper, basic_get_mapper,
    lifeline_report_cb, flags, init_cb, reset_cb, migrate_cb)
    Registers a given command class with version, handler, etc.

#define REGISTER_CC_V5 (cc, version, handler, basic_set_mapper, basic_get_mapper, lifeline_report_cb, flags, init_cb,
    reset_cb)
    Registers a given command class with version, handler, etc.

#define REGISTER_CC_V4 (cc, version, handler, basic_set_mapper, basic_get_mapper, lifeline_report_cb, flags, init_cb,
    reset_cb)
    Registers a given command class with version, handler, etc.

#define REGISTER_CC_V3 (cc, version, handler, basic_set_mapper, basic_get_mapper, lifeline_report_cb, flags)
    Registers a given command class with version, handler, etc.

#define REGISTER_CC_V2 (cc, version, handler)
    Registers a given command class with version, handler, etc.

#define REGISTER_CC (cc, version, handler)
    Registers a given command class with version, handler, etc.

#define ZAF_CC_CONFIG_SECTION "zw_zaf_cc_config"

#define ZAF_CC_REGISTER_CONFIG (cc, config, index)
    Registers a given command class configuration.
```

Enumeration Documentation

e_cmd_handler_return_code_t

e_cmd_handler_return_code_t	
Enumerator	
E_CMD_HANDLER_RETURN_CODE_FAIL	Not accepted or accepted but failed to execute. Command class returns FAIL.
E_CMD_HANDLER_RETURN_CODE_HANDLED	Accepted and executed by the command handler. Command class returns SUCCESS.
E_CMD_HANDLER_RETURN_CODE_WORKING	Accepted but is not fully executed. Command class returns WORKING.
E_CMD_HANDLER_RETURN_CODE_NOT_SUPPORTED	Command handler does not support this command. Command class returns NO_SUPPORT.
E_CMD_HANDLER_RETURN_CODE_NO_CHANGE	Accepted but device was already in final state, so no change was made. Command class returns SUCCESS.

received_frame_status_t

received_frame_status_t

Status on incoming frame.

Use same values as cc_supervision_status_t

Enumerator	
RECEIVED_FRAME_STATUS_NO_SUPPORT	Frame not supported.
RECEIVED_FRAME_STATUS_WORKING	Frame received successfully and timed change started.
RECEIVED_FRAME_STATUS_FAIL	Could not handle incoming frame.
RECEIVED_FRAME_STATUS_CANCEL	Don't care about status. CC or App will send the report.
RECEIVED_FRAME_STATUS_CC_NOT_FOUND	CC handler was not found in CC handler map.
RECEIVED_FRAME_STATUS_SUCCESS	Frame received successfully.

zaf_job_status_t

zaf_job_status_t

Callback status used on framework API for request/response-job.

Enumerator	
JOB_STATUS_SUCCESS	Job has been started.
JOB_STATUS_BUSY	Job couldn't start.
JOB_STATUS_NO_DESTINATIONS	Job couldn't start because there is no destinations.

TRANSMISSION_RESULT_FINISH_STATUS

TRANSMISSION_RESULT_FINISH_STATUS

Indicates whether all transmissions are done.

Used by [TRANSMISSION_RESULT](#).

Enumerator	
TRANSMISSION_RESULT_NOT_FINISHED	Still transmitting.
TRANSMISSION_RESULT_FINISHED	Done transmitting to all nodes.
TRANSMISSION_RESULT_UNKNOWN	Reserved for callbacks from the stack as the stack supplies no valid value for the finish field.

Typedef Documentation

MULTICHAN_SOURCE_NODE_ID

typedef struct _MULTICHAN_SOURCE_NODE_ID_ MULTICHAN_SOURCE_NODE_ID

MULTICHAN_DEST_NODE_ID

```
typedef struct _MULTICHAN_DEST_NODE_ID_ MULTICHAN_DEST_NODE_ID
```

RECEIVE_OPTIONS_TYPE_EX

```
typedef struct _RECEIVE_OPTIONS_TYPE_EX_ RECEIVE_OPTIONS_TYPE_EX
```

Properties of the received frame.

Used mostly by command classes to prepare response to a command.

cc_group_t

```
typedef ccc_pair_t cc_group_t
```

For backwards compatibility.

CMD_CLASS_GRP

```
typedef cc_group_t CMD_CLASS_GRP
```

AGI_PROFILE

```
typedef agi_profile_t AGI_PROFILE
```

For backwards compatibility.

job_status_t

```
typedef zaf_job_status_t job_status_t
```

For backwards compatibility.

JOB_STATUS

```
typedef job_status_t JOB_STATUS
```

TRANSMISSION_RESULT

```
typedef transmission_result_t TRANSMISSION_RESULT
```

For backwards compatibility.

cc_handler_t

```
typedef void(* cc_handler_t) (void) (void)
```

cc_handler_v1_t

```
typedef received_frame_status_t(* cc_handler_v1_t) (RECEIVE_OPTIONS_TYPE_EX *, ZW_APPLICATION_TX_BUFFER *,  
uint8_t) (RECEIVE_OPTIONS_TYPE_EX *, ZW_APPLICATION_TX_BUFFER *, uint8_t)
```

cc_handler_v2_t

```
typedef received_frame_status_t(* cc_handler_v2_t) (RECEIVE_OPTIONS_TYPE_EX *, ZW_APPLICATION_TX_BUFFER *,  
uint8_t, ZW_APPLICATION_TX_BUFFER *, uint8_t *) (RECEIVE_OPTIONS_TYPE_EX *, ZW_APPLICATION_TX_BUFFER *,  
uint8_t, ZW_APPLICATION_TX_BUFFER *, uint8_t *)
```

cc_handler_input_t

```
typedef struct _cc_handler_input_t cc_handler_input_t
```

cc_handler_output_t

```
typedef struct _cc_handler_output_t cc_handler_output_t
```

cc_handler_v3_t

```
typedef received_frame_status_t(* cc_handler_v3_t) (cc_handler_input_t *, cc_handler_output_t *) (cc_handler_input_t *,  
cc_handler_output_t *)
```

basic_set_mapper_t

```
typedef void(* basic_set_mapper_t) (ZW_APPLICATION_TX_BUFFER *p_frame) (ZW_APPLICATION_TX_BUFFER *p_frame)
```

Basic Set mapper function type.

Parameters

Type	Direction	Argument Name	Description
	[out]	p_frame	The frame that must have command class and command replaced by the command class that defines the Basic Set mapping function.

Use this type when defining a function for Basic Set mapping.

basic_get_mapper_t

```
typedef void(* basic_get_mapper_t) (uint8_t endpoint, uint8_t *p_current_value, uint8_t *p_target_value, uint8_t *p_duration) (uint8_t endpoint, uint8_t *p_current_value, uint8_t *p_target_value, uint8_t *p_duration)
```

Basic Get mapper function type.

Parameters

Type	Direction	Argument Name	Description
	[in]	endpoint	The endpoint that received the Basic Get command.
	[out]	p_current_value	Pointer to variable where the current value can be written.
	[out]	p_target_value	Pointer to variable where the target value can be written.
	[out]	p_duration	Pointer to variable where the duration can be written.

Use this type when defining a function for Basic Get mapping.

cc_init_function_t

```
typedef void(* cc_init_function_t) (void) (void)
```

cc_reset_function_t

```
typedef void(* cc_reset_function_t) (void) (void)
```

cc_migrate_function_t

```
typedef void(* cc_migrate_function_t) (void) (void)
```

lifeline_report_get_t

```
typedef uint8_t(* lifeline_report_get_t) (cc_group_t *p_cc_pair) (cc_group_t *p_cc_pair)
```

A lifeline report function must take an array of CC pairs as input and return the number of CC pairs being added to the array.

CC_handler_map_latest_t

```
typedef CC_handler_map_v5_t CC_handler_map_latest_t
```

Defines a type for the latest available CC handle type.

Users of the CC handle must use this type to avoid changes caused by a future handle version.

ZAF_TX_Callback_t

```
typedef void(* ZAF_TX_Callback_t) (transmission_result_t *pTxResult) (transmission_result_t *pTxResult)
```

ZW_TX_Callback_t

```
typedef void(* ZW_TX_Callback_t) (uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus) (uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus)
```

Callback function triggered after completed transmission.

Parameters

Type	Direction	Argument Name	Description
	N/A	txStatus	Transmit complete codes
	N/A	extendedTxStatus	Extended Tx Status

ZW_TX_Multi_Callback_t

```
typedef void(* ZW_TX_Multi_Callback_t) (uint8_t txStatus) (uint8_t txStatus)
```

Callback function triggered after ZW_SendDataMulti_Bridge completed transmission.

Parameters

Type	Direction	Argument Name	Description
	N/A	txStatus	Transmit complete codes

ZW_Void_Callback_t

```
typedef void(* ZW_Void_Callback_t) (void) (void)
```

Generic Callback function type.

zaf_cc_config_entry_latest_t

```
typedef zaf_cc_config_entry_v1_t zaf_cc_config_entry_latest_t
```

Eases the upgrade to future versions of the CC config struct.

Variable Documentation

__start_zw_cc_handlers_v5

```
const CC_handler_map_latest_t __start_zw_cc_handlers_v5
```

This is the first of the registered app handlers.

__stop_zw_cc_handlers_v5

```
const CC_handler_map_latest_t __stop_zw_cc_handlers_v5
```

This marks the end of the handlers.

The element after the last element. This means that this element is not valid.

_MULTICHAN_SOURCE_NODE_ID_

Public Attributes

<code>node_id_t</code>	<code>nodeId</code> The ID of the source node.
<code>uint8_t</code>	<code>endpoint</code> <code>uint8_t</code> 1, bit 0-6 - Can represent endpoints from 1-127 unless used as a bitmask where endpoint 1-7 is represented.
<code>uint8_t</code>	<code>res</code> <code>uint8_t</code> 1, bit 7 - Reserved

Public Attribute Documentation

nodeId

`node_id_t _MULTICHAN_SOURCE_NODE_ID_::nodeId`

The ID of the source node.

endpoint

`uint8_t _MULTICHAN_SOURCE_NODE_ID_::endpoint`

`uint8_t` 1, bit 0-6 - Can represent endpoints from 1-127 unless used as a bitmask where endpoint 1-7 is represented.

res

`uint8_t _MULTICHAN_SOURCE_NODE_ID_::res`

`uint8_t` 1, bit 7 - Reserved

_MULTICHAN_DEST_NODE_ID_

Public Attributes

<code>node_id_t</code>	<code>nodeId</code> The ID of the destination node.
<code>uint8_t</code>	<code>endpoint</code> <code>uint8_t</code> 1, bit 0-6 - Can represent endpoints from 1-127 unless used as a bitmask where endpoint 1-7 is represented.
<code>uint8_t</code>	<code>BitAddress</code> <code>uint8_t</code> 1, bit 7 - Interprets the endpoint fields as a bitmask addressing only the first 7 endpoints in destination.

Public Attribute Documentation

`nodeId`

`node_id_t _MULTICHAN_DEST_NODE_ID_::nodeId`

The ID of the destination node.

`endpoint`

`uint8_t _MULTICHAN_DEST_NODE_ID_::endpoint`

`uint8_t` 1, bit 0-6 - Can represent endpoints from 1-127 unless used as a bitmask where endpoint 1-7 is represented.

`BitAddress`

`uint8_t _MULTICHAN_DEST_NODE_ID_::BitAddress`

`uint8_t` 1, bit 7 - Interprets the endpoint fields as a bitmask addressing only the first 7 endpoints in destination.

MULTICHAN_DEST_NODE_ID_8bit

For the implementation of 16bit NodeId for LR, while keeping backward compatibility, the FRAME-structrues of Z-Wave classic where not copied onto ZAF structures but used with pointers, for this reason, this struct was created to keep those part working.

KEEP THE USE OF THIS STRUCT TO BARE MINIMUM!

Public Attributes

uint8_t	nodeId	The ID of the destination node.
uint8_t	endpoint	uint8_t 1, bit 0-6 - Can represent endpoints from 1-127 unless used as a bitmask where endpoint 1-7 is represented.
uint8_t	BitAddress	uint8_t 1, bit 7 - Interprets the endpoint fields as a bitmask addressing only the first 7 endpoints in destination.

Public Attribute Documentation

nodeId

uint8_t MULTICHAN_DEST_NODE_ID_8bit::nodeId

The ID of the destination node.

endpoint

uint8_t MULTICHAN_DEST_NODE_ID_8bit::endpoint

uint8_t 1, bit 0-6 - Can represent endpoints from 1-127 unless used as a bitmask where endpoint 1-7 is represented.

BitAddress

uint8_t MULTICHAN_DEST_NODE_ID_8bit::BitAddress

uint8_t 1, bit 7 - Interprets the endpoint fields as a bitmask addressing only the first 7 endpoints in destination.

_RECEIVE_OPTIONS_TYPE_EX_

Properties of the received frame.

Used mostly by command classes to prepare response to a command.

Public Attributes

uint8_t	rxStatus	Frame header info.
security_key_t	securityKey	
MULTICHAN_SOURCE_NODE_ID	sourceNode	
MULTICHAN_DEST_NODE_ID	destNode	
uint8_t	bSupervisionActive	true if supervision is active
uint8_t	sessionId	Current sessionId.
uint8_t	statusUpdate	Is statusUpdate enabled for current session.

Public Attribute Documentation

rxStatus

uint8_t _RECEIVE_OPTIONS_TYPE_EX::rxStatus

Frame header info.

securityKey

security_key_t _RECEIVE_OPTIONS_TYPE_EX::securityKey

sourceNode

MULTICHAN_SOURCE_NODE_ID _RECEIVE_OPTIONS_TYPE_EX::sourceNode

destNode

MULTICHAN_DEST_NODE_ID _RECEIVE_OPTIONS_TYPE_EX::destNode

bSupervisionActive

```
uint8_t _RECEIVE_OPTIONS_TYPE_EX_::bSupervisionActive
```

true if supervision is active

sessionId

```
uint8_t _RECEIVE_OPTIONS_TYPE_EX_::sessionId
```

Current sessionId.

statusUpdate

```
uint8_t _RECEIVE_OPTIONS_TYPE_EX_::statusUpdate
```

Is statusUpdate enabled for current session.

ZAF_TRANSPORT_TX_BUFFER

Allocates space for Supervision when used to declare a buffer.

Public Attributes

uint8_t	multichanCmdEncap
ZW_SUPERVISION_GET_FRAME	supervisionGet
ZW_APPLICATION_TX_BUFFER	appTxBuf

Public Attribute Documentation

multichanCmdEncap

`uint8_t ZAF_TRANSPORT_TX_BUFFER::multichanCmdEncap[4]`

supervisionGet

`ZW_SUPERVISION_GET_FRAME ZAF_TRANSPORT_TX_BUFFER::supervisionGet`

appTxBuf

`ZW_APPLICATION_TX_BUFFER ZAF_TRANSPORT_TX_BUFFER::appTxBuf`

ccc_pair_t

Can be used for pairing a command class with a specific command in the command class.

Public Attributes

uint8_t [cmdClass](#)
Command class.

uint8_t [cmd](#)
Command.

Public Attribute Documentation

cmdClass

uint8_t ccc_pair_t::cmdClass

Command class.

cmd

uint8_t ccc_pair_t::cmd

Command.

agi_profile_t

Can be used for pairing AGI profile identifiers.

Public Attributes

- uint8_t

[profile_MS](#)
AGI profile of type: ASSOCIATION_GROUP_INFO_REPORT_AGI_PROFILE_...
- uint8_t

[profile_LS](#)
AGI profile of type: ASSOCIATION_GROUP_INFO_REPORT_AGI_PROFILE_...

Public Attribute Documentation

profile_MS

uint8_t agi_profile_t::profile_MS

AGI profile of type: ASSOCIATION_GROUP_INFO_REPORT_AGI_PROFILE_...

profile_LS

uint8_t agi_profile_t::profile_LS

AGI profile of type: ASSOCIATION_GROUP_INFO_REPORT_AGI_PROFILE_...

transmission_result_t

This struct defines the values which can be parsed to a callback function upon an ended transmission regardless of the result.

Public Attributes

<code>node_id_t</code>	<code>nodeId</code>	The ID of the node to which the transmission has been done.
<code>uint8_t</code>	<code>status</code>	Status of the transmission.
<code>TRANSMISSION_RESULT_FINISH_STATUS</code>	<code>isFinished</code>	If transmission to several nodes, this flag is set if transmission for the last node has ended.

Public Attribute Documentation

nodeId

`node_id_t transmission_result_t::nodeId`

The ID of the node to which the transmission has been done.

status

`uint8_t transmission_result_t::status`

Status of the transmission.

See [ZW_transport_api.h](#).

isFinished

`TRANSMISSION_RESULT_FINISH_STATUS transmission_result_t::isFinished`

If transmission to several nodes, this flag is set if transmission for the last node has ended.

_cc_handler_input_t

Public Attributes

RECEIVE_OPTION
S_TYPE_EX * rx_options

ZW_APPLICATION
_TX_BUFFER * frame

uint8_t length

Public Attribute Documentation

rx_options

RECEIVE_OPTIONS_TYPE_EX* _cc_handler_input_t::rx_options

frame

ZW_APPLICATION_TX_BUFFER* _cc_handler_input_t::frame

length

uint8_t _cc_handler_input_t::length

_cc_handler_output_t

Public Attributes

ZW_APPLICATION_TX_BUFFER *	frame
uint8_t	length
uint8_t	duration

Public Attribute Documentation

frame

ZW_APPLICATION_TX_BUFFER* _cc_handler_output_t::frame

length

uint8_t _cc_handler_output_t::length

duration

uint8_t _cc_handler_output_t::duration

CC_handler_map_t

First generation of the CC handler item.

Public Attributes

uint16_t [CC](#)

uint16_t [version](#)

[cc_handler_t](#) [pHandler](#)

Public Attribute Documentation

CC

uint16_t CC_handler_map_t::CC

version

uint16_t CC_handler_map_t::version

pHandler

cc_handler_t CC_handler_map_t::pHandler

CC_handler_map_v2_t

Second generation of the CC handler item.

Public Attributes

uint32_t	handler_api_version
uint16_t	CC
uint16_t	version
cc_handler_t	handler

Public Attribute Documentation

handler_api_version

uint32_t CC_handler_map_v2_t::handler_api_version

CC

uint16_t CC_handler_map_v2_t::CC

version

uint16_t CC_handler_map_v2_t::version

handler

cc_handler_t CC_handler_map_v2_t::handler

CC_handler_map_v3_t

Third generation of the CC handler item.

Public Attributes

uint32_t	handler_api_version
uint16_t	CC
uint16_t	version
cc_handler_t	handler
basic_set_mapper_t	basic_set_mapper
basic_get_mapper_t	basic_get_mapper
lifeline_report_get_t	lifeline_report_get
uint32_t	flags

Public Attribute Documentation

handler_api_version

uint32_t CC_handler_map_v3_t::handler_api_version

CC

uint16_t CC_handler_map_v3_t::CC

version

uint16_t CC_handler_map_v3_t::version

handler

cc_handler_t CC_handler_map_v3_t::handler

basic_set_mapper

basic_set_mapper_t CC_handler_map_v3_t::basic_set_mapper

basic_get_mapper

basic_get_mapper_t CC_handler_map_v3_t::basic_get_mapper

lifeline_report_get

lifeline_report_get_t CC_handler_map_v3_t::lifeline_report_get

flags

uint32_t CC_handler_map_v3_t::flags

CC_handler_map_v4_t

Fourth generation of the CC handler item.

Public Attributes

uint8_t	handler_api_version
uint16_t	CC
uint8_t	version
cc_handler_t	handler
basic_set_mapper_t	basic_set_mapper
basic_get_mapper_t	basic_get_mapper
lifeline_report_get_t	lifeline_report_get
uint32_t	flags
cc_init_function_t	init
cc_reset_function_t	reset

Public Attribute Documentation

handler_api_version

uint8_t CC_handler_map_v4_t::handler_api_version

CC

uint16_t CC_handler_map_v4_t::CC

version

uint8_t CC_handler_map_v4_t::version

handler

cc_handler_t CC_handler_map_v4_t::handler

basic_set_mapper

```
basic_set_mapper_t CC_handler_map_v4_t::basic_set_mapper
```

basic_get_mapper

```
basic_get_mapper_t CC_handler_map_v4_t::basic_get_mapper
```

lifeline_report_get

```
lifeline_report_get_t CC_handler_map_v4_t::lifeline_report_get
```

flags

```
uint32_t CC_handler_map_v4_t::flags
```

init

```
cc_init_function_t CC_handler_map_v4_t::init
```

reset

```
cc_reset_function_t CC_handler_map_v4_t::reset
```

CC_handler_map_v5_t

Fifth generation of the CC handler item.

Public Attributes

uint8_t	handler_api_version
uint16_t	CC
uint8_t	version
cc_handler_t	handler
basic_set_mapper_t	basic_set_mapper
basic_get_mapper_t	basic_get_mapper
lifeline_report_get_t	lifeline_report_get
uint32_t	flags
cc_init_function_t	init
cc_reset_function_t	reset
cc_migrate_function_t	migrate

Public Attribute Documentation

handler_api_version

uint8_t CC_handler_map_v5_t::handler_api_version

CC

uint16_t CC_handler_map_v5_t::CC

version

uint8_t CC_handler_map_v5_t::version

handler

cc_handler_t CC_handler_map_v5_t::handler

basic_set_mapper

basic_set_mapper_t CC_handler_map_v5_t::basic_set_mapper

basic_get_mapper

basic_get_mapper_t CC_handler_map_v5_t::basic_get_mapper

lifeline_report_get

lifeline_report_get_t CC_handler_map_v5_t::lifeline_report_get

flags

uint32_t CC_handler_map_v5_t::flags

init

cc_init_function_t CC_handler_map_v5_t::init

reset

cc_reset_function_t CC_handler_map_v5_t::reset

migrate

cc_migrate_function_t CC_handler_map_v5_t::migrate

zaf_cc_config_t

Defines the first two elements for all CC configurations.

Public Attributes

RECEIVE_OPTION
S_TYPE_EX

rxOpt

uint8_t

endpoint

Public Attribute Documentation

rxOpt

RECEIVE_OPTIONS_TYPE_EX zaf_cc_config_t::rxOpt

endpoint

uint8_t zaf_cc_config_t::endpoint

zaf_cc_config_entry_v1_t

Public Attributes

uint16_t [command_class](#)

[zaf_cc_config_t](#) * [p_cc_configuration](#)
Pointer to first element in an array of configurations or to a single configuration element.

Public Attribute Documentation

command_class

uint16_t zaf_cc_config_entry_v1_t::command_class

p_cc_configuration

zaf_cc_config_t* zaf_cc_config_entry_v1_t::p_cc_configuration

Pointer to first element in an array of configurations or to a single configuration element.

zaf_cc_list_t

Public Attributes

uint8_t	list_size
uint8_t *	cc_list

Public Attribute Documentation

list_size

uint8_t zaf_cc_list_t::list_size

cc_list

uint8_t* zaf_cc_list_t::cc_list

PRODUCT_ID_ENUM

PRODUCT_ID_ENUM

Enumerations

```
enum \_PRODUCT\_TYPE\_ID\_ENUM\_ {  
    PRODUCT_TYPE_ID_ZWAVE_ZIP_GATEWAY = 1  
    PRODUCT_TYPE_ID_ZWAVE  
    PRODUCT_TYPE_ID_ZWAVE_PLUS  
    PRODUCT_TYPE_ID_ZWAVE_PLUS_V2  
}  
Standard enum Product type Id.  
  
enum \_PRODUCT\_PLUS\_ID\_ENUM\_ {  
    PRODUCT_ID_DoorLockKeyPad = 0x0001  
    PRODUCT_ID_SwitchOnOff = 0x0002  
    PRODUCT_ID_SensorPIR = 0x0003  
    PRODUCT_ID_SerialAPI = 0x0004  
    PRODUCT_ID_PowerStrip = 0x0005  
    PRODUCT_ID_WallController = 0x0006  
    PRODUCT_ID_LEDBulb = 0x0007  
    PRODUCT_ID_MultilevelSensor = 0x0008  
    PRODUCT_ID_KeyFob = 0x0009  
}  
Contains the product IDs for the Silabs Z-Wave 700 applications.
```

Typedefs

```
typedef enum \_PRODUCT\_TYPE\_ID\_ENUM\_ eProductTypeID  
Standard enum Product type Id.  
  
typedef enum \_PRODUCT\_PLUS\_ID\_ENUM\_ eProductPlusID  
Contains the product IDs for the Silabs Z-Wave 700 applications.
```

Enumeration Documentation

[_PRODUCT_TYPE_ID_ENUM_](#)

_PRODUCT_TYPE_ID_ENUM_	
Standard enum Product type Id.	
Enumerator	
PRODUCT_TYPE_ID_ZWAVE_ZIP_GATEWAY	
PRODUCT_TYPE_ID_ZWAVE	
PRODUCT_TYPE_ID_ZWAVE_PLUS	
PRODUCT_TYPE_ID_ZWAVE_PLUS_V2	

_PRODUCT_PLUS_ID_ENUM_

_PRODUCT_PLUS_ID_ENUM_

Contains the product IDs for the Silabs Z-Wave 700 applications.

Enumerator	
PRODUCT_ID_DoorLockKeyPad	
PRODUCT_ID_SwitchOnOff	
PRODUCT_ID_SensorPIR	
PRODUCT_ID_SerialAPI	
PRODUCT_ID_PowerStrip	
PRODUCT_ID_WallController	
PRODUCT_ID_LEDBulb	
PRODUCT_ID_MultilevelSensor	
PRODUCT_ID_KeyFob	

Typedef Documentation

eProductTypeID

typedef enum _PRODUCT_TYPE_ID_ENUM_ eProductTypeID

Standard enum Product type Id.

eProductPlusID

typedef enum _PRODUCT_PLUS_ID_ENUM_ eProductPlusID

Contains the product IDs for the Silabs Z-Wave 700 applications.

ZAF_ProtocolConfig

ZAF_ProtocolConfig

Functions

<code>SAppNodeInfo_t *</code>	<code>zaf_get_app_node_info(void)</code> Gets the application node info.
<code>SRadioConfig_t *</code>	<code>zaf_get_radio_config(void)</code> Gets the radio configuration.
<code>const SProtocolConfig_t *</code>	<code>zaf_get_protocol_config(void)</code> Gets the protocol configuration.

Function Documentation

zaf_get_app_node_info

`SAppNodeInfo_t * zaf_get_app_node_info (void )`

Gets the application node info.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- `SAppNodeInfo_t*` Pointer to the application node info

zaf_get_radio_config

`SRadioConfig_t * zaf_get_radio_config (void )`

Gets the radio configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- `SRadioConfig_t*` Pointer to the radio configuration

zaf_get_protocol_config

```
const SProtocolConfig_t * zaf_get_protocol_config (void )
```

Gets the protocol configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- const SProtocolConfig_t* Pointer to the protocol configuration

Apps

Apps

Modules

- [SerialAPI](#)
- [Zniffer](#)

SerialAPI

SerialAPI

Modules

- [cmd_handler_map_t](#)
- [_transport_t](#)
- [comm_interface_frame_ptr](#)

Enumerations

```
enum transport_type_t {  
    TRANSPORT_TYPE_UART  
    TRANSPORT_TYPE_SPI  
    TRANSPORT_TYPE_ETHERNET  
    TRANSPORT_TYPE_UART  
    TRANSPORT_TYPE_SPI  
    TRANSPORT_TYPE_ETHERNET  
}  
  
enum comm_interface_parse_result_t {  
    PARSE_IDLE  
    PARSE_FRAME_RECEIVED  
    PARSE_FRAME_SENT  
    PARSE_FRAME_ERROR  
    PARSE_RX_TIMEOUT  
    PARSE_TX_TIMEOUT  
    PARSE_IDLE  
    PARSE_FRAME_RECEIVED  
    PARSE_FRAME_SENT  
    PARSE_FRAME_ERROR  
    PARSE_RX_TIMEOUT  
    PARSE_TX_TIMEOUT  
}
```

Typedefs

```
typedef void(*) cmd_handler_t(const comm_interface_frame_ptr)  
  
typedef bool(*) cmd_foreach_callback_t(cmd_handler_map_t const *const p_cmd_entry, uint8_t *supported_cmds_mask)  
  
typedef void * transport_handle_t  
  
typedef void(*) transmit_done_cb_t(transport_handle_t transport)  
  
typedef struct transport_t  
_transport_t
```

Variables

```
comm_interface_f serial_frame  
rame_ptr const
```

Functions

bool	<code>invoke_cmd_handler</code> (const comm_interface_frame_ptr frame)	Invoke command handler.
void	<code>cmd_foreach</code> (cmd_foreach_callback_t callback, uint8_t *supported_cmds_mask)	Invokes callback for each registered command.
uint8_t	<code>frame_payload_len</code> (const comm_interface_frame_ptr frame)	
void	<code>comm_interface_transmit_frame</code> (uint8_t cmd, uint8_t type, const uint8_t *payload, uint8_t len, transmit_done_cb_t cb)	
void	<code>comm_interface_wait_transmit_done</code> (void)	
void	<code>comm_interface_init</code> (void)	
uint32_t	<code>comm_interface_get_ack_timeout_ms</code> (void)	
void	<code>comm_interface_set_ack_timeout_ms</code> (uint32_t t)	
uint32_t	<code>comm_interface_get_byte_timeout_ms</code> (void)	
void	<code>comm_interface_set_byte_timeout_ms</code> (uint32_t t)	
<code>comm_interface_parse_result_t</code>	<code>comm_interface_parse_data</code> (bool ack)	

Macros

#define	CMD_HANDLER_SECTION "zw_cmd_handlers"
#define	ZW_ADD_CMD (cmd)
#define	RECEIVE_BUFFER_SIZE 180
#define	FRAME_LENGTH_MIN 3
#define	FRAME_LENGTH_MAX RECEIVE_BUFFER_SIZE

Enumeration Documentation

transport_type_t

transport_type_t	
Enumerator	
TRANSPORT_TYPE_UART	
TRANSPORT_TYPE_SPI	
TRANSPORT_TYPE_ETHERNET	
TRANSPORT_TYPE_UART	
TRANSPORT_TYPE_SPI	
TRANSPORT_TYPE_ETHERNET	

comm_interface_parse_result_t

comm_interface_parse_result_t

Enumerator

PARSE_IDLE	
PARSE_FRAME_RECEIVED	
PARSE_FRAME_SENT	
PARSE_FRAME_ERROR	
PARSE_RX_TIMEOUT	
PARSE_TX_TIMEOUT	
PARSE_IDLE	
PARSE_FRAME_RECEIVED	
PARSE_FRAME_SENT	
PARSE_FRAME_ERROR	
PARSE_RX_TIMEOUT	
PARSE_TX_TIMEOUT	

Typedef Documentation

cmd_handler_t

typedef void(* cmd_handler_t) (const comm_interface_frame_ptr)(const comm_interface_frame_ptr)

cmd_foreach_callback_t

typedef bool(* cmd_foreach_callback_t) (cmd_handler_map_t const *const p_cmd_entry, uint8_t *supported_cmds_mask))
(cmd_handler_map_t const *const p_cmd_entry, uint8_t *supported_cmds_mask)

transport_handle_t

typedef void* transport_handle_t

transmit_done_cb_t

typedef void(* transmit_done_cb_t) (transport_handle_t transport))(transport_handle_t transport)

transport_t

typedef struct _transport_t transport_t

Variable Documentation

serial_frame

```
comm_interface_frame_ptr const serial_frame
```

Function Documentation

invoke_cmd_handler

```
bool invoke_cmd_handler (const comm_interface_frame_ptr frame)
```

Invoke command handler.

Parameters

Type	Direction	Argument Name	Description
const comm_interface_frame_ptr	[in]	frame	Frame

Returns

- true if handler for given `frame` was invoked, false if no handler was found

cmd_foreach

```
void cmd_foreach (cmd_foreach_callback_t callback, uint8_t * supported_cmds_mask)
```

Invokes callback for each registered command.

Parameters

Type	Direction	Argument Name	Description
cmd_foreach_callback_t	N/A	callback	Callback function to invoke.
uint8_t *	N/A	supported_cmds_mask	Context to pass on to the callback function.

Will stop if the callback returns true.

frame_payload_len

```
static uint8_t frame_payload_len (const comm_interface_frame_ptr frame)
```

Parameters

Type	Direction	Argument Name	Description
const comm_interface_frame_ptr	N/A	frame	

comm_interface_transmit_frame

```
void comm_interface_transmit_frame (uint8_t cmd, uint8_t type, const uint8_t * payload, uint8_t len, transmit\_done\_cb\_t cb)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	cmd	
uint8_t	N/A	type	
const uint8_t *	N/A	payload	
uint8_t	N/A	len	
transmit_done_cb_t	N/A	cb	

comm_interface_wait_transmit_done

```
void comm_interface_wait_transmit_done (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

comm_interface_init

```
void comm_interface_init (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

comm_interface_get_ack_timeout_ms

```
uint32_t comm_interface_get_ack_timeout_ms (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

comm_interface_set_ack_timeout_ms

```
void comm_interface_set_ack_timeout_ms (uint32_t t)
```

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	t	

comm_interface_get_byte_timeout_ms

```
uint32_t comm_interface_get_byte_timeout_ms (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

comm_interface_set_byte_timeout_ms

```
void comm_interface_set_byte_timeout_ms (uint32_t t)
```

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	t	

comm_interface_parse_data

```
comm_interface_parse_result_t comm_interface_parse_data (bool ack)
```

Parameters

Type	Direction	Argument Name	Description
bool	N/A	ack	

cmd_handler_map_t

Public Attributes

uint8_t	cmd
cmd_handler_t	pHandler

Public Attribute Documentation

cmd

uint8_t cmd_handler_map_t::cmd

pHandler

cmd_handler_t cmd_handler_map_t::pHandler

_transport_t

Public Attributes

transport_type_t type

transport_handle_
t handle

Public Attribute Documentation

type

transport_type_t _transport_t::type

handle

transport_handle_t _transport_t::handle

comm_interface_frame_ptr

Public Attributes

- uint8_t sof
- uint8_t len
- uint8_t type
- uint8_t cmd
- uint8_t payload

Public Attribute Documentation

sof

uint8_t comm_interface_frame_ptr::sof

len

uint8_t comm_interface_frame_ptr::len

type

uint8_t comm_interface_frame_ptr::type

cmd

uint8_t comm_interface_frame_ptr::cmd

payload

uint8_t comm_interface_frame_ptr::payload

Zniffer

Zniffer

Modules

- [_transport_t](#)
- [comm_interface_frame_ptr](#)
- [tx_cmd_frame_t](#)
- [tx_data_frame_t](#)
- [tx_beam_start_frame_t](#)
- [tx_beam_stop_frame_t](#)

Enumerations

```
enum transport_type_t {  
    TRANSPORT_TYPE_UART  
    TRANSPORT_TYPE_SPI  
    TRANSPORT_TYPE_ETHERNET  
    TRANSPORT_TYPE_UART  
    TRANSPORT_TYPE_SPI  
    TRANSPORT_TYPE_ETHERNET  
}  
  
enum comm_interface_parse_result_t {  
    PARSE_IDLE  
    PARSE_FRAME_RECEIVED  
    PARSE_FRAME_SENT  
    PARSE_FRAME_ERROR  
    PARSE_RX_TIMEOUT  
    PARSE_TX_TIMEOUT  
    PARSE_IDLE  
    PARSE_FRAME_RECEIVED  
    PARSE_FRAME_SENT  
    PARSE_FRAME_ERROR  
    PARSE_RX_TIMEOUT  
    PARSE_TX_TIMEOUT  
}
```

Typedefs

```
typedef void * transport_handle_t  
  
typedef void(* transmit_done_cb_t)(transport_handle_t transport)  
  
typedef struct _transport_t
```

Variables

```
comm_interface_f
```


rame_ptr

const

serial_frame

Functions

uint8_t

frame_payload_len(const comm_interface_frame_ptr frame)

void

comm_interface_transmit_frame(uint8_t cmd, uint8_t type, const uint8_t *payload, uint8_t len, transmit_done_cb_t cb)

void

comm_interface_wait_transmit_done(void)

void

comm_interface_init(void)

uint32_t

comm_interface_get_byte_timeout_ms(void)

void

comm_interface_set_byte_timeout_ms(uint32_t t)

comm_interface_parse_result_t

comm_interface_parse_data()

Macros

#define

SOF '#'

#define

SODF 'I'

#define

SODM 0x3

#define

SOWU 0x2

#define

CMD_FRAME 0x00

#define

DATA_FRAME 0x01

#define

BEAM_FRAME 0x02

#define

BEAM_START 0x04

#define

BEAM_STOP 0x05

#define

RECEIVE_BUFFER_SIZE 180

Enumeration Documentation

transport_type_t

transport_type_t

Enumerator

TRANSPORT_TYPE_UART	
TRANSPORT_TYPE_SPI	
TRANSPORT_TYPE_ETHERNET	
TRANSPORT_TYPE_UART	
TRANSPORT_TYPE_SPI	
TRANSPORT_TYPE_ETHERNET	

comm_interface_parse_result_t

comm_interface_parse_result_t	
Enumerator	
PARSE_IDLE	
PARSE_FRAME_RECEIVED	
PARSE_FRAME_SENT	
PARSE_FRAME_ERROR	
PARSE_RX_TIMEOUT	
PARSE_TX_TIMEOUT	
PARSE_IDLE	
PARSE_FRAME_RECEIVED	
PARSE_FRAME_SENT	
PARSE_FRAME_ERROR	
PARSE_RX_TIMEOUT	
PARSE_TX_TIMEOUT	

Typedef Documentation

transport_handle_t

typedef void* transport_handle_t

transmit_done_cb_t

typedef void(* transmit_done_cb_t) (transport_handle_t transport))(transport_handle_t transport)

transport_t

typedef struct _transport_t transport_t

Variable Documentation

serial_frame

comm_interface_frame_ptr const serial_frame

Function Documentation

frame_payload_len

```
static uint8_t frame_payload_len (const comm\_interface\_frame\_ptr frame)
```

Parameters

Type	Direction	Argument Name	Description
const comm_interface_frame_ptr	N/A	frame	

comm_interface_transmit_frame

```
void comm_interface_transmit_frame (uint8_t cmd, uint8_t type, const uint8_t * payload, uint8_t len, transmit\_done\_cb\_t cb)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	cmd	
uint8_t	N/A	type	
const uint8_t *	N/A	payload	
uint8_t	N/A	len	
transmit_done_cb_t	N/A	cb	

comm_interface_wait_transmit_done

```
void comm_interface_wait_transmit_done (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

comm_interface_init

```
void comm_interface_init (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

comm_interface_get_byte_timeout_ms

```
uint32_t comm_interface_get_byte_timeout_ms (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

comm_interface_set_byte_timeout_ms

```
void comm_interface_set_byte_timeout_ms (uint32_t t)
```

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	t	

comm_interface_parse_data

```
comm_interface_parse_result_t comm_interface_parse_data ()
```

tx_cmd_frame_t

Public Attributes

uint8_t	sof
uint8_t	cmd
uint8_t	len
uint8_t	payload

Public Attribute Documentation

sof

uint8_t tx_cmd_frame_t::sof

cmd

uint8_t tx_cmd_frame_t::cmd

len

uint8_t tx_cmd_frame_t::len

payload

uint8_t tx_cmd_frame_t::payload[UINT8_MAX]

tx_data_frame_t

Public Attributes

uint8_t	sof
uint8_t	type
uint8_t	timestamp1
uint8_t	timestamp2
uint8_t	ch_speed
uint8_t	region_no
int8_t	rssi
uint8_t	sodm
uint8_t	sod
uint8_t	len
uint8_t	payload

Public Attribute Documentation

sof

uint8_t tx_data_frame_t::sof

type

uint8_t tx_data_frame_t::type

timestamp1

uint8_t tx_data_frame_t::timestamp1

timestamp2

uint8_t tx_data_frame_t::timestamp2

ch_speed

uint8_t tx_data_frame_t::ch_speed

region_no

uint8_t tx_data_frame_t::region_no

rss

int8_t tx_data_frame_t::rss

sodm

uint8_t tx_data_frame_t::sodm

sod

uint8_t tx_data_frame_t::sod

len

uint8_t tx_data_frame_t::len

payload

uint8_t tx_data_frame_t::payload[UINT8_MAX]

tx_beam_start_frame_t

Public Attributes

uint8_t	sof
uint8_t	type
uint8_t	timestamp1
uint8_t	timestamp2
uint8_t	ch_speed
uint8_t	region_no
uint8_t	rssi
uint8_t	payload

Public Attribute Documentation

sof

uint8_t tx_beam_start_frame_t::sof

type

uint8_t tx_beam_start_frame_t::type

timestamp1

uint8_t tx_beam_start_frame_t::timestamp1

timestamp2

uint8_t tx_beam_start_frame_t::timestamp2

ch_speed

uint8_t tx_beam_start_frame_t::ch_speed

uint8_t tx_beam_start_frame_t::region_no

rss

uint8_t tx_beam_start_frame_t::rss

payload

uint8_t tx_beam_start_frame_t::payload[4]

tx_beam_stop_frame_t

Public Attributes

uint8_t	sof
uint8_t	type
uint8_t	timestamp1
uint8_t	timestamp2
uint8_t	rssi
uint16_t	counter

Public Attribute Documentation

sof

uint8_t tx_beam_stop_frame_t::sof

type

uint8_t tx_beam_stop_frame_t::type

timestamp1

uint8_t tx_beam_stop_frame_t::timestamp1

timestamp2

uint8_t tx_beam_stop_frame_t::timestamp2

rssi

uint8_t tx_beam_stop_frame_t::rssi

counter

```
uint16_t tx_beam_stop_frame_t::counter
```

Z-Wave API

Z-Wave API

Modules

[Z-Wave Serial API](#)

[ZW Learn Mode defines](#)

[Z-Wave Application Transport Interface](#)

[Z-Wave Basis API](#)

[Z-Wave Security API](#)

[Z-Wave System Startup API](#)

[Z-Wave Transport API](#)

[Z-Wave Typedefs](#)

[Z-Wave User Task API](#)

Z-Wave Serial API

Z-Wave Serial API

Modules

- Frame Types
- GET_INIT_DATA_FLAG
- FUNC_ID

Enumerations

```
enum nvm_backup_restore_addr_size_t {
    NVM_BACKUP_RESTORE_ADDR_SIZE = 2
    NVM_EXT_BACKUP_RESTORE_ADDR_SIZE = 4
}
size of the address field of NVM_BACKUP_RESTORE or NVM_EXT_BACKUP_RESTORE frames.
```

Macros

```
#define SOF 0x01
Start Of Frame.

#define ACK 0x06
Acknowledge successful frame reception.

#define NAK 0x15
Not Acknowledge successful frame reception - please retransmit...

#define CAN 0x18
Frame received (from host) was dropped - waiting for ACK.
```

Enumeration Documentation

nvm_backup_restore_addr_size_t

nvm_backup_restore_addr_size_t

size of the address field of NVM_BACKUP_RESTORE or NVM_EXT_BACKUP_RESTORE frames.

Item value is the size in byte of the address field.

Enumerator	
NVM_BACKUP_RESTORE_ADDR_SIZE	2 bytes for the legacy NVM backup & restore command
NVM_EXT_BACKUP_RESTORE_ADDR_SIZE	4 bytes for the extended NVM backup & restore command

Frame Types

Frame Types

Macros

#define REQUEST 0x00

#define RESPONSE 0x01

GET_INIT_DATA_FLAG

GET_INIT_DATA_FLAG

Flags used in FUNC_ID_SERIAL_API_GET_INIT_DATA functionality.

Macros

```
#define GET_INIT_DATA_FLAG_SLAVE_API 0x01

#define GET_INIT_DATA_FLAG_SECONDARY_CTRL 0x04

#define GET_INIT_DATA_FLAG_IS_SUC 0x08
```


FUNC_ID

FUNC_ID

Function IDs.

Macros

```
#define FUNC_ID_SERIAL_API_GET_INIT_DATA 0x02

#define FUNC_ID_SERIAL_API_APPL_NODE_INFORMATION 0x03

#define FUNC_ID_APPLICATION_COMMAND_HANDLER 0x04

#define FUNC_ID_ZW_GET_CONTROLLER_CAPABILITIES 0x05

#define FUNC_ID_SERIAL_API_SET_TIMEOUTS 0x06

#define FUNC_ID_SERIAL_API_GET_CAPABILITIES 0x07

#define FUNC_ID_SERIAL_API_SOFT_RESET 0x08

#define FUNC_ID_ZW_GET_PROTOCOL_VERSION 0x09

#define FUNC_ID_SERIAL_API_STARTED 0x0A

#define FUNC_ID_SERIAL_API_SETUP 0x0B

#define FUNC_ID_SERIAL_API_APPL_NODE_INFORMATION_CMD_CLASSES 0x0C

#define FUNC_ID_ZW_SEND_DATA_EX 0x0E

#define FUNC_ID_ZW_SEND_DATA_MULTI_EX 0x0F

#define FUNC_ID_ZW_SET_RF_RECEIVE_MODE 0x10

#define FUNC_ID_ZW_SET_SLEEP_MODE 0x11

#define FUNC_ID_ZW_SEND_NODE_INFORMATION 0x12

#define FUNC_ID_ZW_SEND_DATA 0x13

#define FUNC_ID_ZW_SEND_DATA_MULTI 0x14

#define FUNC_ID_ZW_GET_VERSION 0x15

#define FUNC_ID_ZW_SEND_DATA_ABORT 0x16

#define FUNC_ID_ZW_RF_POWER_LEVEL_SET 0x17

#define FUNC_ID_ZW_SEND_DATA_META 0x18

#define FUNC_ID_ZW_RESERVED_SD 0x19

#define FUNC_ID_ZW_RESERVED_SDM 0x1A

#define FUNC_ID_ZW_RESERVED_SRI 0x1B
```

```
#define FUNC_ID_ZW_SET_ROUTING_INFO 0x1B
#define FUNC_ID_ZW_GET_RANDOM 0x1C
#define FUNC_ID_ZW_RANDOM 0x1D
#define FUNC_ID_SET_RF_POWERLEVEL_REDISCOVERY 0x1E
#define FUNC_ID_MEMORY_GET_ID 0x20
#define FUNC_ID_MEMORY_GET_BYTE 0x21
#define FUNC_ID_MEMORY_PUT_BYTE 0x22
#define FUNC_ID_MEMORY_GET_BUFFER 0x23
#define FUNC_ID_MEMORY_PUT_BUFFER 0x24
#define FUNC_ID_SERIAL_API_GET_APPL_HOST_MEMORY_OFFSET 0x25
#define FUNC_ID_DEBUG_OUTPUT 0x26
#define FUNC_ID_AUTO_PROGRAMMING 0x27
#define FUNC_ID_NVR_GET_VALUE 0x28
#define FUNC_ID_NVM_GET_ID 0x29
#define FUNC_ID_NVM_EXT_READ_LONG_BUFFER 0x2A
#define FUNC_ID_NVM_EXT_WRITE_LONG_BUFFER 0x2B
#define FUNC_ID_NVM_EXT_READ_LONG_BYTE 0x2C
#define FUNC_ID_NVM_EXT_WRITE_LONG_BYTE 0x2D
#define FUNC_ID_NVM_BACKUP_RESTORE 0x2E
#define FUNC_ID_ZW_NVR_GET_APP_VALUE 0x2F
#define FUNC_ID_CLOCK_SET 0x30
#define FUNC_ID_CLOCK_GET 0x31
#define FUNC_ID_CLOCK_CMP 0x32
#define FUNC_ID_RTC_TIMER_CREATE 0x33
#define FUNC_ID_RTC_TIMER_READ 0x34
#define FUNC_ID_RTC_TIMER_DELETE 0x35
#define FUNC_ID_RTC_TIMER_CALL 0x36
#define FUNC_ID_CLEAR_TX_TIMERS 0x37
#define FUNC_ID_GET_TX_TIMERS 0x38
#define FUNC_ID_ZW_CLEAR_NETWORK_STATS 0x39
#define FUNC_ID_ZW_GET_NETWORK_STATS 0x3A
```

```
#define FUNC_ID_ZW_GET_BACKGROUND_RSSI 0x3B
#define FUNC_ID_ZW_SET_LISTEN_BEFORE_TALK_THRESHOLD 0x3C
#define FUNC_ID_NVM_EXT_BACKUP_RESTORE 0x3D
#define FUNC_ID_ZW_REMOVE_NODE_ID_FROM_NETWORK 0x3F
#define FUNC_ID_ZW_SET_LEARN_NODE_STATE 0x40
#define FUNC_ID_ZW_GET_NODE_PROTOCOL_INFO 0x41
#define FUNC_ID_ZW_SET_DEFAULT 0x42
#define FUNC_ID_ZW_NEW_CONTROLLER 0x43
#define FUNC_ID_ZW_REPLICATION_COMMAND_COMPLETE 0x44
#define FUNC_ID_ZW_REPLICATION_SEND_DATA 0x45
#define FUNC_ID_ZW_ASSIGN_RETURN_ROUTE 0x46
#define FUNC_ID_ZW_DELETE_RETURN_ROUTE 0x47
#define FUNC_ID_ZW_REQUEST_NODE_NEIGHBOR_UPDATE 0x48
#define FUNC_ID_ZW_APPLICATION_UPDATE 0x49
#define FUNC_ID_ZW_APPLICATION_CONTROLLER_UPDATE 0x49
#define FUNC_ID_ZW_ADD_NODE_TO_NETWORK 0x4A
#define FUNC_ID_ZW_REMOVE_NODE_FROM_NETWORK 0x4B
#define FUNC_ID_ZW_ADD_CTRL_AND_ASSIGN_PRIMARY_CTRL 0x4C
#define FUNC_ID_ZW_CONTROLLER_CHANGE 0x4D
#define FUNC_ID_ZW_RESERVED_FN 0x4E
#define FUNC_ID_ZW_ASSIGN_PRIORITY_RETURN_ROUTE 0x4F
#define FUNC_ID_ZW_SET_LEARN_MODE 0x50
#define FUNC_ID_ZW_ASSIGN_SUC_RETURN_ROUTE 0x51
#define FUNC_ID_ZW_ENABLE_SUC 0x52
#define FUNC_ID_ZW_REQUEST_NETWORK_UPDATE 0x53
#define FUNC_ID_ZW_SET_SUC_NODE_ID 0x54
#define FUNC_ID_ZW_DELETE_SUC_RETURN_ROUTE 0x55
#define FUNC_ID_ZW_GET_SUC_NODE_ID 0x56
#define FUNC_ID_ZW_SEND_SUC_ID 0x57
#define FUNC_ID_ZW_ASSIGN_PRIORITY_SUC_RETURN_ROUTE 0x58
#define FUNC_ID_ZW_REDISCOVERY_NEEDED 0x59
```

```
#define FUNC_ID_ZW_REQUEST_NODE_NEIGHBOR_UPDATE_OPTION 0x5A
#define FUNC_ID_ZW_SUPPORT9600_ONLY 0x5B
#define FUNC_ID_ZW_REQUEST_NEW_ROUTE_DESTINATIONS 0x5C
#define FUNC_ID_ZW_IS_NODE_WITHIN_DIRECT_RANGE 0x5D
#define FUNC_ID_ZW_EXPLORE_REQUEST_INCLUSION 0x5E
#define FUNC_ID_ZW_EXPLORE_REQUEST_EXCLUSION 0x5F
#define FUNC_ID_ZW_REQUEST_NODE_INFO 0x60
#define FUNC_ID_ZW_REMOVE_FAILED_NODE_ID 0x61
#define FUNC_ID_ZW_IS_FAILED_NODE_ID 0x62
#define FUNC_ID_ZW_REPLACE_FAILED_NODE 0x63
#define FUNC_ID_ZW_SET_ROUTING_MAX_6_00 0x65
#define FUNC_ID_ZW_IS_PRIMARY_CTRL 0x66
#define FUNC_ID_ZW_AES_ECB 0x67
#define FUNC_ID_ZW_REQUEST_NODETYPE_NEIGHBOR_UPDATE 0x68
#define FUNC_ID_ZW_TRANSFER_PROTOCOL_CC 0x69
#define FUNC_ID_ZW_ENABLE_NODE_NLS 0x6A
#define FUNC_ID_ZW_GET_NODE_NLS_STATE 0x6B
#define FUNC_ID_ZW_REQUEST_PROTOCOL_CC_ENCRYPTION 0x6C
#define FUNC_ID_TIMER_START 0x70
#define FUNC_ID_TIMER_RESTART 0x71
#define FUNC_ID_TIMER_CANCEL 0x72
#define FUNC_ID_TIMER_CALL 0x73
#define FUNC_ID_ZW_FIRMWARE_UPDATE_NVM 0x78
#define FUNC_ID_GET_ROUTING_TABLE_LINE 0x80
#define FUNC_ID_GET_TX_COUNTER 0x81
#define FUNC_ID_RESET_TX_COUNTER 0x82
#define FUNC_ID_STORE_NODEINFO 0x83
#define FUNC_ID_STORE_HOMEID 0x84
#define FUNC_ID_LOCK_ROUTE_RESPONSE 0x90
#define FUNC_ID_ZW_GET_PRIORITY_ROUTE 0x92
#define FUNC_ID_ZW_SET_PRIORITY_ROUTE 0x93
```

```
#define FUNC_ID_ZW_GET_LAST_WORKING_ROUTE 0x92

#define FUNC_ID_ZW_SET_LAST_WORKING_ROUTE 0x93

#define FUNC_ID_SERIAL_API_TEST 0x95

#define FUNC_ID_SERIAL_API_EXT 0x98

#define FUNC_ID_ZW_SECURITY_SETUP 0x9C

#define FUNC_ID_APPLICATION_SECURITY_EVENT 0x9D

#define FUNC_ID_SERIAL_API_APPL_SLAVE_NODE_INFORMATION 0xA0

#define FUNC_ID_APPLICATION_SLAVE_COMMAND_HANDLER 0xA1

#define FUNC_ID_ZW_SEND_SLAVE_NODE_INFORMATION 0xA2

#define FUNC_ID_ZW_SEND_SLAVE_DATA 0xA3

#define FUNC_ID_ZW_SET_SLAVE_LEARN_MODE 0xA4

#define FUNC_ID_ZW_GET_VIRTUAL_NODES 0xA5

#define FUNC_ID_ZW_IS_VIRTUAL_NODE 0xA6

#define FUNC_ID_ZW_RESERVED_SSD 0xA7

#define FUNC_ID_APPLICATION_COMMAND_HANDLER_BRIDGE 0xA8

#define FUNC_ID_ZW_SEND_DATA_BRIDGE 0xA9

#define FUNC_ID_ZW_SEND_DATA_META_BRIDGE 0xAA

#define FUNC_ID_ZW_SEND_DATA_MULTI_BRIDGE 0xAB

#define FUNC_ID_PWR_SETSTOPMODE 0xB0

#define FUNC_ID_PWR_CLK_PD 0xB1

#define FUNC_ID_PWR_CLK_PUP 0xB2

#define FUNC_ID_PWR_SELECT_CLK 0xB3

#define FUNC_ID_ZW_SET_WUT_TIMEOUT 0xB4

#define FUNC_ID_ZW_IS_WUT_KICKED 0xB5

#define FUNC_ID_ZW_WATCHDOG_ENABLE 0xB6

#define FUNC_ID_ZW_WATCHDOG_DISABLE 0xB7

#define FUNC_ID_ZW_WATCHDOG_KICK 0xB8

#define FUNC_ID_ZW_SET_EXT_INT_LEVEL 0xB9

#define FUNC_ID_ZW_INT_EXT_LEVEL_SET 0xB9

#define FUNC_ID_ZW_RF_POWER_LEVEL_GET 0xBA

#define FUNC_ID_ZW_GET_NEIGHBOR_COUNT 0xBB
```

```
#define FUNC_ID_ZW_ARE_NODES_NEIGHBOURS 0xBC

#define FUNC_ID_ZW_TYPE_LIBRARY 0xBD

#define FUNC_ID_ZW_SEND_TEST_FRAME 0xBE

#define FUNC_ID_ZW_GET_PROTOCOL_STATUS 0xBF

#define FUNC_ID_ZW_SEND_PROTOCOL_DATA 0xAC

#define FUNC_ID_ZW_SET_PROMISCUOUS_MODE 0xD0

#define FUNC_ID_PROMISCUOUS_APPLICATION_COMMAND_HANDLER 0xD1

#define FUNC_ID_ZW_WATCHDOG_START 0xD2

#define FUNC_ID_ZW_WATCHDOG_STOP 0xD3

#define FUNC_ID_ZW_SET_ROUTING_MAX 0xD4

#define FUNC_ID_ZW_GET_ROUTING_MAX 0xD5

#define FUNC_ID_PM_STAY_AWAKE 0xD7

#define FUNC_ID_PM_CANCEL 0xD8

#define FUNC_ID_ZW_NETWORK_MANAGEMENT_SET_MAX_INCLUSION_REQUEST_INTERVALS 0xD6

#define FUNC_ID_ZW_INITIATE_SHUTDOWN 0xD9

#define FUNC_ID_SERIAL_API_GET_LR_NODES 0xDA

#define FUNC_ID_GET_LR_CHANNEL 0xDB

#define FUNC_ID_SET_LR_CHANNEL 0xDC

#define FUNC_ID_ZW_SET_LR_VIRTUAL_IDS 0xDD

#define FUNC_ID_ZW_GET-NLS_NODES 0xC0

#define FUNC_ID_GET_DCDC_CONFIG 0xDE

#define FUNC_ID_SET_DCDC_CONFIG 0xDF

#define FUNC_ID_ZW_NUNIT_CMD 0xE0

#define FUNC_ID_ZW_NUNIT_INIT 0xE1

#define FUNC_ID_ZW_NUNIT_LIST 0xE2

#define FUNC_ID_ZW_NUNIT_RUN 0xE3

#define FUNC_ID_ZW_NUNIT_END 0xE4

#define FUNC_ID_ENABLE_RADIO_PT1 0xE7

#define FUNC_ID_GET_RADIO_PT1 0xE8

#define FUNC_ID_SEND_NOP 0xE9

#define FUNC_ID_SERIAL_API_POWER_MANAGEMENT 0xEE
```

```
#define FUNC_ID_SERIAL_API_READY 0xEF

#define FUNC_ID_PROPRIETARY_0 0xF0

#define FUNC_ID_PROPRIETARY_1 0xF1

#define FUNC_ID_PROPRIETARY_2 0xF2

#define FUNC_ID_PROPRIETARY_3 0xF3

#define FUNC_ID_PROPRIETARY_4 0xF4

#define FUNC_ID_PROPRIETARY_5 0xF5

#define FUNC_ID_PROPRIETARY_6 0xF6

#define FUNC_ID_PROPRIETARY_7 0xF7

#define FUNC_ID_PROPRIETARY_8 0xF8

#define FUNC_ID_PROPRIETARY_9 0xF9

#define FUNC_ID_PROPRIETARY_A 0xFA

#define FUNC_ID_PROPRIETARY_B 0xFB

#define FUNC_ID_PROPRIETARY_C 0xFC

#define FUNC_ID_PROPRIETARY_D 0xFD

#define FUNC_ID_PROPRIETARY_E 0xFE

#define FUNC_ID_UNKNOWN 0xFF
```

ZW Learn Mode defines

ZW Learn Mode defines

Mode parameters to ZW_SetLearnMode.

Macros

```
#define ZW_SET_LEARN_MODE_DISABLE 0x00
Disable Learn Mode.

#define ZW_SET_LEARN_MODE_CLASSIC 0x01
Start Classic Learn Mode.

#define ZW_SET_LEARN_MODE_NWI 0x02
Enable Network Wide Inclusion.

#define ZW_SET_LEARN_MODE_NWE 0x03
Enable Network Wide Exclusion.

#define ZW_SET_LEARN_MODE_SMARTSTART 0x04
Enable SmartStart Inclusion.
```


Z-Wave Application Transport Interface

Z-Wave Application Transport Interface

Modules

[SNetworkUpdateRequest](#)
[SNodeInfoRequest](#)
[SNodeInfo](#)
[SCommandClassList_t](#)
[SCommandClassSet_t](#)
[SCommandClassVersions](#)
[SCommandGenerateRandom](#)
[SCommandNodeInfo](#)
[SCommandClearNetworkStatistics](#)
[SCommandSetLearnMode](#)
[SCommandSetSmartStartLearnMode](#)
[SSetSucNodeId](#)
[SCommandSetRfPowerLevel](#)
[SSendSucNodeId](#)
[SCommandSetPromiscuousMode](#)
[SAssignReturnRoute](#)
[SCommandSetRfReceiveMode](#)
[SDeleteReturnRoute](#)
[SCommandGeneric8bParameter](#)
[SCommandGeneric16bParameter](#)
[SCommandAreNodesNeighbours](#)
[SCommandGetRoutingInfo](#)
[SSendSlaveNodeInformation](#)
[SCommandSetRoutingInfo](#)
[SCommandStoreNodeInfo](#)
[SCommandStoreHomeID](#)
[SCommandSetPriorityRoute](#)
[SCommandAesEcb](#)
[SCommandGetPriorityRoute](#)

SCommandFailedNodeIDCmd
SCommandPMSetPowerDownCallback
SCommandSetLBTTreshold
SCommandSetMaxInclReqInterval
SCommandNvmBackupRestore
SCommandSetSecurityKeys
SCommandEnableNodeNLS
SCommandGetNodeNLSState
SCommandSendProtocolDataCb
SCommandGetNLSNodes
SProtocolVersion
SProtocolInfo
SNetworkInfo
SLongRangeInfo
t_ExtNodeInfo
SRadioStatus
SApplicationHandles
SRadioConfig_t
SAppNodeInfo_t
SVirtualSlaveNodeInfo_t
SVirtualSlaveNodeInfoTable_t
SProtocolConfig_t
STransmitFrameConfig
STransmitProtocolFrameConfig
SExploreInclusionRequest
SExploreExclusionRequest
SIncludedNodeInfo
STest
SSendData
SSendDataEx
SSendDataBridge
SSendDataMulti
SSendDataMultiEx
SSendDataMultiBridge
SSendProtocolData

SCommandNetworkManagement
SRequestNewRouteDestinations
SCommandNetworkManagementDSK
SZWaveTransmitStatus
SZWaveGenerateRandomStatus
SZWaveNodeInfoStatus
SZWaveLearnModeStatus
SZWaveInvalidTxRequestStatus
SZWaveInvalidCommandStatus
SZWaveGeneric8bStatus
SZWaveGenericBoolStatus
SCommandRequestNodeNeighborUpdate
SCommandRequestNodeTypeNeighborUpdate
SZWaveGetRoutingInfoStatus
SCommandInitiateShutdown
SCommandSetSecurityFlags
SZWaveGetPriorityRouteStatus
SZWaveSetPriorityRouteStatus
SZWaveGetVirtualNodesStatus
SZWaveAesEcbStatus
SZWaveGetBackgroundRssiStatus
SZWaveNetworkManagementStatus
SNvmBackupRestoreStatus
SCommandGetNodeNLSStateStatus
SCommandEnableNodeNLSStatus
SZWaveGetIncludedNodes
SZWaveGetIncludedNodesLR
SZWaveGetIncludedNodesNLS
SZWaveTxPowerMaxSupported
SCommandTransferProtocolCC
SReceiveNodeUpdate
SReceiveSecurityEvent
UTransmitParameters
UCommandStatus

[UReceiveCmdPayload](#)[UCommandParameters](#)[SZwaveTransmitPackage](#)[SZwaveCommandStatusPackage](#)[SZwaveCommandPackage](#)[SReceiveSingle](#)[SReceiveMulti](#)[SProtocolTxOptions](#)[SRequestEncryption](#)[UReceiveParameters](#)[SZwaveReceivePackage](#)

Enumerations

```
enum   EProtocolType {
    EPROTOCOLTYPE_ZWAVE = 0
    EPROTOCOLTYPE_ZWAVE_AV
    EPROTOCOLTYPE_ZWAVE_FOR_IP
}

enum   ELibraryType {
    ELIBRARYTYPE_CONTROLLER_STATIC = 1
    ELIBRARYTYPE_CONTROLLER_PORTABLE = 2
    ELIBRARYTYPE_SLAVE = 3
    ELIBRARYTYPE_SLAVE_BEFORE_ENHANCED = 4
    ELIBRARYTYPE_INSTALLER = 5
    ELIBRARYTYPE_SLAVE_ROUTING = 6
    ELIBRARYTYPE_CONTROLLER = 7
    ELIBRARYTYPE_DUT = 8
    ELIBRARYTYPE_AVREMOTE = 10
    ELIBRARYTYPE_AVDEVICE = 11
}

enum   EInclusionState_t {
    EINCLUSIONSTATE_EXCLUDED = 0
    EINCLUSIONSTATE_UNSECURE_INCLUDED
    EINCLUSIONSTATE_SECURE_INCLUDED
}

enum   EListenBeforeTalkThreshold_t {
    ELISTENBEFORETALKTRESHOLD_DEFAULT = 127
}

enum   EtxPowerLevel_t {
    ETXPOWERLEVEL_DEFAULT = 127
}
```

```
enum EZwaveTransmitType {  
    EZWAVETRANSMITTYPE_STD = APPLICATION_INTERFACE_TRANSMIT_ENUM_OFFSET  
    EZWAVETRANSMITTYPE_EX  
    EZWAVETRANSMITTYPE_BRIDGE  
    EZWAVETRANSMITTYPE_MULTI  
    EZWAVETRANSMITTYPE_MULTI_EX  
    EZWAVETRANSMITTYPE_MULTI_BRIDGE  
    EZWAVETRANSMITTYPE_EXPLOREINCLUSIONREQUEST  
    EZWAVETRANSMITTYPE_EXPLOREEXCLUSIONREQUEST  
    EZWAVETRANSMITTYPE_NETWORKUPDATEREQUEST  
    EZWAVETRANSMITTYPE_NODEINFORMATION  
    EZWAVETRANSMITTYPE_NODEINFORMATIONREQUEST  
    EZWAVETRANSMITTYPE_TESTFRAME  
    EZWAVETRANSMITTYPE_SETSUCNODEID  
    EZWAVETRANSMITTYPE_SENDSUCNODEID  
    EZWAVETRANSMITTYPE_ASSIGNRETURNROUTE  
    EZWAVETRANSMITTYPE_DELETERETURNROUTE  
    EZWAVETRANSMITTYPE_REQUESTNEWROUTEDESTINATIONS  
    EZWAVETRANSMITTYPE_SEND_SLAVE_NODE_INFORMATION  
    EZWAVETRANSMITTYPE_SEND_SLAVE_DATA  
    EZWAVETRANSMITTYPE_INCLUDEDNODEINFORMATION  
    EZWAVETRANSMITTYPE_SECURE  
    EZWAVETRANSMITTYPE_NON_SECURE  
    EZWAVETRANSMITTYPE_NLS  
    NUM_EZWAVETRANSMITTYPE  
}  
Transmit Type enum used by SZwaveTransmitPackage.
```

```
enum ELearnStatus {  
    ELEARNSTATUS_ASSIGN_COMPLETE  
    ELEARNSTATUS_ASSIGN_NODEID_DONE  
    ELEARNSTATUS_ASSIGN_RANGE_INFO_UPDATE  
    ELEARNSTATUS_ASSIGN_INFO_PENDING  
    ELEARNSTATUS_ASSIGN_WAITING_FOR_FIND  
    ELEARNSTATUS_SMART_START_IN_PROGRESS  
    ELEARNSTATUS_LEARN_IN_PROGRESS  
    ELEARNSTATUS_LEARN_MODE_COMPLETED_TIMEOUT  
    ELEARNSTATUS_LEARN_MODE_COMPLETED_FAILED  
    ELEARNSTATUS_LEARN_MODE_UNSECURE_INCLUSION_DONE  
}  
Learn mode status updates.
```

```
enum EZwaveCommandType {
    EZWAVECOMMANDTYPE_GENERATE_RANDOM = APPLICATION_INTERFACE_COMMAND_ENUM_OFFSET
    EZWAVECOMMANDTYPE_NODE_INFO
    EZWAVECOMMANDTYPE_CLEAR_NETWORK_STATISTICS
    EZWAVECOMMANDTYPE_SET_LEARN_MODE
    EZWAVECOMMANDTYPE_SET_DEFAULT
    EZWAVECOMMANDTYPE_SEND_DATA_ABORT
    EZWAVECOMMANDTYPE_SET_PROMISCUOUS_MODE
    EZWAVECOMMANDTYPE_SET_RF_RECEIVE_MODE
    EZWAVECOMMANDTYPE_IS_NODE_WITHIN_DIRECT_RANGE
    EZWAVECOMMANDTYPE_GET_NEIGHBOR_COUNT
    EZWAVECOMMANDTYPE_ARE_NODES_NEIGHBOURS
    EZWAVECOMMANDTYPE_IS_FAILED_NODE_ID
    EZWAVECOMMANDTYPE_GET_ROUTING_TABLE_LINE
    EZWAVECOMMANDTYPE_SET_ROUTING_INFO
    EZWAVECOMMANDTYPE_STORE_NODE_INFO
    EZWAVECOMMANDTYPE_STORE_HOMEID
    EZWAVECOMMANDTYPE_LOCK_ROUTE_RESPONSE
    EZWAVECOMMANDTYPE_GET_PRIORITY_ROUTE
    EZWAVECOMMANDTYPE_SET_PRIORITY_ROUTE
    EZWAVECOMMANDTYPE_SET_SLAVE_LEARN_MODE
    EZWAVECOMMANDTYPE_IS_VIRTUAL_NODE
    EZWAVECOMMANDTYPE_GET_VIRTUAL_NODES
    EZWAVECOMMANDTYPE_GET_CONTROLLER_CAPABILITIES
    EZWAVECOMMANDTYPE_SET_ROUTING_MAX
    EZWAVECOMMANDTYPE_IS_PRIMARY_CTRL
    EZWAVECOMMANDTYPE_ADD_NODE_TO_NETWORK
    EZWAVECOMMANDTYPE_REMOVE_NODE_FROM_NETWORK
    EZWAVECOMMANDTYPE_AES_ECB
    EZWAVECOMMANDTYPE_GET_BACKGROUND_RSSI
    EZWAVECOMMANDTYPE_REMOVE_FAILED_NODE_ID
    EZWAVECOMMANDTYPE_REPLACE_FAILED_NODE_ID
    EZWAVECOMMANDTYPE_PM_STAY_AWAKE
    EZWAVECOMMANDTYPE_PM_CANCEL
    EZWAVECOMMANDTYPE_PM_REGISTER
    EZWAVECOMMANDTYPE_ZW_UPDATE_CTRL_NODE_INFORMATION
    EZWAVECOMMANDTYPE_ZW_SET_LBT_THRESHOLD
    EZWAVECOMMANDTYPE_ADD_NODE_DSK_TO_NETWORK
    EZWAVECOMMANDTYPE_NETWORK_LEARN_MODE_START
    EZWAVECOMMANDTYPE_CREAT_NEW_PRIMARY_CTRL
    EZWAVECOMMANDTYPE_CONTROLLER_CHANGE
    EZWAVECOMMANDTYPE_CLEAR_TX_TIMERS
    EZWAVECOMMANDTYPE_ZW_SET_MAX_INCL_REQ_INTERVALS
    EZWAVECOMMANDTYPE_NVM_BACKUP_OPEN
    EZWAVECOMMANDTYPE_NVM_BACKUP_READ
    EZWAVECOMMANDTYPE_NVM_BACKUP_WRITE
    EZWAVECOMMANDTYPE_NVM_BACKUP_CLOSE
    EZWAVECOMMANDTYPE_PM_SET_POWERDOWN_CALLBACK
    EZWAVECOMMANDTYPE_SET_SECURITY_KEYS
    EZWAVECOMMANDTYPE_SOFT_RESET
    EZWAVECOMMANDTYPE_BOOTLOADER_REBOOT
    EZWAVECOMMANDTYPE_REMOVE_NODEID_FROM_NETWORK
    EZWAVECOMMANDTYPE_ZW_GET_INCLUDED_NODES
    EZWAVECOMMANDTYPE_REQUESTNODENEIGHBORUPDATE
    EZWAVECOMMANDTYPE_ZW_INITIATE_SHUTDOWN
    EZWAVECOMMANDTYPE_ZW_GET_INCLUDED_LR_NODES
    EZWAVECOMMANDTYPE_ZW_GET_LR_CHANNEL
    EZWAVECOMMANDTYPE_ZW_SET_LR_CHANNEL
    EZWAVECOMMANDTYPE_ZW_SET_LR_VIRTUAL_IDS
    EZWAVECOMMANDTYPE_UNUSED1
    EZWAVECOMMANDTYPE_UNUSED2
    EZWAVECOMMANDTYPE_ZW_GET_PTI_CONFIG
    EZWAVECOMMANDTYPE_ZW_SET_TX_ATTENUATION
    EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_ADD_NODE
    EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_ABORT
    EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_REMOVE_NODE
    EZWAVECOMMANDTYPE_SECURE_SEND_DATA
    EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_LEARN_MODE_INCLUSION
    EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_LEARN_MODE_EXCLUSION
    EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_SET_SECURITY_FLAGS
}
```

```
EZWAVECOMMANDTYPE_ZW_GET_TX_POWER_MAX_SUPPORTED
EZWAVECOMMANDTYPE_REQUESTNODETYPE_NEIGHBORUPDATE
EZWAVECOMMANDTYPE_TRANSFER_PROTOCOL_CC
EZWAVECOMMANDTYPE_ENABLE_NODE_NLS
EZWAVECOMMANDTYPE_GET_NODE_NLS_STATE
EZWAVECOMMANDTYPE_ZW_GET_INCLUDED_NLS_NODES
EZWAVECOMMANDTYPE_SEND_PROTOCOL_DATA_CB
NUM_EZWAVECOMMANDTYPE
```

```
}
```

Z-Wave Command Types.

```
enum EZwaveCommandStatusType {
    EZWAVECOMMANDSTATUS_TX = APPLICATION_INTERFACE_STATUS_ENUM_OFFSET
    EZWAVECOMMANDSTATUS_GENERATE_RANDOM
    EZWAVECOMMANDSTATUS_NODE_INFO
    EZWAVECOMMANDSTATUS_LEARN_MODE_STATUS
    EZWAVECOMMANDSTATUS_SET_DEFAULT
    EZWAVECOMMANDSTATUS_INVALID_TX_REQUEST
    EZWAVECOMMANDSTATUS_INVALID_COMMAND
    EZWAVECOMMANDSTATUS_SET_RF_RECEIVE_MODE
    EZWAVECOMMANDSTATUS_IS_NODE_WITHIN_DIRECT_RANGE
    EZWAVECOMMANDSTATUS_GET_NEIGHBOR_COUNT
    EZWAVECOMMANDSTATUS_ARE_NODES_NEIGHBOURS
    EZWAVECOMMANDSTATUS_IS_FAILED_NODE_ID
    EZWAVECOMMANDSTATUS_GET_ROUTING_TABLE_LINE
    EZWAVECOMMANDSTATUS_SET_ROUTING_INFO
    EZWAVECOMMANDSTATUS_STORE_NODE_INFO
    EZWAVECOMMANDSTATUS_GET_PRIORITY_ROUTE
    EZWAVECOMMANDSTATUS_SET_PRIORITY_ROUTE
    EZWAVECOMMANDSTATUS_SET_SLAVE_LEARN_MODE
    EZWAVECOMMANDSTATUS_SET_SLAVE_LEARN_MODE_RESULT
    EZWAVECOMMANDSTATUS_IS_VIRTUAL_NODE
    EZWAVECOMMANDSTATUS_GET_VIRTUAL_NODES
    EZWAVECOMMANDSTATUS_GET_CONTROLLER_CAPABILITIES
    EZWAVECOMMANDSTATUS_IS_PRIMARY_CTRL
    EZWAVECOMMANDSTATUS_NETWORK_MANAGEMENT
    EZWAVECOMMANDSTATUS_GET_BACKGROUND_RSSI
    EZWAVECOMMANDSTATUS_AES_ECB
    EZWAVECOMMANDSTATUS_REMOVE_FAILED_NODE_ID
    EZWAVECOMMANDSTATUS_REPLACE_FAILED_NODE_ID
    EZWAVECOMMANDSTATUS_NETWORK_LEARN_MODE_START
    EZWAVECOMMANDSTATUS_ZW_SET_MAX_INCL_REQ_INTERVALS
    EZWAVECOMMANDSTATUS_NVM_BACKUP_RESTORE
    EZWAVECOMMANDSTATUS_PM_SET_POWERDOWN_CALLBACK
    EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_NODES
    EZWAVECOMMANDSTATUS_ZW_REQUESTNODENEIGHBORUPDATE
    EZWAVECOMMANDSTATUS_ZW_REQUESTNODETYPE_NEIGHBORUPDATE
    EZWAVECOMMANDSTATUS_ZW_INITIATE_SHUTDOWN
    EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_LR_NODES
    EZWAVECOMMANDSTATUS_ZW_GET_LR_CHANNEL
    EZWAVECOMMANDSTATUS_UNUSED1
    EZWAVECOMMANDSTATUS_UNUSED2
    EZWAVECOMMANDSTATUS_ZW_GET_PTI_CONFIG
    EZWAVECOMMANDSTATUS_ZW_SET_TX_ATTENUATION
    EZWAVECOMMANDSTATUS_SECURE_ON_NODE_ADDED
    EZWAVECOMMANDSTATUS_SECURE_ON_NODE_DELETED
    EZWAVECOMMANDSTATUS_SECURE_ON_NEW_NETWORK_ENTERED
    EZWAVECOMMANDSTATUS_SECURE_ON_NETWORK_MANAGEMENT_STATE_UPDATE
    EZWAVECOMMANDSTATUS_SECURE_ON_NEW_SUC
    EZWAVECOMMANDSTATUS_SECURE_ON_FRAME_TRANSMISSION
    EZWAVECOMMANDSTATUS_SECURE_ON_RX_FRAME_RECEIVED_INDICATOR
    EZWAVECOMMANDSTATUS_ZW_SET_LR_CHANNEL
    EZWAVECOMMANDSTATUS_ZW_GET_TX_POWER_MAX_SUPPORTED
    EZWAVECOMMANDSTATUS_GET_NODE_NLS_STATE
    EZWAVECOMMANDSTATUS_ENABLE_NODE_NLS
    EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_NLS_NODES
    NUM_EZWAVECOMMANDSTATUS
    EZWAVECOMMANDSTATUS_INVALID = 0xFF
}
```

Z-Wave Command Status Type used by [SZwaveCommandStatusPackage](#).


```
enum EZwaveReceiveType {
    EZWAVERECEIVETYPE_SINGLE = APPLICATION_INTERFACE_RECEIVE_ENUM_OFFSET
    EZWAVERECEIVETYPE_MULTI
    EZWAVERECEIVETYPE_NODE_UPDATE
    EZWAVERECEIVETYPE_SECURITY_EVENT
    EZWAVERECEIVETYPE_STAY_AWAKE
    EZWAVERECEIVETYPE_SECURE_FRAME_RECEIVED
    EZWAVERECEIVETYPE_REQUEST_ENCRYPTION_FRAME
    NUM_EZWAVERECEIVETYPE
    EZWAVERECEIVETYPE_INVALID = 0xFF
}
Z-Wave Receive type.

enum EProtocolEvent {
    EPROTOCOLEVENT_RFRXBEAM = 0
    EPROTOCOLEVENT_RFTXCOMPLETE
    EPROTOCOLEVENT_RFRX
    EPROTOCOLEVENT_RFTXBEAM
    EPROTOCOLEVENT_RFTXFAILBT
    EPROTOCOLEVENT_RFTXFAIL
    EPROTOCOLEVENT_RFRXABORT
    EPROTOCOLEVENT_TIMER
    EPROTOCOLEVENT_EXPLORE
    EPROTOCOLEVENT_APP_TX
    EPROTOCOLEVENT_APP_COMMAND
    EPROTOCOLEVENT_NETWORKID_UPDATE
    EPROTOCOLEVENT_CHANGE_RADIO_PHY
    EPROTOCOLEVENT_RADIO_ASSERT
    EPROTOCOLEVENT_RADIO_TX_TIMEOUT
    EPROTOCOLEVENT_RADIO_CALIBRATE
    NUM_EPROTOCOLEVENT
}

enum ERequestEncryptionEvent {
    ERPCCEEVENT_SERIALAPI_FAIL = NUM_EPROTOCOLEVENT
    ERPCCEEVENT_SERIALAPI_OK
}

enum ELearnMode {
    ELearnMode_DISABLED = 0
    ELearnMode_CLASSIC = 1
    ELearnMode_NETWORK_WIDE_INCLUSION = 2
    ELearnMode_NETWORK_WIDE_EXCLUSION = 3
}
```

Typedefs

```
typedef enum EProtocolType
EProtocolType

typedef enum ELibraryType
ELibraryType

typedef enum EInclusionState_t
EInclusionState_t

typedef enum EListenBeforeTalkThreshold_t
EListenBeforeTalkThreshold_t
```

typedef enum EtxPowerLevel_t	EtxPowerLevel_t
typedef enum EZwaveTransmitType	EZwaveTransmitType Transmit Type enum used by SZwaveTransmitPackage .
typedef enum ELearnStatus	ELearnStatus Learn mode status updates.
typedef enum EZwaveCommandType	EZwaveCommandType Z-Wave Command Types.
typedef enum EZwaveCommandStatusType	EZwaveCommandStatusType Z-Wave Command Status Type used by SZwaveCommandStatusPackage .
typedef enum EZwaveReceiveType	EZwaveReceiveType Z-Wave Receive type.
typedef enum EProtocolEvent	EProtocolEvent
typedef enum ERequestEncryptionEvent	ERequestEncryptionEvent
typedef struct SNetworkUpdateRequest	SNetworkUpdateRequest Network Update Request.
typedef enum ELearnMode	ELearnMode
typedef struct SNodeInfoRequest	SNodeInfoRequest
typedef struct SNodeInfo	SNodeInfo
typedef struct SCommandClassList_t	SCommandClassList_t
typedef struct SCommandClassSet_t	SCommandClassSet_t List of Command Classes supported when device is not included, included insecurely or included securely.
typedef struct SCommandClassVersions	SCommandClassVersions
typedef struct SCommandGenerateRandom	SCommandGenerateRandom Generates true random word.
typedef struct SCommandNodeInfo	SCommandNodeInfo

typedef struct SCommandClearNetworkStatistics	SCommandClearNetworkStatistics
typedef struct SCommandSetLearnMode	SCommandSetLearnMode
typedef struct SCommandSetSmartStartLearnMode	SCommandSetSmartStartLearnMode
typedef struct SSetSucNodeId	SSetSucNodeId Controller API.
typedef struct SCommandSetRfPowerLevel	SCommandSetRfPowerLevel
typedef struct SSendSucNodeId	SSendSucNodeId
typedef struct SCommandSetPromiscuousMode	SCommandSetPromiscuousMode
typedef struct SAssignReturnRoute	SAssignReturnRoute
typedef struct SCommandSetRfReceiveMode	SCommandSetRfReceiveMode
typedef struct SDeleteReturnRoute	SDeleteReturnRoute
typedef struct SCommandGeneric8bParameter	SCommandGeneric8bParameter
typedef struct SCommandGeneric16bParameter	SCommandGeneric16bParameter
typedef struct SCommandAreNodesNeighbours	SCommandAreNodesNeighbours
typedef struct SCommandGetRoutingInfo	SCommandGetRoutingInfo
typedef struct SSendSlaveNodeInformation	SSendSlaveNodeInformation
typedef struct SCommandSetRoutingInfo	SCommandSetRoutingInfo

typedef struct SCommandStore NodeInfo	SCommandStoreNodeInfo
typedef struct SCommandStore HomeID	SCommandStoreHomeID
typedef struct SCommandSetPri orityRoute	SCommandSetPriorityRoute
typedef struct SCommandAesEc b	SCommandAesEcb
typedef struct SCommandGetPri orityRoute	SCommandGetPriorityRoute
typedef struct SCommandFailed NodeIDCmd	SCommandFailedNodeIDCmd
typedef struct SCommandPMSet PowerDownCallb ack	SCommandPMSetPowerDownCallback
typedef struct SCommandSetLB TThreshold	SCommandSetLBTThreshold
typedef struct SCommandSetMa xInclReqInterval	SCommandSetMaxInclReqInterval
typedef struct SCommandNvmB ackupRestore	SCommandNvmBackupRestore
typedef struct SCommandSetSe curityKeys	SCommandSetSecurityKeys
typedef struct SCommandEnable NodeNLS	SCommandEnableNodeNLS
typedef struct SCommandGetNo deNLSState	SCommandGetNodeNLSState
typedef struct SCommandSendP rotocolDataCb	SCommandSendProtocolDataCb
typedef struct SCommandGetNL SNodes	SCommandGetNLSNodes
typedef struct SProtocolVersion	SProtocolVersion

typedef struct SProtocolInfo	SProtocolInfo Protocol Info data.
typedef struct SNetworkInfo	SNetworkInfo Network Info.
typedef struct SLongRangeInfo	SLongRangeInfo
typedef struct t_ExtNodeInfo	t_ExtNodeInfo
typedef struct SRadioStatus	SRadioStatus
typedef struct SApplicationHandles	SApplicationHandles
typedef struct SRadioConfig_t	SRadioConfig_t
typedef struct SAppNodeInfo_t	SAppNodeInfo_t
typedef struct SVirtualSlaveNodeInfo_t	SVirtualSlaveNodeInfo_t
typedef struct SVirtualSlaveNodeInfoTable_t	SVirtualSlaveNodeInfoTable_t
typedef struct SProtocolConfig_t	SProtocolConfig_t This allows "nulling" a pointer while modifying a virtual slave node info Or changing pointer to point to a different one.
typedef struct STransmitFrameConfig	STransmitFrameConfig
typedef struct STransmitProtocolFrameConfig	STransmitProtocolFrameConfig
typedef struct SExploreInclusionRequest	SExploreInclusionRequest
typedef struct SExploreExclusionRequest	SExploreExclusionRequest
typedef struct SIncludedNodeInfo	SIncludedNodeInfo Contains info related to sending an INIF.
typedef struct STest	STest
typedef struct SSendData	SSendData
typedef struct SSendDataEx	SSendDataEx

typedef struct SSendDataBridge	SSendDataBridge
typedef struct SSendDataMulti	SSendDataMulti
typedef struct SSendDataMultiEx x	SSendDataMultiEx
typedef struct SSendDataMultiB ridge	SSendDataMultiBridge
typedef struct SSendProtocolDa ta	SSendProtocolData
typedef struct SCommandNetwo rkManagement	SCommandNetworkManagement
typedef struct SRequestNewRo uteDestinations	SRequestNewRouteDestinations
typedef struct SCommandNetwo rkManagementDS K	SCommandNetworkManagementDSK
typedef struct SZWaveTransmitS tatus	SZWaveTransmitStatus
typedef struct SZWaveGenerate RandomStatus	SZWaveGenerateRandomStatus
typedef struct SZWaveNodeInfo Status	SZWaveNodeInfoStatus
typedef struct SZWaveLearnMo deStatus	SZWaveLearnModeStatus
typedef struct SZWaveInvalidTxR equestStatus	SZWaveInvalidTxRequestStatus
typedef struct SZWaveInvalidCo mmandStatus	SZWaveInvalidCommandStatus
typedef struct SZWaveGeneric8 bStatus	SZWaveGeneric8bStatus
typedef struct SZWaveGenericB oolStatus	SZWaveGenericBoolStatus

typedef struct SCommandRequestNodeNeighborUpdate	SCommandRequestNodeNeighborUpdate
typedef struct SCommandRequestNodeTypeNeighborUpdate	SCommandRequestNodeTypeNeighborUpdate
typedef struct SZWaveGetRoutingInfoStatus	SZWaveGetRoutingInfoStatus
typedef struct SCommandInitiateShutdown	SCommandInitiateShutdown
typedef struct SCommandSetSecurityFlags	SCommandSetSecurityFlags
typedef struct SZWaveGetPriorityRouteStatus	SZWaveGetPriorityRouteStatus
typedef struct SZWaveSetPriorityRouteStatus	SZWaveSetPriorityRouteStatus
typedef struct SZWaveGetVirtualNodesStatus	SZWaveGetVirtualNodesStatus
typedef struct SZWaveAesEcbStatus	SZWaveAesEcbStatus
typedef struct SZWaveGetBackgroundRssiStatus	SZWaveGetBackgroundRssiStatus
typedef struct SZWaveNetworkManagementStatus	SZWaveNetworkManagementStatus
typedef struct SNvmBackupRestoreStatus	SNvmBackupRestoreStatus
typedef struct SCommandGetNodeNLSStateStatus	SCommandGetNodeNLSStateStatus
typedef struct SCommandEnableNodeNLSStatus	SCommandEnableNodeNLSStatus
typedef struct SZWaveGetIncludedNodes	SZWaveGetIncludedNodes

typedef struct SZWaveGetInclud edNodesLR	SZWaveGetIncludedNodesLR	
typedef struct SZWaveGetInclud edNodesNLS	SZWaveGetIncludedNodesNLS	
typedef struct SZWaveTxPower MaxSupported	SZWaveTxPowerMaxSupported	
typedef struct SCommandTransf erProtocolCC	SCommandTransferProtocolCC	
typedef struct SReceiveNodeUp date	SReceiveNodeUpdate	The Z Wave protocol MAY notify an application by sending SReceiveNodeUpdate when a Node Information Frame has been received.
typedef struct SReceiveSecurity Event	SReceiveSecurityEvent	
typedef union UTransmitParamet ers	UTransmitParameters	Transmit Parameters Union used by SZwaveTransmitPackage .
typedef union UCommandStatus	UCommandStatus	Command Status Union used by SZwaveCommandStatusPackage .
typedef union UReceiveCmdPay load	UReceiveCmdPayload	
typedef union UCommandParam eters	UCommandParameters	
typedef struct SZwaveTransmitP ackage	SZwaveTransmitPackage	
typedef struct SZwaveCommand StatusPackage	SZwaveCommandStatusPackage	Z-Wave Command Status Package.
typedef struct SZwaveCommand Package	SZwaveCommandPackage	
typedef struct SReceiveSingle	SReceiveSingle	
typedef struct SReceiveMulti	SReceiveMulti	
typedef struct SProtocolTxOptio ns	SProtocolTxOptions	
typedef struct SRequestEncrypti on	SRequestEncryption	

typedef union

UReceiveParameters

UReceiveParameters

typedef struct

SZwaveReceivePackage

SZwaveReceivePackage

Macros

- #define ZW_MAX_PAYLOAD_SIZE 160
- Max theoretical Z-Wave frame payload size in a Z-Wave protocol using 3CH network The real Z-Wave frame payload type depends on various parameters (routed, multicast, explore, security and/or number of RF channels) Customer must not use this value in their application.
- #define MULTICAST_NODE_LIST_SIZE 64
- Numbers of nodes we can support when sending multicast frames on LR channel.
- #define TX_BUFFER_SIZE 170
- #define MAX_NODE_INFO_LENGTH 159
- Maximum size for the node info frame contained in the [SReceiveNodeUpdate](#) struct.
- #define PROTOCOL_METADATA_SPEED_IDX 0
- #define PROTOCOL_METADATA_TX_POWER_IDX 1
- #define PROTOCOL_METADATA_TX_OPTIONS_IDX 2
- #define PROTOCOL_METADATA_CALLBACK_ID_IDX 6
- #define PROTOCOL_METADATA_LENGTH (sizeof(uint8_t) + sizeof(uint8_t) + sizeof(TxOptions_t) + sizeof(uint8_t))
- #define APPLICATION_INTERFACE_TRANSMIT_ENUM_OFFSET (0x00)
- #define APPLICATION_INTERFACE_COMMAND_ENUM_OFFSET (0x40)
- #define APPLICATION_INTERFACE_RECEIVE_ENUM_OFFSET (0x80)
- #define APPLICATION_INTERFACE_STATUS_ENUM_OFFSET (0xC0)

Enumeration Documentation

EProtocolType

EProtocolType	
Enumerator	
EPROTOCOLTYPE_ZWAVE	
EPROTOCOLTYPE_ZWAVE_AV	
EPROTOCOLTYPE_ZWAVE_FOR_IP	

ELibraryType

ELibraryType

Enumerator

ELIBRARYTYPE_CONTROLLER_STATIC	
ELIBRARYTYPE_CONTROLLER_PORTABLE	DEPRECATED.
ELIBRARYTYPE_SLAVE	DEPRECATED.
ELIBRARYTYPE_SLAVE_BEFORE_ENHANCED	Previously Slave Enhanced.
ELIBRARYTYPE_INSTALLER	DEPRECATED.
ELIBRARYTYPE_SLAVE_ROUTING	DEPRECATED.
ELIBRARYTYPE_CONTROLLER	DEPRECATED.
ELIBRARYTYPE_DUT	Previously Controller Bridge.
ELIBRARYTYPE_AVREMOTE	DEPRECATED.
ELIBRARYTYPE_AVDEVICE	DEPRECATED.

InclusionState_t

InclusionState_t

Enumerator

EINCLUSIONSTATE_EXCLUDED	
EINCLUSIONSTATE_UNSECURE_INCLUDED	
EINCLUSIONSTATE_SECURE_INCLUDED	

EListenBeforeTalkThreshold_t

EListenBeforeTalkThreshold_t

Enumerator

ELISTENBEFORETALKTRESHOLD_DEFAULT	

EtXPowLevel_t

EtXPowLevel_t

Enumerator

ETXPOWERLEVEL_DEFAULT	

EZwaveTransmitType

EZwaveTransmitType

Transmit Type enum used by [SZwaveTransmitPackage](#).

Enumerator

EZWAVETRANSMITTYPE_STD	EZWAVETRANSMITTYPE_STD.
EZWAVETRANSMITTYPE_EX	EZWAVETRANSMITTYPE_EX.

EZWAVETRANSMITTYPE_BRIDGE	EZWAVETRANSMITTYPE_BRIDGE.
EZWAVETRANSMITTYPE_MULTI	Multi types requires SZwaveTransmitPackage.NodeMask to be setup.
EZWAVETRANSMITTYPE_MULTI_EX	EZWAVETRANSMITTYPE_MULTI_EX.
EZWAVETRANSMITTYPE_MULTI_BRIDGE	EZWAVETRANSMITTYPE_MULTI_BRIDGE.
EZWAVETRANSMITTYPE_EXPLOREINCLUSIONREQUEST	EZWAVETRANSMITTYPE_EXPLOREINCLUSIONREQUEST.
EZWAVETRANSMITTYPE_EXPLOREEXCLUSIONREQUEST	EZWAVETRANSMITTYPE_EXPLOREEXCLUSIONREQUEST.
EZWAVETRANSMITTYPE_NETWORKUPDATEREQUEST	EZWAVETRANSMITTYPE_NETWORKUPDATEREQUEST.
EZWAVETRANSMITTYPE_NODEINFORMATION	EZWAVETRANSMITTYPE_NODEINFORMATION.
EZWAVETRANSMITTYPE_NODEINFORMATIONREQUEST	EZWAVETRANSMITTYPE_NODEINFORMATIONREQUEST.
EZWAVETRANSMITTYPE_TESTFRAME	EZWAVETRANSMITTYPE_TESTFRAME.
EZWAVETRANSMITTYPE_SETSUCNODEID	EZWAVETRANSMITTYPE_SETSUCNODEID.
EZWAVETRANSMITTYPE_SENDSUCNODEID	EZWAVETRANSMITTYPE_SENDSUCNODEID.
EZWAVETRANSMITTYPE_ASSIGNRETURNROUTE	EZWAVETRANSMITTYPE_ASSIGNRETURNROUTE.
EZWAVETRANSMITTYPE_DELETEReturnROUTE	EZWAVETRANSMITTYPE_DELETEReturnROUTE.
EZWAVETRANSMITTYPE_REQUESTNEWROUTEDESTINATIONS	EZWAVETRANSMITTYPE_REQUESTNEWROUTEDESTINATIONS.
EZWAVETRANSMITTYPE_SEND_SLAVE_NODE_INFORMATION	EZWAVETRANSMITTYPE_SEND_SLAVE_NODE_INFORMATION.
EZWAVETRANSMITTYPE_SEND_SLAVE_DATA	EZWAVETRANSMITTYPE_SEND_SLAVE_DATA.
EZWAVETRANSMITTYPE_INCLUDEDNODEINFORMATION	EZWAVETRANSMITTYPE_INCLUDEDNODEINFORMATION.
EZWAVETRANSMITTYPE_SECURE	EZWAVETRANSMITTYPE_SECURE.
EZWAVETRANSMITTYPE_NON_SECURE	EZWAVETRANSMITTYPE_NON_SECURE.
EZWAVETRANSMITTYPE_NLS	EZWAVETRANSMITTYPE_NLS.
NUM_EZWAVETRANSMITTYPE	NUM_EZWAVETRANSMITTYPE.

ELearnStatus

ELearnStatus

Learn mode status updates.

Enumerator

ELEARNSTATUS_ASSIGN_COMPLETE	Internal status.
ELEARNSTATUS_ASSIGN_NODEID_DONE	Internal status.
ELEARNSTATUS_ASSIGN_RANGE_INFO_UPDATE	Internal status.
ELEARNSTATUS_ASSIGN_INFO_PENDING	Internal status.
ELEARNSTATUS_ASSIGN_WAITING_FOR_FIND	Internal status.
ELEARNSTATUS_SMART_START_IN_PROGRESS	Passed to application when Smart Start learn mode goes into progress.
ELEARNSTATUS_LEARN_IN_PROGRESS	Passed to application when classic learn mode goes into progress.
ELEARNSTATUS_LEARN_MODE_COMPLETED_TIMEOUT	Passed to application if classic learn mode times out.
ELEARNSTATUS_LEARN_MODE_COMPLETED_FAILED	Passed to application if learn mode failed.
ELEARNSTATUS_LEARN_MODE_UNSECURE_INCLUSION_DONE	Passed to application when unsecure inclusion is done.

EZwaveCommandType

EZwaveCommandType

Z-Wave Command Types.

Enumerator

EZWAVECOMMANDTYPE_GENERATE_RANDOM	Generate a specific number of random bytes.
EZWAVECOMMANDTYPE_NODE_INFO	Returns the Node Information Frame without command classes from the NVM for a given node ID.
EZWAVECOMMANDTYPE_CLEAR_NETWORK_STATISTICS	Clear the current network statistics collected by the Z-Wave protocol.
EZWAVECOMMANDTYPE_SET_LEARN_MODE	Enable/disable home/node ID learn mode.
EZWAVECOMMANDTYPE_SET_DEFAULT	Set the device back to the factory default state.
EZWAVECOMMANDTYPE_SEND_DATA_ABORT	Stop any application initiated transmits.
EZWAVECOMMANDTYPE_SET_PROMISCUOUS_MODE	Enable/disable the installer library promiscuous mode.
EZWAVECOMMANDTYPE_SET_RF_RECEIVE_MODE	Initialize the Z-Wave RF chip.
EZWAVECOMMANDTYPE_IS_NODE_WITHIN_DIRECT_RANGE	
EZWAVECOMMANDTYPE_GET_NEIGHBOR_COUNT	
EZWAVECOMMANDTYPE_ARE_NODES_NEIGHBOURS	
EZWAVECOMMANDTYPE_IS_FAILED_NODE_ID	Test if a node ID is stored in the failed node ID list.
EZWAVECOMMANDTYPE_GET_ROUTING_TABLE_LINE	Read out neighbor information.
EZWAVECOMMANDTYPE_SET_ROUTING_INFO	
EZWAVECOMMANDTYPE_STORE_NODE_INFO	
EZWAVECOMMANDTYPE_STORE_HOMEID	
EZWAVECOMMANDTYPE_LOCK_ROUTE_RESPONSE	If true then any attempt to purge a LastWorkingRoute entry is denied.
EZWAVECOMMANDTYPE_GET_PRIORITY_ROUTE	Get the route with the highest priority.
EZWAVECOMMANDTYPE_SET_PRIORITY_ROUTE	Set the Priority Route for a destination node.
EZWAVECOMMANDTYPE_SET_SLAVE_LEARN_MODE	Enable/disable home/node ID learn mode for virtual nodes.
EZWAVECOMMANDTYPE_IS_VIRTUAL_NODE	Check if node is a Virtual Slave node.
EZWAVECOMMANDTYPE_GET_VIRTUAL_NODES	Read the virtual node flags for all nodes in the network.
EZWAVECOMMANDTYPE_GET_CONTROLLER_CAPABILITIES	Returns the Controller capabilities.
EZWAVECOMMANDTYPE_SET_ROUTING_MAX	Set the maximum number of route tries which should be done before failing or resorting to exploring if this is specified.
EZWAVECOMMANDTYPE_IS_PRIMARY_CTRL	Returns whether the controller is a primary controller.
EZWAVECOMMANDTYPE_ADD_NODE_TO_NETWORK	Add any type of node to the network.
EZWAVECOMMANDTYPE_REMOVE_NODE_FROM_NETWORK	
EZWAVECOMMANDTYPE_AES_ECB	
EZWAVECOMMANDTYPE_GET_BACKGROUND_RSSI	Returns the most recent background RSSI levels detected.
EZWAVECOMMANDTYPE_REMOVE_FAILED_NODE_ID	Remove a node from the failed node list.
EZWAVECOMMANDTYPE_REPLACE_FAILED_NODE_ID	Replace a node from the failed node list.
EZWAVECOMMANDTYPE_PM_STAY_AWAKE	

EZWAVECOMMANDTYPE_PM_CANCEL	
EZWAVECOMMANDTYPE_PM_REGISTER	
EZWAVECOMMANDTYPE_ZW_UPDATE_CTRL_NODE_INFORMATION	
EZWAVECOMMANDTYPE_ZW_SET_LBT_THRESHOLD	Set the LBT RSSI level.
EZWAVECOMMANDTYPE_ADD_NODE_DSK_TO_NETWORK	If NetworkManagement.pHandle is not NULL
EZWAVECOMMANDTYPE_NETWORK_LEARN_MODE_START	Enable/disable the Network Management module inclusion/exclusion mode sequence according to specified mode.
EZWAVECOMMANDTYPE_CREAT_NEW_PRIMARY_CTRL	
EZWAVECOMMANDTYPE_CONTROLLER_CHANGE	Add a controller to the Z-Wave network and transfer the role as primary controller to it.
EZWAVECOMMANDTYPE_CLEAR_TX_TIMERS	Clear current Channel Transmit timers.
EZWAVECOMMANDTYPE_ZW_SET_MAX_INCL_REQ_INTERVALS	Set the maximum interval between SmartStart inclusion requests.
EZWAVECOMMANDTYPE_NVM_BACKUP_OPEN	
EZWAVECOMMANDTYPE_NVM_BACKUP_READ	
EZWAVECOMMANDTYPE_NVM_BACKUP_WRITE	
EZWAVECOMMANDTYPE_NVM_BACKUP_CLOSE	
EZWAVECOMMANDTYPE_PM_SET_POWERDOWN_CALLBACK	Register application functions to be called just before power down.
EZWAVECOMMANDTYPE_SET_SECURITY_KEYS	Initiate the S2 inclusion engine.
EZWAVECOMMANDTYPE_SOFT_RESET	Perform a soft reset.
EZWAVECOMMANDTYPE_BOOTLOADER_REBOOT	
EZWAVECOMMANDTYPE_REMOVE_NODEID_FROM_NETWORK	Remove a specific node from a Z-Wave network.
EZWAVECOMMANDTYPE_ZW_GET_INCLUDED_NODES	Get the node IDs for the nodes included in the network.
EZWAVECOMMANDTYPE_REQUESTNODENEIGHBORUPDATE	Start neighbor discovery for node if primary and other nodes are present.
EZWAVECOMMANDTYPE_ZW_INITIATE_SHUTDOWN	Initiate graceful shutdown.
EZWAVECOMMANDTYPE_ZW_GET_INCLUDED_LR_NODES	Get the node IDs for the long range nodes included in the network.
EZWAVECOMMANDTYPE_ZW_GET_LR_CHANNEL	Get the current Primary Long Range Channel.
EZWAVECOMMANDTYPE_ZW_SET_LR_CHANNEL	Set the Primary Long Range Channel.
EZWAVECOMMANDTYPE_ZW_SET_LR_VIRTUAL_IDS	Enable Long Range virtual node IDs.
EZWAVECOMMANDTYPE_UNUSED1	Legacy command.
EZWAVECOMMANDTYPE_UNUSED2	
EZWAVECOMMANDTYPE_ZW_GET_PTI_CONFIG	Check if the PTI Ziffer functionality is enabled/disabled.
EZWAVECOMMANDTYPE_ZW_SET_TX_ATTENUATION	Reduce Tx power of classic non-listening devices.
EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_ADD_NODE	Add a new node to the network.
EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_ABORT	Abort Inclusion or Exclusion of node.
EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_REMOVE_NODE	Remove a node from the network.

EZWAVECOMMANDTYPE_SECURE_SEND_DATA	Unused.
EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_LEARN_MODE_INCLUSION	Put the application in Network Wide Inclusion mode so it can be included in a network.
EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_LEARN_MODE_EXCLUSION	Put the application in Network Wide Exclusion mode so it can be excluded from a network.
EZWAVECOMMANDTYPE_SECURE_NETWORK_MANAGEMENT_SET_SECURITY_FLAGS	Set security flags for a network node.
EZWAVECOMMANDTYPE_ZW_GET_TX_POWER_MAX_SUPPORTED	Get the maximum settable tx power.
EZWAVECOMMANDTYPE_REQUESTNODETYPE_NEIGHBORUPDATE	
EZWAVECOMMANDTYPE_TRANSFER_PROTOCOL_CC	
EZWAVECOMMANDTYPE_ENABLE_NODE_NLS	
EZWAVECOMMANDTYPE_GET_NODE_NLS_STATE	
EZWAVECOMMANDTYPE_ZW_GET_INCLUDED_NLS_NODES	
EZWAVECOMMANDTYPE_SEND_PROTOCOL_DATA_CB	
NUM_EZWAVECOMMANDTYPE	

EZwaveCommandStatusType

EZwaveCommandStatusType

Z-Wave Command Status Type used by [SZwaveCommandStatusPackage](#).

Enumerator	
EZWAVECOMMANDSTATUS_TX	EZWAVECOMMANDSTATUS_TX.
EZWAVECOMMANDSTATUS_GENERATE_RANDOM	EZWAVECOMMANDSTATUS_GENERATE_RANDOM.
EZWAVECOMMANDSTATUS_NODE_INFO	EZWAVECOMMANDSTATUS_NODE_INFO.
EZWAVECOMMANDSTATUS_LEARN_MODE_STATUS	EZWAVECOMMANDSTATUS_LEARN_MODE_STATUS.
EZWAVECOMMANDSTATUS_SET_DEFAULT	
EZWAVECOMMANDSTATUS_INVALID_TX_REQUEST	Received when protocol has finished starting up, and after receiving a set default command, has no content.
EZWAVECOMMANDSTATUS_INVALID_COMMAND	Received Tx Request that was not a EZWAVETRANSMITTYPE.
EZWAVECOMMANDSTATUS_SET_RF_RECEIVE_MODE	Receveid command that was not a EZWAVECOMMANDTYPE value.
EZWAVECOMMANDSTATUS_IS_NODE_WITHIN_DIRECT_RANGE	EZWAVECOMMANDSTATUS_IS_NODE_WITHIN_DIRECT_RANGE.
EZWAVECOMMANDSTATUS_GET_NEIGHBOR_COUNT	EZWAVECOMMANDSTATUS_GET_NEIGHBOR_COUNT.
EZWAVECOMMANDSTATUS_ARE_NODES_NEIGHBOURS	EZWAVECOMMANDSTATUS_ARE_NODES_NEIGHBOURS.
EZWAVECOMMANDSTATUS_IS_FAILED_NODE_ID	EZWAVECOMMANDSTATUS_IS_FAILED_NODE_ID.
EZWAVECOMMANDSTATUS_GET_ROUTING_TABLE_LINE	EZWAVECOMMANDSTATUS_GET_ROUTING_TABLE_LINE.
EZWAVECOMMANDSTATUS_SET_ROUTING_INFO	EZWAVECOMMANDSTATUS_SET_ROUTING_INFO.
EZWAVECOMMANDSTATUS_STORE_NODE_INFO	EZWAVECOMMANDSTATUS_STORE_NODE_INFO.
EZWAVECOMMANDSTATUS_GET_PRIORITY_ROUTE	EZWAVECOMMANDSTATUS_GET_PRIORITY_ROUTE.
EZWAVECOMMANDSTATUS_SET_PRIORITY_ROUTE	EZWAVECOMMANDSTATUS_SET_PRIORITY_ROUTE.
EZWAVECOMMANDSTATUS_SET_SLAVE_LEARN_MODE	EZWAVECOMMANDSTATUS_SET_SLAVE_LEARN_MODE.

EZWAVECOMMANDSTATUS_SET_SLAVE_LEARN_MODE_RESULT	EZWAVECOMMANDSTATUS_SET_SLAVE_LEARN_
EZWAVECOMMANDSTATUS_IS_VIRTUAL_NODE	EZWAVECOMMANDSTATUS_IS_VIRTUAL_NODE.
EZWAVECOMMANDSTATUS_GET_VIRTUAL_NODES	EZWAVECOMMANDSTATUS_GET_VIRTUAL_NODE
EZWAVECOMMANDSTATUS_GET_CONTROLLER_CAPABILITIES	EZWAVECOMMANDSTATUS_GET_CONTROLLER_C
EZWAVECOMMANDSTATUS_IS_PRIMARY_CTRL	EZWAVECOMMANDSTATUS_IS_PRIMARY_CTRL.
EZWAVECOMMANDSTATUS_NETWORK_MANAGEMENT	EZWAVECOMMANDSTATUS_NETWORK_MANAGE
EZWAVECOMMANDSTATUS_GET_BACKGROUND_RSSI	EZWAVECOMMANDSTATUS_GET_BACKGROUND_
EZWAVECOMMANDSTATUS_AES_ECB	EZWAVECOMMANDSTATUS_AES_ECB.
EZWAVECOMMANDSTATUS_REMOVE_FAILED_NODE_ID	EZWAVECOMMANDSTATUS_REMOVE_FAILED_NO
EZWAVECOMMANDSTATUS_REPLACE_FAILED_NODE_ID	EZWAVECOMMANDSTATUS_REPLACE_FAILED_NO
EZWAVECOMMANDSTATUS_NETWORK_LEARN_MODE_START	EZWAVECOMMANDSTATUS_NETWORK_LEARN_M
EZWAVECOMMANDSTATUS_ZW_SET_MAX_INCL_REQ_INTERVALS	EZWAVECOMMANDSTATUS_ZW_SET_MAX_INCL_
EZWAVECOMMANDSTATUS_NVM_BACKUP_RESTORE	EZWAVECOMMANDSTATUS_NVM_BACKUP_REST
EZWAVECOMMANDSTATUS_PM_SET_POWERDOWN_CALLBACK	EZWAVECOMMANDSTATUS_PM_SET_POWERDOV
EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_NODES	EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_
EZWAVECOMMANDSTATUS_ZW_REQUESTNODENEIGHBORUPDATE	EZWAVECOMMANDSTATUS_ZW_REQUESTNODE
EZWAVECOMMANDSTATUS_ZW_REQUESTNODETYPE_NEIGHBORUPDATE	EZWAVECOMMANDSTATUS_ZW_REQUESTNODE
EZWAVECOMMANDSTATUS_ZW_INITIATE_SHUTDOWN	EZWAVECOMMANDSTATUS_ZW_INITIATE_SHUTD
EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_LR_NODES	EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_
EZWAVECOMMANDSTATUS_ZW_GET_LR_CHANNEL	EZWAVECOMMANDSTATUS_ZW_GET_LR_CHANNE
EZWAVECOMMANDSTATUS_UNUSED1	EZWAVECOMMANDSTATUS_UNUSED1.
EZWAVECOMMANDSTATUS_UNUSED2	EZWAVECOMMANDSTATUS_UNUSED2.
EZWAVECOMMANDSTATUS_ZW_GET_PTI_CONFIG	EZWAVECOMMANDSTATUS_ZW_GET_PTI_CONFIG
EZWAVECOMMANDSTATUS_ZW_SET_TX_ATTENUATION	EZWAVECOMMANDSTATUS_ZW_SET_TX_ATTENU
EZWAVECOMMANDSTATUS_SECURE_ON_NODE_ADDED	The node is completely added using the Controller
EZWAVECOMMANDSTATUS_SECURE_ON_NODE_DELETED	The node is completely removed using the Contro
EZWAVECOMMANDSTATUS_SECURE_ON_NEW_NETWORK_ENTERED	Node entered a new network.
EZWAVECOMMANDSTATUS_SECURE_ON_NETWORK_MANAGEMENT_STATE_UPDATE	Secure network management changed state.
EZWAVECOMMANDSTATUS_SECURE_ON_NEW_SUC	New SUC NodeID received.
EZWAVECOMMANDSTATUS_SECURE_ON_FRAME_TRANSMISSION	Frame transmission result.
EZWAVECOMMANDSTATUS_SECURE_ON_RX_FRAME_RECEIVED_INDICATOR	Frame received from NodeID indicator.
EZWAVECOMMANDSTATUS_ZW_SET_LR_CHANNEL	EZWAVECOMMANDSTATUS_ZW_SET_LR_CHANNE
EZWAVECOMMANDSTATUS_ZW_GET_TX_POWER_MAX_SUPPORTED	EZWAVECOMMANDSTATUS_ZW_GET_TX_POWER.
EZWAVECOMMANDSTATUS_GET_NODE-NLS_STATE	EZWAVECOMMANDSTATUS_GET_NODE-NLS_STA
EZWAVECOMMANDSTATUS_ENABLE_NODE-NLS	EZWAVECOMMANDSTATUS_ENABLE_NODE-NLS
EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED-NLS_NODES	EZWAVECOMMANDSTATUS_ZW_GET_INCLUDED_
NUM_EZWAVECOMMANDSTATUS	NUM_EZWAVECOMMANDSTATUS.
EZWAVECOMMANDSTATUS_INVALID	

EZwaveReceiveType

EZwaveReceiveType

Z-Wave Receive type.

Enumerator	
EZWAVERECEIVETYPE_SINGLE	EZWAVERECEIVETYPE_SINGLE.
EZWAVERECEIVETYPE_MULTI	EZWAVERECEIVETYPE_MULTI.
EZWAVERECEIVETYPE_NODE_UPDATE	EZWAVERECEIVETYPE_NODE_UPDATE.
EZWAVERECEIVETYPE_SECURITY_EVENT	EZWAVERECEIVETYPE_SECURITY_EVENT.
EZWAVERECEIVETYPE_STAY_AWAKE	EZWAVERECEIVETYPE_STAY_AWAKE.
EZWAVERECEIVETYPE_SECURE_FRAME_RECEIVED	Event received from the SECURE module.
EZWAVERECEIVETYPE_REQUEST_ENCRYPTION_FRAME	EZWAVERECEIVETYPE_REQUEST_ENCRYPTION_FRAME.
NUM_EZWAVERECEIVETYPE	NUM_EZWAVERECEIVETYPE.
EZWAVERECEIVETYPE_INVALID	

EProtocolEvent

EProtocolEvent

Enumerator	
EPROTOCOLEVENT_RFRXBEAM	
EPROTOCOLEVENT_RFTXCOMPLETE	
EPROTOCOLEVENT_RFRX	
EPROTOCOLEVENT_RFTXBEAM	
EPROTOCOLEVENT_RFTXFAILLBT	
EPROTOCOLEVENT_RFTXFAIL	
EPROTOCOLEVENT_RFRXABORT	
EPROTOCOLEVENT_TIMER	
EPROTOCOLEVENT_EXPLORE	
EPROTOCOLEVENT_APP_TX	
EPROTOCOLEVENT_APP_COMMAND	
EPROTOCOLEVENT_NETWORKID_UPDATE	
EPROTOCOLEVENT_CHANGE_RADIO_PHY	
EPROTOCOLEVENT_RADIO_ASSERT	
EPROTOCOLEVENT_RADIO_TX_TIMEOUT	
EPROTOCOLEVENT_RADIO_CALIBRATE	
NUM_EPROTOCOLEVENT	

ERequestEncryptionEvent

ERequestEncryptionEvent

Enumerator	
ERPCCEEVENT_SERIALAPI_FAIL	
ERPCCEEVENT_SERIALAPI_OK	

ELearnMode

ELearnMode

Enumerator

ELEARNMODE_DISABLED	
ELEARNMODE_CLASSIC	
ELEARNMODE_NETWORK_WIDE_INCLUSION	
ELEARNMODE_NETWORK_WIDE_EXCLUSION	

Typedef Documentation

EProtocolType

```
typedef enum EProtocolType EProtocolType
```

ELibraryType

```
typedef enum ELibraryType ELibraryType
```

EInclusionState_t

```
typedef enum EInclusionState_t EInclusionState_t
```

EListenBeforeTalkThreshold_t

```
typedef enum EListenBeforeTalkThreshold_t EListenBeforeTalkThreshold_t
```

EtxPowerLevel_t

```
typedef enum EtxPowerLevel_t EtxPowerLevel_t
```

EZwaveTransmitType

```
typedef enum EZwaveTransmitType EZwaveTransmitType
```

Transmit Type enum used by [SZwaveTransmitPackage](#).

ELearnStatus

```
typedef enum ELearnStatus ELearnStatus
```

Learn mode status updates.

EZwaveCommandType

```
typedef enum EZwaveCommandType EZwaveCommandType
```

Z-Wave Command Types.

EZwaveCommandStatusType

```
typedef enum EZwaveCommandStatusType EZwaveCommandStatusType
```

Z-Wave Command Status Type used by [SZwaveCommandStatusPackage](#).

EZwaveReceiveType

```
typedef enum EZwaveReceiveType EZwaveReceiveType
```

Z-Wave Receive type.

EProtocolEvent

```
typedef enum EProtocolEvent EProtocolEvent
```

ERequestEncryptionEvent

```
typedef enum ERequestEncryptionEvent ERequestEncryptionEvent
```

SNetworkUpdateRequest

```
typedef struct SNetworkUpdateRequest SNetworkUpdateRequest
```

Network Update Request.

ELearnMode

```
typedef enum ELearnMode ELearnMode
```

SNodeInfoRequest

```
typedef struct SNodeInfoRequest SNodeInfoRequest
```

SNodeInfo

```
typedef struct SNodeInfo SNodeInfo
```

SCommandClassList_t

```
typedef struct SCommandClassList_t SCommandClassList_t
```

SCommandClassSet_t

```
typedef struct SCommandClassSet_t SCommandClassSet_t
```

List of Command Classes supported when device is not included, included insecurely or included securely.

SCommandClassVersions

```
typedef struct SCommandClassVersions SCommandClassVersions
```

SCommandGenerateRandom

```
typedef struct SCommandGenerateRandom SCommandGenerateRandom
```

Generates true random word.

SCommandNodeInfo

```
typedef struct SCommandNodeInfo SCommandNodeInfo
```

SCommandClearNetworkStatistics

```
typedef struct SCommandClearNetworkStatistics SCommandClearNetworkStatistics
```

SCommandSetLearnMode

```
typedef struct SCommandSetLearnMode SCommandSetLearnMode
```

SCommandSetSmartStartLearnMode

```
typedef struct SCommandSetSmartStartLearnMode SCommandSetSmartStartLearnMode
```

SSetSucNodeId

```
typedef struct SSetSucNodeId SSetSucNodeId
```

Controller API.

SCommandSetRfPowerLevel

```
typedef struct SCommandSetRfPowerLevel SCommandSetRfPowerLevel
```

SSendSucNodeId

```
typedef struct SSendSucNodeId SSendSucNodeId
```

SCommandSetPromiscuousMode

```
typedef struct SCommandSetPromiscuousMode SCommandSetPromiscuousMode
```

SAssignReturnRoute

```
typedef struct SAssignReturnRoute SAssignReturnRoute
```

SCommandSetRfReceiveMode

```
typedef struct SCommandSetRfReceiveMode SCommandSetRfReceiveMode
```

SDeleteReturnRoute

```
typedef struct SDeleteReturnRoute SDeleteReturnRoute
```

SCommandGeneric8bParameter

```
typedef struct SCommandGeneric8bParameter SCommandGeneric8bParameter
```

SCommandGeneric16bParameter

```
typedef struct SCommandGeneric16bParameter SCommandGeneric16bParameter
```

SCommandAreNodesNeighbours

```
typedef struct SCommandAreNodesNeighbours SCommandAreNodesNeighbours
```

SCommandGetRoutingInfo

```
typedef struct SCommandGetRoutingInfo SCommandGetRoutingInfo
```

SSendSlaveNodeInformation

```
typedef struct SSendSlaveNodeInformation SSendSlaveNodeInformation
```

SCommandSetRoutingInfo

```
typedef struct SCommandSetRoutingInfo SCommandSetRoutingInfo
```

SCommandStoreNodeInfo

```
typedef struct SCommandStoreNodeInfo SCommandStoreNodeInfo
```

SCommandStoreHomeID

```
typedef struct SCommandStoreHomeID SCommandStoreHomeID
```

SCommandSetPriorityRoute

```
typedef struct SCommandSetPriorityRoute SCommandSetPriorityRoute
```

SCommandAesEcb

```
typedef struct SCommandAesEcb SCommandAesEcb
```

SCommandGetPriorityRoute

```
typedef struct SCommandGetPriorityRoute SCommandGetPriorityRoute
```

SCommandFailedNodeIDCmd

```
typedef struct SCommandFailedNodeIDCmd SCommandFailedNodeIDCmd
```

SCommandPMSetPowerDownCallback

```
typedef struct SCommandPMSetPowerDownCallback SCommandPMSetPowerDownCallback
```

SCommandSetLBTThreshold

```
typedef struct SCommandSetLBTThreshold SCommandSetLBTThreshold
```

SCommandSetMaxInclReqInterval

```
typedef struct SCommandSetMaxInclReqInterval SCommandSetMaxInclReqInterval
```

SCommandNvmBackupRestore

```
typedef struct SCommandNvmBackupRestore SCommandNvmBackupRestore
```

SCommandSetSecurityKeys

```
typedef struct SCommandSetSecurityKeys SCommandSetSecurityKeys
```

SCommandEnableNodeNLS

```
typedef struct SCommandEnableNodeNLS SCommandEnableNodeNLS
```

SCommandGetNodeNLSState

```
typedef struct SCommandGetNodeNLSState SCommandGetNodeNLSState
```

SCommandSendProtocolDataCb

```
typedef struct SCommandSendProtocolDataCb SCommandSendProtocolDataCb
```

SCommandGetNLSNodes

```
typedef struct SCommandGetNLSNodes SCommandGetNLSNodes
```

SProtocolVersion

```
typedef struct SProtocolVersion SProtocolVersion
```

SProtocolInfo

```
typedef struct SProtocolInfo SProtocolInfo
```

Protocol Info data.

SNetworkInfo

```
typedef struct SNetworkInfo SNetworkInfo
```

Network Info.

SLongRangeInfo

```
typedef struct SLongRangeInfo SLongRangeInfo
```

t_ExtNodeInfo

```
typedef struct t_ExtNodeInfo t_ExtNodeInfo
```

SRadioStatus

```
typedef struct SRadioStatus SRadioStatus
```

SApplicationHandles

```
typedef struct SApplicationHandles SApplicationHandles
```

SRadioConfig_t

```
typedef struct SRadioConfig_t SRadioConfig_t
```

SAppNodeInfo_t

```
typedef struct SAppNodeInfo_t SAppNodeInfo_t
```

SVirtualSlaveNodeInfo_t

```
typedef struct SVirtualSlaveNodeInfo_t SVirtualSlaveNodeInfo_t
```

SVirtualSlaveNodeInfoTable_t

```
typedef struct SVirtualSlaveNodeInfoTable_t SVirtualSlaveNodeInfoTable_t
```

SProtocolConfig_t


```
typedef struct SProtocolConfig_t SProtocolConfig_t
```

This allows "nulling" a pointer while modifying a virtual slave node info Or changing pointer to point to a different one.

This struct content must be set up by application before enabling protocol (enabling radio) Direct content (the pointers) may not be changed runtime, but the data they point to can be edited by application run time

STransmitFrameConfig

```
typedef struct STransmitFrameConfig STransmitFrameConfig
```

STransmitProtocolFrameConfig

```
typedef struct STransmitProtocolFrameConfig STransmitProtocolFrameConfig
```

SExploreInclusionRequest

```
typedef struct SExploreInclusionRequest SExploreInclusionRequest
```

SExploreExclusionRequest

```
typedef struct SExploreExclusionRequest SExploreExclusionRequest
```

SIncludedNodeInfo

```
typedef struct SIncludedNodeInfo SIncludedNodeInfo
```

Contains info related to sending an INIF.

STest

```
typedef struct STest STest
```

SSendData

```
typedef struct SSendData SSendData
```

SSendDataEx

```
typedef struct SSendDataEx SSendDataEx
```

SSendDataBridge

```
typedef struct SSendDataBridge SSendDataBridge
```

SSendDataMulti

```
typedef struct SSendDataMulti SSendDataMulti
```

SSendDataMultiEx

```
typedef struct SSendDataMultiEx SSendDataMultiEx
```

SSendDataMultiBridge

```
typedef struct SSendDataMultiBridge SSendDataMultiBridge
```

SSendProtocolData

```
typedef struct SSendProtocolData SSendProtocolData
```

SCommandNetworkManagement

```
typedef struct SCommandNetworkManagement SCommandNetworkManagement
```

SRequestNewRouteDestinations

```
typedef struct SRequestNewRouteDestinations SRequestNewRouteDestinations
```

SCommandNetworkManagementDSK

```
typedef struct SCommandNetworkManagementDSK SCommandNetworkManagementDSK
```

SZWaveTransmitStatus

```
typedef struct SZWaveTransmitStatus SZWaveTransmitStatus
```

SZWaveGenerateRandomStatus

```
typedef struct SZWaveGenerateRandomStatus SZWaveGenerateRandomStatus
```

SZWaveNodeInfoStatus

```
typedef struct SZWaveNodeInfoStatus SZWaveNodeInfoStatus
```

SZWaveLearnModeStatus

```
typedef struct SZWaveLearnModeStatus SZWaveLearnModeStatus
```

SZWaveInvalidTxRequestStatus

```
typedef struct SZWaveInvalidTxRequestStatus SZWaveInvalidTxRequestStatus
```

SZWaveInvalidCommandStatus

```
typedef struct SZWaveInvalidCommandStatus SZWaveInvalidCommandStatus
```

SZWaveGeneric8bStatus

```
typedef struct SZWaveGeneric8bStatus SZWaveGeneric8bStatus
```

SZWaveGenericBoolStatus

```
typedef struct SZWaveGenericBoolStatus SZWaveGenericBoolStatus
```

SCommandRequestNodeNeighborUpdate

```
typedef struct SCommandRequestNodeNeighborUpdate SCommandRequestNodeNeighborUpdate
```

SCommandRequestNodeTypeNeighborUpdate

```
typedef struct SCommandRequestNodeTypeNeighborUpdate SCommandRequestNodeTypeNeighborUpdate
```

SZWaveGetRoutingInfoStatus

```
typedef struct SZWaveGetRoutingInfoStatus SZWaveGetRoutingInfoStatus
```

SCommandInitiateShutdown

```
typedef struct SCommandInitiateShutdown SCommandInitiateShutdown
```

SCommandSetSecurityFlags

```
typedef struct SCommandSetSecurityFlags SCommandSetSecurityFlags
```

SZWaveGetPriorityRouteStatus

```
typedef struct SZWaveGetPriorityRouteStatus SZWaveGetPriorityRouteStatus
```

SZWaveSetPriorityRouteStatus

```
typedef struct SZWaveSetPriorityRouteStatus SZWaveSetPriorityRouteStatus
```

SZWaveGetVirtualNodesStatus

```
typedef struct SZWaveGetVirtualNodesStatus SZWaveGetVirtualNodesStatus
```

SZWaveAesEcbStatus

```
typedef struct SZWaveAesEcbStatus SZWaveAesEcbStatus
```

SZWaveGetBackgroundRssiStatus

```
typedef struct SZWaveGetBackgroundRssiStatus SZWaveGetBackgroundRssiStatus
```

SZWaveNetworkManagementStatus

```
typedef struct SZWaveNetworkManagementStatus SZWaveNetworkManagementStatus
```

SNvmBackupRestoreStatus

```
typedef struct SNvmBackupRestoreStatus SNvmBackupRestoreStatus
```

SCommandGetNodeNLSStateStatus

```
typedef struct SCommandGetNodeNLSStateStatus SCommandGetNodeNLSStateStatus
```

SCommandEnableNodeNLSStatus

```
typedef struct SCommandEnableNodeNLSStatus SCommandEnableNodeNLSStatus
```

SZWaveGetIncludedNodes

```
typedef struct SZWaveGetIncludedNodes SZWaveGetIncludedNodes
```

SZWaveGetIncludedNodesLR

```
typedef struct SZWaveGetIncludedNodesLR SZWaveGetIncludedNodesLR
```

SZWaveGetIncludedNodesNLS

```
typedef struct SZWaveGetIncludedNodesNLS SZWaveGetIncludedNodesNLS
```

SZWaveTxPowerMaxSupported

```
typedef struct SZWaveTxPowerMaxSupported SZWaveTxPowerMaxSupported
```

SCommandTransferProtocolCC

```
typedef struct SCommandTransferProtocolCC SCommandTransferProtocolCC
```

SReceiveNodeUpdate

```
typedef struct SReceiveNodeUpdate SReceiveNodeUpdate
```

The Z Wave protocol MAY notify an application by sending [SReceiveNodeUpdate](#) when a Node Information Frame has been received.

Parameters

Type	Direction	Argument Name	Description
	[in]	bNodeID	
	[in]	pCmd	Pointer of the updated node's node info.
	[in]	bLen	The length of the pCmd parameter.

The Z Wave protocol MAY refrain from sending the information if the protocol is currently expecting node information.

A controller application MAY use the information provided by [SReceiveNodeUpdate](#) to update local data structures.

The Z Wave protocol MUST notify a controller application by calling [SReceiveNodeUpdate](#) when a new node has been added or deleted from the controller through the network management features.

The Z Wave protocol MUST pass [SReceiveNodeUpdate](#) to application in response to [SNodeInfoRequest](#) being passed to protocol by the controller application. The Z Wave protocol MAY notify a controller application by sending [SReceiveNodeUpdate](#) when a Node Information Frame has been received. The Z Wave protocol MAY refrain from sending the information if the protocol is currently expecting a Node Information frame.

[SReceiveNodeUpdate](#) MUST be sent in a controller node operating as SIS each time a node is added or deleted by the primary controller. [SReceiveNodeUpdate](#) MUST be sent in a controller node operating as SIS each time a node is added/deleted by an inclusion controller.

A controller application MAY send a ZW_RequestNetWorkUpdate command to a SIS or SIS node. In response, the SIS MUST return update information for each node change since the last update handled by the requesting controller node. The application of the requesting controller node MAY receive multiple instances of [SReceiveNodeUpdate](#) in response to application passing [SNetworkUpdateRequest](#) to protocol.

The Z Wave protocol MUST NOT send [SReceiveNodeUpdate](#) in a controller node acting as primary controller or inclusion controller when a node is added or deleted.

Any controller application MUST implement this function.

```
serialapi{ZW->HOST: REQ | 0x49 | bStatus | bNodeID | bLen | basic | generic | specific | commandclasses[ ]}
```

[SReceiveNodeUpdate](#) via the Serial API also have the possibility for receiving the status UPDATE_STATE_NODE_INFO_REQ_FAILED, which means that a node did not acknowledge a [SNodeInfoRequest](#) .

SReceiveSecurityEvent

```
typedef struct SReceiveSecurityEvent SReceiveSecurityEvent
```

UTransmitParameters

```
typedef union UTransmitParameters UTransmitParameters
```

Transmit Parameters Union used by [SZwaveTransmitPackage](#).

UCommandStatus

```
typedef union UCommandStatus UCommandStatus
```

Command Status Union used by [SZwaveCommandStatusPackage](#).

UReceiveCmdPayload

```
typedef union UReceiveCmdPayload UReceiveCmdPayload
```

UCommandParameters

```
typedef union UCommandParameters UCommandParameters
```

SZwaveTransmitPackage

```
typedef struct SZwaveTransmitPackage SZwaveTransmitPackage
```

SZwaveCommandStatusPackage

```
typedef struct SZwaveCommandStatusPackage SZwaveCommandStatusPackage
```

Z-Wave Command Status Package.

SZwaveCommandPackage

```
typedef struct SZwaveCommandPackage SZwaveCommandPackage
```

SReceiveSingle

```
typedef struct SReceiveSingle SReceiveSingle
```

SReceiveMulti

```
typedef struct SReceiveMulti SReceiveMulti
```

SProtocolTxOptions

```
typedef struct SProtocolTxOptions SProtocolTxOptions
```

SRequestEncryption

```
typedef struct SRequestEncryption SRequestEncryption
```

UReceiveParameters

```
typedef union UReceiveParameters UReceiveParameters
```

SZwaveReceivePackage

```
typedef struct SZwaveReceivePackage SZwaveReceivePackage
```


SNetworkUpdateRequest

Network Update Request.

Public Attributes

void(* [Handle](#))

Public Attribute Documentation

Handle

```
void(* SNetworkUpdateRequest::Handle) (void)
```

SNodeInfoRequest

Public Attributes

void(*)
node_id_t

Handle
DestNodeId

Public Attribute Documentation

Handle

void(*) SNodeInfoRequest::Handle) (void)

DestNodeId

node_id_t SNodeInfoRequest::DestNodeId

SNodeInfo

Public Attributes

void(*)	Handle
node_id_t	DestNodeId
uint8_t	TransmitOptions

Public Attribute Documentation

Handle

```
void(*) SNodeInfo::Handle) (void)
```

DestNodeId

```
node_id_t SNodeInfo::DestNodeId
```

TransmitOptions

```
uint8_t SNodeInfo::TransmitOptions
```

SCommandClassList_t

Public Attributes

uint8_t	iListLength
uint8_t *	pCommandClasses

Public Attribute Documentation

iListLength

uint8_t SCommandClassList_t::iListLength

pCommandClasses

uint8_t* SCommandClassList_t::pCommandClasses

SCommandClassSet_t

List of Command Classes supported when device is not included, included insecurely or included securely.

Public Attributes

SCommandClassList_t	UnSecureIncludedCC List of UNsecure supported command classes.
SCommandClassList_t	SecureIncludedUnSecureCC List of UNsecure supported command classes in secure network.
SCommandClassList_t	SecureIncludedSecureCC List of Secure supported command classes.

Public Attribute Documentation

UnSecureIncludedCC

SCommandClassList_t SCommandClassSet_t::UnSecureIncludedCC

List of UNsecure supported command classes.

Available when node is NOT included or UNsecure included

SecureIncludedUnSecureCC

SCommandClassList_t SCommandClassSet_t::SecureIncludedUnSecureCC

List of UNsecure supported command classes in secure network.

Available when node is secure included

SecureIncludedSecureCC

SCommandClassList_t SCommandClassSet_t::SecureIncludedSecureCC

List of Secure supported command classes.

Available when node is secure included

SCommandClassVersions

Public Attributes

uint8_t	SecurityVersion
uint8_t	Security2Version
uint8_t	TransportServiceVersion

Public Attribute Documentation

SecurityVersion

uint8_t SCommandClassVersions::SecurityVersion

Security2Version

uint8_t SCommandClassVersions::Security2Version

TransportServiceVersion

uint8_t SCommandClassVersions::TransportServiceVersion

SCommandGenerateRandom

Generates true random word.

Public Attributes

uint8_t iLength

Public Attribute Documentation

iLength

uint8_t SCommandGenerateRandom::iLength

SCommandNodeInfo

Public Attributes

`node_id_t` `NodeId`

Public Attribute Documentation

NodeId

`node_id_t` SCommandNodeInfo::NodeId

SCommandClearNetworkStatistics

Public Attributes

uint8_t [Reserved](#)

Public Attribute Documentation

Reserved

uint8_t SCommandClearNetworkStatistics::Reserved

SCommandSetLearnMode

Public Attributes

- [ELearnMode](#)
- [eLearnMode](#)
- [uint8_t](#)
- [useCB](#)

Public Attribute Documentation

eLearnMode

ELearnMode SCommandSetLearnMode::eLearnMode

useCB

uint8_t SCommandSetLearnMode::useCB

SCommandSetSmartStartLearnMode

Public Attributes

E_NETWORK_LEA eLearnMode
RN_MODE_ACTIO
N

Public Attribute Documentation

eLearnMode

E_NETWORK_LEARN_MODE_ACTION SCommandSetSmartStartLearnMode::eLearnMode

SSetSucNodeId

Controller API.

Public Attributes

void(*)	Handle	
node_id_t	SucNodeId	Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.
bool	bSucEnable	
bool	bTxLowPower	
uint8_t	Capabilities	

Public Attribute Documentation

Handle

```
void(* SSetSucNodeId::Handle) (void)
```

SucNodeId

```
node_id_t SSetSucNodeId::SucNodeId
```

Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.

bSucEnable

```
bool SSetSucNodeId::bSucEnable
```

bTxLowPower

```
bool SSetSucNodeId::bTxLowPower
```

Capabilities

```
uint8_t SSetSucNodeId::Capabilities
```

SCommandSetRfPowerLevel

Public Attributes

uint8_t powerLevelDBm

Public Attribute Documentation

powerLevelDBm

uint8_t SCommandSetRfPowerLevel::powerLevelDBm

SSendSucNodeId

Public Attributes

<code>void(*</code>	<code>Handle</code>	
<code>node_id_t</code>	<code>DestNodeId</code>	Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.
<code>uint8_t</code>	<code>TransmitOptions</code>	

Public Attribute Documentation

Handle

```
void(* SSendSucNodeId::Handle) (void)
```

DestNodeId

```
node_id_t SSendSucNodeId::DestNodeId
```

Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.

TransmitOptions

```
uint8_t SSendSucNodeId::TransmitOptions
```

SCommandSetPromiscuousMode

Public Attributes

uint8_t [Enable](#)

Public Attribute Documentation

Enable

uint8_t SCommandSetPromiscuousMode::Enable

SAssignReturnRoute

Public Attributes

void(*)	Handle	
node_id_t	ReturnRouteReceiverNodeId	Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.
node_id_t	RouteDestinationNodeId	Routing slave to recieve route.
uint8_t	aPriorityRouteRepeaters	Destination of route (if 0 destination will be self). Destination can be a SUC.
uint8_t	PriorityRouteSpeed	Route to be assigned as priority route - set to zeroes to NOT supply a priority route (recommended)
uint8_t	isSucRoute	

Public Attribute Documentation

Handle

```
void(* SAssignReturnRoute::Handle) (void)
```

ReturnRouteReceiverNodeId

```
node_id_t SAssignReturnRoute::ReturnRouteReceiverNodeId
```

Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.

RouteDestinationNodeId

```
node_id_t SAssignReturnRoute::RouteDestinationNodeId
```

Routing slave to recieve route.

aPriorityRouteRepeaters

```
uint8_t SAssignReturnRoute::aPriorityRouteRepeaters[4]
```

Destination of route (if 0 destination will be self). Destination can be a SUC.

PriorityRouteSpeed

uint8_t SAssignReturnRoute::PriorityRouteSpeed

Route to be assigned as priority route - set to zeroes to NOT supply a priority route (recommended)

isSucRoute

uint8_t SAssignReturnRoute::isSucRoute

SCommandSetRfReceiveMode

Public Attributes

uint8_t mode

Public Attribute Documentation

mode

uint8_t SCommandSetRfReceiveMode::mode

SDeleteReturnRoute

Public Attributes

void(*)	Handle	
node_id_t	DestNodeId	Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.
bool	bDeleteSuc	Node to have its return routes deleted..

Public Attribute Documentation

Handle

```
void(*) SDeleteReturnRoute::Handle) (void)
```

DestNodeId

```
node_id_t SDeleteReturnRoute::DestNodeId
```

Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.

bDeleteSuc

```
bool SDeleteReturnRoute::bDeleteSuc
```

Node to have its return routes deleted..

SCommandGeniric8bParameter

Public Attributes

uint8_t [value](#)

Public Attribute Documentation

value

uint8_t SCommandGeniric8bParameter::value

SCommandGeniric16bParameter

Public Attributes

uint16_t value

Public Attribute Documentation

value

uint16_t SCommandGeniric16bParameter::value

SCommandAreNodesNeighbours

Public Attributes

uint8_t [NodeA](#)

uint8_t [NodeB](#)

Public Attribute Documentation

NodeA

uint8_t SCommandAreNodesNeighbours::NodeA

NodeB

uint8_t SCommandAreNodesNeighbours::NodeB

SCommandGetRoutingInfo

Public Attributes

node_id_t

nodeID

uint8_t

options

Public Attribute Documentation

nodeID

node_id_t SCommandGetRoutingInfo::nodeID

options

uint8_t SCommandGetRoutingInfo::options

SSendSlaveNodeInformation

Public Attributes

void(*)	Handle	
node_id_t	sourceId	Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.
node_id_t	destinationId	
uint8_t	txOptions	

Public Attribute Documentation

Handle

```
void(*) SSendSlaveNodeInformation::Handle) (void)
```

sourceId

```
node_id_t SSendSlaveNodeInformation::sourceId
```

Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.

destinationId

```
node_id_t SSendSlaveNodeInformation::destinationId
```

txOptions

```
uint8_t SSendSlaveNodeInformation::txOptions
```


SCommandSetRoutingInfo

Public Attributes

- uint8_t [nodeID](#)
- uint8_t [length](#)
- uint8_t [nodeMask](#)

Public Attribute Documentation

nodeID

uint8_t SCommandSetRoutingInfo::nodeID

length

uint8_t SCommandSetRoutingInfo::length

nodeMask

uint8_t SCommandSetRoutingInfo::nodeMask[MAX_NODEMASK_LENGTH]

SCommandStoreNodeInfo

Public Attributes

uint8_t [nodeID](#)

uint8_t [nodeInfo](#)

Public Attribute Documentation

nodeID

```
uint8_t SCommandStoreNodeInfo::nodeID
```

nodeInfo

```
uint8_t SCommandStoreNodeInfo::nodeInfo[6]
```

SCommandStoreHomeID

Public Attributes

uint8_t [homeID](#)
uint8_t [nodeID](#)

Public Attribute Documentation

homeID

uint8_t SCommandStoreHomeID::homeID[HOMEID_LENGTH]

nodeID

uint8_t SCommandStoreHomeID::nodeID

SCommandSetPriorityRoute

Public Attributes

<code>node_id_t</code>	<code>nodeID</code>
<code>uint8_t</code>	<code>repeaters</code>
<code>uint8_t</code>	<code>routeSpeed</code>
<code>uint8_t</code>	<code>clearGolden</code>

Public Attribute Documentation

nodeID

`node_id_t SCommandSetPriorityRoute::nodeID`

repeaters

`uint8_t SCommandSetPriorityRoute::repeaters[MAX_REPEATERS]`

routeSpeed

`uint8_t SCommandSetPriorityRoute::routeSpeed`

clearGolden

`uint8_t SCommandSetPriorityRoute::clearGolden`

SCommandAesEcb

Public Attributes

```
uint8_t  key
uint8_t  inputData
```

Public Attribute Documentation

key

```
uint8_t SCommandAesEcb::key[16]
```

inputData

```
uint8_t SCommandAesEcb::inputData[16]
```

SCommandGetPriorityRoute

Public Attributes

<code>uint8_t *</code>	<code>pPriRouteBuffer</code>
<code>node_id_t</code>	<code>nodeID</code>
	Will be returned with transmit status.

Public Attribute Documentation

pPriRouteBuffer

`uint8_t*` SCommandGetPriorityRoute::pPriRouteBuffer

nodeID

`node_id_t` SCommandGetPriorityRoute::nodeID

Will be returned with transmit status.

SCommandFailedNodeIDCmd

Public Attributes

<code>node_id_t</code>	<code>nodeID</code>
<code>uint8_t</code>	<code>normalPower</code>

Public Attribute Documentation

nodeID

`node_id_t SCommandFailedNodeIDCmd::nodeID`

normalPower

`uint8_t SCommandFailedNodeIDCmd::normalPower`

SCommandPMSetPowerDownCallback

Public Attributes

void(* [callback](#))

Public Attribute Documentation

callback

void(* SCommandPMSetPowerDownCallback::callback) (void)

SCommandSetLBTTreshold

Public Attributes

```
uint8_t channel
int8_t level
```

Public Attribute Documentation

channel

```
uint8_t SCommandSetLBTTreshold::channel
```

level

```
int8_t SCommandSetLBTTreshold::level
```

SCommandSetMaxInclReqInterval

Public Attributes

uint32_t inclusionRequestInterval

Public Attribute Documentation

inclusionRequestInterval

uint32_t SCommandSetMaxInclReqInterval::inclusionRequestInterval

SCommandNvmBackupRestore

Public Attributes

uint32_t	offset
uint32_t	length
uint8_t *	nvmData

Public Attribute Documentation

offset

uint32_t SCommandNvmBackupRestore::offset

length

uint32_t SCommandNvmBackupRestore::length

nvmData

uint8_t* SCommandNvmBackupRestore::nvmData

SCommandSetSecurityKeys

Public Attributes

uint8_t [keys](#)

Public Attribute Documentation

keys

uint8_t SCommandSetSecurityKeys::keys

SCommandEnableNodeNLS

Public Attributes

`node_id_t` `nodeID`

Public Attribute Documentation

`nodeID`

`node_id_t` SCommandEnableNodeNLS::`nodeID`

SCommandGetNodeNLSSState

Public Attributes

`node_id_t` `nodeID`

Public Attribute Documentation

`nodeID`

`node_id_t SCommandGetNodeNLSSState::nodeID`

SCommandSendProtocolDataCb

Public Attributes

uint8_t	callback_id
uint8_t	tx_status
TX_STATUS_TYPE	extended_tx_status

Public Attribute Documentation

callback_id

```
uint8_t SCommandSendProtocolDataCb::callback_id
```

tx_status

```
uint8_t SCommandSendProtocolDataCb::tx_status
```

extended_tx_status

```
TX_STATUS_TYPE SCommandSendProtocolDataCb::extended_tx_status
```

SCommandGetNLSNodes

Public Attributes

<code>node_id_t</code>	<code>nodeID</code>
<code>uint8_t</code>	<code>bitmaskOffset</code>

Public Attribute Documentation

nodeID

`node_id_t SCommandGetNLSNodes::nodeID`

bitmaskOffset

`uint8_t SCommandGetNLSNodes::bitmaskOffset`

SProtocolVersion

Public Attributes

uint8_t	Major
uint8_t	Minor
uint8_t	Revision

Public Attribute Documentation

Major

uint8_t SProtocolVersion::Major

Minor

uint8_t SProtocolVersion::Minor

Revision

uint8_t SProtocolVersion::Revision

SProtocolInfo

Protocol Info data.

Public Attributes

SCommandClassVersions	CommandClassVersions Versions of Command Classes supplied by protocol.
SProtocolVersion	ProtocolVersion Protocol version.
EProtocolType	eProtocolType
ELibraryType	eLibraryType Protocol library type.

Public Attribute Documentation

CommandClassVersions

SCommandClassVersions SProtocolInfo::CommandClassVersions

Versions of Command Classes supplied by protocol.

ProtocolVersion

SProtocolVersion SProtocolInfo::ProtocolVersion

Protocol version.

eProtocolType

EProtocolType SProtocolInfo::eProtocolType

eLibraryType

ELibraryType SProtocolInfo::eLibraryType

Protocol library type.

SNetworkInfo

Network Info.

Public Attributes

EInclusionState_t	eInclusionState
node_id_t	SucNodeId
uint8_t	SecurityKeys
node_id_t	NodeId
uint32_t	HomeId
uint16_t	MaxPayloadSize

Public Attribute Documentation

eInclusionState

EInclusionState_t SNetworkInfo::eInclusionState

SucNodeId

node_id_t SNetworkInfo::SucNodeId

SecurityKeys

uint8_t SNetworkInfo::SecurityKeys

NodeId

node_id_t SNetworkInfo::NodeId

HomeId

uint32_t SNetworkInfo::HomeId

MaxPayloadSize

uint16_t SNetworkInfo::MaxPayloadSize

SLongRangeInfo

Public Attributes

uint16_t [MaxLongRangePayloadSize](#)

Public Attribute Documentation

MaxLongRangePayloadSize

uint16_t SLongRangeInfo::MaxLongRangePayloadSize

t_ExtNodeInfo

Public Attributes

- [NODEINFO](#)[NodeInfo](#)
- [uint8_t](#)[extInfo](#)

Public Attribute Documentation

NodeInfo

NODEINFO t_ExtNodeInfo::NodeInfo

extInfo

uint8_t t_ExtNodeInfo::extInfo

SRadioStatus

Public Attributes

int8_t iRadioPowerLevel

Public Attribute Documentation

iRadioPowerLevel

int8_t SRadioStatus::iRadioPowerLevel

SApplicationHandles

Public Attributes

SQueueNotifying *	pZwTxQueue Notifying Queue object (contains FreeRTOS queue)
QueueHandle_t	ZwRxQueue FreeRTOS Queue handle - Queue for ZW frames forwarded from protocol to application.
SQueueNotifying *	pZwCommandQueue Notifying Queue object (contains FreeRTOS queue)
QueueHandle_t	ZwCommandStatusQueue FreeRTOS Queue handle - Queue for status replies from protocol to application (status on commands from application to protocol and Tx request from App to protocol)
const zpal_radio_network_stats_t *	pNetworkStatistics Network statistics supplied by protocol.
const SProtocolInfo *	pProtocolInfo
const SNetworkInfo *	pNetworkInfo
const SLongRangeInfo *	pLongRangeInfo Only used by Controller nodes.
const SRadioStatus *	pRadioStatus

Public Attribute Documentation

pZwTxQueue

SQueueNotifying* SApplicationHandles::pZwTxQueue

Notifying Queue object (contains FreeRTOS queue)

- Queue for ZW frames from application to Protocol for transmission

ZwRxQueue

QueueHandle_t SApplicationHandles::ZwRxQueue

FreeRTOS Queue handle - Queue for ZW frames forwarded from protocol to application.

pZwCommandQueue


```
SQueueNotifying* SApplicationHandles::pZwCommandQueue
```

Notifying Queue object (contains FreeRTOS queue)

- Queue for commands from Application to protocol

ZwCommandStatusQueue

```
QueueHandle_t SApplicationHandles::ZwCommandStatusQueue
```

FreeRTOS Queue handle - Queue for status replies from protocol to application (status on commands from application to protocol and Tx request from App to protocol)

pNetworkStatistics

```
const zpal_radio_network_stats_t* SApplicationHandles::pNetworkStatistics
```

Network statistics supplied by protocol.

pProtocolInfo

```
const SProtocolInfo* SApplicationHandles::pProtocolInfo
```

pNetworkInfo

```
const SNetworkInfo* SApplicationHandles::pNetworkInfo
```

pLongRangeInfo

```
const SLongRangeInfo* SApplicationHandles::pLongRangeInfo
```

Only used by Controller nodes.

Content is set to 0 on Slave nodes.

pRadioStatus

```
const SRadioStatus* SApplicationHandles::pRadioStatus
```

SRadioConfig_t

Public Attributes

int8_t	iListenBeforeTalkThreshold Db (negative) or EListenBeforeTalkThreshold_t.
zpal_tx_power_t	iTxPowerLevelMax Db (negative) or EtxPowerLevel_t.
zpal_tx_power_t	iTxPowerLevelAdjust Db (negative) or EtxPowerLevel_t.
int16_t	iTxPowerLevelMaxLR Maximum transmission power for Z-Wave LR.
zpal_radio_region_t	eRegion RF Region setting.
uint8_t	radio_debug_enable Enable radio PTI.
zpal_radio_application_t	radio_application Radio Application type.

Public Attribute Documentation

iListenBeforeTalkThreshold

```
int8_t SRadioConfig_t::iListenBeforeTalkThreshold
```

Db (negative) or EListenBeforeTalkThreshold_t.

iTxPowerLevelMax

```
zpal_tx_power_t SRadioConfig_t::iTxPowerLevelMax
```

Db (negative) or EtxPowerLevel_t.

iTxPowerLevelAdjust

```
zpal_tx_power_t SRadioConfig_t::iTxPowerLevelAdjust
```

Db (negative) or EtxPowerLevel_t.

iTxPowerLevelMaxLR

```
int16_t SRadioConfig_t::iTxPowerLevelMaxLR
```

Maximum transmission power for Z-Wave LR.

eRegion

zpal_radio_region_t SRadioConfig_t::eRegion

RF Region setting.

radio_debug_enable

uint8_t SRadioConfig_t::radio_debug_enable

Enable radio PTL.

radio_application

zpal_radio_application_t SRadioConfig_t::radio_application

Radio Application type.

SAppNodeInfo_t

Public Attributes

uint8_t DeviceOptionsMask

APPL_NODE_TYPE
E NodeType

Public Attribute Documentation

DeviceOptionsMask

uint8_t SAppNodeInfo_t::DeviceOptionsMask

NodeType

APPL_NODE_TYPE SAppNodeInfo_t::NodeType

SVirtualSlaveNodeInfo_t

Public Attributes

node_id_t	NodeId
bool	bListening
APPL_NODE_TYPE E	NodeType True if this node is always on air.
SCommandClassList_t	CommandClasses

Public Attribute Documentation

NodeId

`node_id_t SVirtualSlaveNodeInfo_t::NodeId`

bListening

`bool SVirtualSlaveNodeInfo_t::bListening`

NodeType

`APPL_NODE_TYPE SVirtualSlaveNodeInfo_t::NodeType`

True if this node is always on air.

CommandClasses

`SCommandClassList_t SVirtualSlaveNodeInfo_t::CommandClasses`

SVirtualSlaveNodeInfoTable_t

Public Attributes

```
uint8_t      iTableLength
const SVirtualSlaveNodeInfo_t** ppNodeInfo
```

Public Attribute Documentation

iTableLength

```
uint8_t SVirtualSlaveNodeInfoTable_t::iTableLength
```

ppNodeInfo

```
const SVirtualSlaveNodeInfo_t** SVirtualSlaveNodeInfoTable_t::ppNodeInfo
```

SProtocolConfig_t

This allows "nulling" a pointer while modifying a virtual slave node info Or changing pointer to point to a different one.

This struct content must be set up by application before enabling protocol (enabling radio) Direct content (the pointers) may not be changed runtime, but the data they point to can be edited by application run time

Public Attributes

const SVirtualSlaveNodeInfoTable_t *	pVirtualSlaveNodeInfoTable	
const uint8_t *	pSecureKeysRequested	NULL is acceptable if no virtual slave nodes.
const SAppNodeInfo_t *	pNodeInfo	values are ref SECURITY_KEY_S2_ACCESS_BIT, ref SECURITY_KEY_S2_AUTHENTICATED_BIT, ref SECURITY_KEY_S2_UNAUTHENTICATED_BIT, - consider making a bit field struct for it
const SRadioConfig_t *	pRadioConfig	

Public Attribute Documentation

pVirtualSlaveNodeInfoTable

```
const SVirtualSlaveNodeInfoTable_t* SProtocolConfig_t::pVirtualSlaveNodeInfoTable
```

pSecureKeysRequested

```
const uint8_t* SProtocolConfig_t::pSecureKeysRequested
```

NULL is acceptable if no virtual slave nodes.

pNodeInfo

```
const SAppNodeInfo_t* SProtocolConfig_t::pNodeInfo
```

values are ref SECURITY_KEY_S2_ACCESS_BIT, ref SECURITY_KEY_S2_AUTHENTICATED_BIT, ref SECURITY_KEY_S2_UNAUTHENTICATED_BIT, - consider making a bit field struct for it

pRadioConfig

```
const SRadioConfig_t* SProtocolConfig_t::pRadioConfig
```

STransmitFrameConfig

Public Attributes

void(*)	Handle	Will be returned with transmit status Allows application to recognize frames.
uint8_t	TransmitOptions	
uint8_t	iFrameLength	
uint8_t	aFrame	

Public Attribute Documentation

Handle

```
void(* STransmitFrameConfig::Handle) (uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus)
```

Will be returned with transmit status Allows application to recognize frames.

TransmitOptions

```
uint8_t STransmitFrameConfig::TransmitOptions
```

iFrameLength

```
uint8_t STransmitFrameConfig::iFrameLength
```

aFrame

```
uint8_t STransmitFrameConfig::aFrame[TX_BUFFER_SIZE]
```


STransmitProtocolFrameConfig

Public Attributes

void(*)	Handle
uint8_t	protocolMetadataLength
uint8_t	protocolMetadata
uint8_t	FrameLength
uint8_t	aFrame

Public Attribute Documentation

Handle

```
void(* STransmitProtocolFrameConfig::Handle) (uint8_t txStatus, TX_STATUS_TYPE *extendedTxStatus)
```

protocolMetadataLength

```
uint8_t STransmitProtocolFrameConfig::protocolMetadataLength
```

protocolMetadata

```
uint8_t STransmitProtocolFrameConfig::protocolMetadata[PROTOCOL_METADATA_LENGTH]
```

FrameLength

```
uint8_t STransmitProtocolFrameConfig::FrameLength
```

aFrame

```
uint8_t STransmitProtocolFrameConfig::aFrame[TX_BUFFER_SIZE]
```

SExploreInclusionRequest

Public Attributes

uint8_t [Reserved](#)

Public Attribute Documentation

Reserved

uint8_t SExploreInclusionRequest::Reserved

SExploreExclusionRequest

Public Attributes

uint8_t [Reserved](#)

Public Attribute Documentation

Reserved

uint8_t SExploreExclusionRequest::Reserved

SIncludedNodeInfo

Contains info related to sending an INIF.

Public Attributes

void(* [Handle](#)

Callback handle that Will be invoked with transmit status.

Public Attribute Documentation

Handle

```
void(* SIncludedNodeInfo::Handle) (uint8_t, TX_STATUS_TYPE *)
```

Callback handle that Will be invoked with transmit status.

STest

Public Attributes

void(*)	Handle
node_id_t	DestNodeId
uint8_t	PowerLevel

Public Attribute Documentation

Handle

```
void(*) STest::Handle) (uint8_t, TX_STATUS_TYPE *)
```

DestNodeId

```
node_id_t STest::DestNodeId
```

PowerLevel

```
uint8_t STest::PowerLevel
```

SSendData

Public Attributes

STransmitFrameConfig	FrameConfig
node_id_t	DestNodeId

Public Attribute Documentation

FrameConfig

STransmitFrameConfig SSendData::FrameConfig

DestNodeId

node_id_t SSendData::DestNodeId

SSendDataEx

Public Attributes

<code>STransmitFrameConfig</code>	<code>FrameConfig</code>
<code>node_id_t</code>	<code>DestNodeId</code>
<code>node_id_t</code>	<code>SourceNodeId</code>
<code>uint8_t</code>	<code>TransmitSecurityOptions</code>
<code>uint8_t</code>	<code>TransmitOptions2</code>
<code>enum SECURITY_KEY</code>	<code>eKeyType</code>

Public Attribute Documentation

FrameConfig

`STransmitFrameConfig SSendDataEx::FrameConfig`

DestNodeId

`node_id_t SSendDataEx::DestNodeId`

SourceNodeId

`node_id_t SSendDataEx::SourceNodeId`

TransmitSecurityOptions

`uint8_t SSendDataEx::TransmitSecurityOptions`

TransmitOptions2

`uint8_t SSendDataEx::TransmitOptions2`

eKeyType

```
enum SECURITY_KEY SSendDataEx::eKeyType
```


SSendDataBridge

Public Attributes

STransmitFrameConfig	FrameConfig
node_id_t	DestNodeId
node_id_t	SourceNodeId

Public Attribute Documentation

FrameConfig

STransmitFrameConfig SSendDataBridge::FrameConfig

DestNodeId

node_id_t SSendDataBridge::DestNodeId

SourceNodeId

node_id_t SSendDataBridge::SourceNodeId

SSendDataMulti

Public Attributes

STransmitFrameConfig	FrameConfig
NODE_MASK_TYPE	NodeMask

Public Attribute Documentation

FrameConfig

STransmitFrameConfig	SSendDataMulti::FrameConfig

NodeMask

NODE_MASK_TYPE	SSendDataMulti::NodeMask

SSendDataMultiEx

Public Attributes

STransmitFrameConfig	FrameConfig
uint8_t	SourceNodeId
uint8_t	GroupId
enum SECURITY_KEY	eKeyType

Public Attribute Documentation

FrameConfig

`STransmitFrameConfig SSendDataMultiEx::FrameConfig`

SourceNodeId

`uint8_t SSendDataMultiEx::SourceNodeId`

GroupId

`uint8_t SSendDataMultiEx::GroupId`

eKeyType

`enum SECURITY_KEY SSendDataMultiEx::eKeyType`

SSendDataMultiBridge

Public Attributes

STransmitFrameConfig	FrameConfig
uint8_t	NodeMask
node_id_t	SourceNodeId
bool	lr_nodeid_list

Public Attribute Documentation

FrameConfig

`STransmitFrameConfig SSendDataMultiBridge::FrameConfig`

NodeMask

`uint8_t SSendDataMultiBridge::NodeMask[MULTICAST_NODE_LIST_SIZE]`

SourceNodeId

`node_id_t SSendDataMultiBridge::SourceNodeId`

lr_nodeid_list

`bool SSendDataMultiBridge::lr_nodeid_list`

SSendProtocolData

Public Attributes

STransmitProtocol	FrameConfig
FrameConfig	
node_id_t	DestNodeID

Public Attribute Documentation

FrameConfig

STransmitProtocolFrameConfig SSendProtocolData::FrameConfig

DestNodeID

node_id_t SSendProtocolData::DestNodeID

SCommandNetworkManagement

Public Attributes

```
void(*      pHandle
uint8_t    mode
node_id_t  nodeID
```

Public Attribute Documentation

pHandle

```
void(* SCommandNetworkManagement::pHandle) (void)
```

mode

```
uint8_t SCommandNetworkManagement::mode
```

nodeID

```
node_id_t SCommandNetworkManagement::nodeID
```

SRequestNewRouteDestinations

Public Attributes

void(*)	Handle	
uint8_t	iDestinationCount	Will be returned with transmit status Allows application to recognize frames.
uint8_t	aNewDestinations	Number of new destinations Array containing new destinations.

Public Attribute Documentation

Handle

```
void(*) SRequestNewRouteDestinations::Handle) (void)
```

iDestinationCount

```
uint8_t SRequestNewRouteDestinations::iDestinationCount
```

Will be returned with transmit status Allows application to recognize frames.

aNewDestinations

```
uint8_t SRequestNewRouteDestinations::aNewDestinations[ZW_MAX_CACHED_RETURN_ROUTE_DESTINATIONS]
```

Number of new destinations Array containing new destinations.

SCommandNetworkManagementDSK

Public Attributes

void(*)	pHandle	
uint8_t	mode	Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.
uint8_t	dsk	

Public Attribute Documentation

pHandle

```
void(*) SCommandNetworkManagementDSK::pHandle) (void)
```

mode

```
uint8_t SCommandNetworkManagementDSK::mode
```

Placeholder for callback function Will be returned with transmit status Allows application to recognize frames.

dsk

```
uint8_t SCommandNetworkManagementDSK::dsk[8]
```


SZWaveTransmitStatus

Public Attributes

void(*)	Handle	
bool	blsTxFrameLegal	Placeholder for callback function.
uint8_t	TxStatus	False if frame rejected by protocol, can be due to content/configuration or due to timing (e.g. inclusion request when not in learn mode)
TX_STATUS_TYPE	ExtendedTxStatus	

Public Attribute Documentation

Handle

```
void(*) SZWaveTransmitStatus::Handle) (void)
```

blsTxFrameLegal

```
bool SZWaveTransmitStatus::blsTxFrameLegal
```

Placeholder for callback function.

TxStatus

```
uint8_t SZWaveTransmitStatus::TxStatus
```

False if frame rejected by protocol, can be due to content/configuration or due to timing (e.g. inclusion request when not in learn mode)

ExtendedTxStatus

```
TX_STATUS_TYPE SZWaveTransmitStatus::ExtendedTxStatus
```

SZWaveGenerateRandomStatus

Public Attributes

uint8_t iLength

uint8_t aRandomNumber

Public Attribute Documentation

iLength

```
uint8_t SZWaveGenerateRandomStatus::iLength
```

aRandomNumber

```
uint8_t SZWaveGenerateRandomStatus::aRandomNumber[32]
```

SZWaveNodeInfoStatus

Public Attributes

`node_id_t` `NodeId`

`t_ExtNodeInfo` `extNodeInfo`

Public Attribute Documentation

NodeId

`node_id_t` SZWaveNodeInfoStatus::NodeId

extNodeInfo

`t_ExtNodeInfo` SZWaveNodeInfoStatus::extNodeInfo

SZWaveLearnModeStatus

Public Attributes

[ELearnStatus](#) [Status](#)

Public Attribute Documentation

Status

ELearnStatus SZWaveLearnModeStatus::Status

SZWaveInvalidTxRequestStatus

Public Attributes

uint8_t InvalidTxRequest

Public Attribute Documentation

InvalidTxRequest

uint8_t SZWaveInvalidTxRequestStatus::InvalidTxRequest

SZWaveInvalidCommandStatus

Public Attributes

[EZwaveCommand](#) [InvalidCommand](#)
Type

Public Attribute Documentation

InvalidCommand

EZwaveCommandType SZWaveInvalidCommandStatus::InvalidCommand

SZWaveGeneric8bStatus

Public Attributes

uint8_t [result](#)

Public Attribute Documentation

result

uint8_t SZWaveGeneric8bStatus::result

SZWaveGenericBoolStatus

Public Attributes

bool [result](#)

Public Attribute Documentation

result

bool SZWaveGenericBoolStatus::result

SCommandRequestNodeNeighborUpdate

Public Attributes

void(*)[Handle](#)

[node_id_t](#)[NodeId](#)

Public Attribute Documentation

Handle

void(*) SCommandRequestNodeNeighborUpdate::Handle) (void)

NodeId

node_id_t SCommandRequestNodeNeighborUpdate::NodeId

SCommandRequestNodeTypeNeighborUpdate

Public Attributes

<code>void(*</code>	<code>Handle</code>
<code>E_SYSTEM_TYPE</code>	<code>NodeType</code>
<code>node_id_t</code>	<code>NodeId</code>

Public Attribute Documentation

Handle

```
void(* SCommandRequestNodeTypeNeighborUpdate::Handle) (void)
```

NodeType

```
E_SYSTEM_TYPE SCommandRequestNodeTypeNeighborUpdate::NodeType
```

NodeId

```
node_id_t SCommandRequestNodeTypeNeighborUpdate::NodeId
```

SZWaveGetRoutingInfoStatus

Public Attributes

uint8_t [RoutingInfo](#)

Public Attribute Documentation

RoutingInfo

uint8_t SZWaveGetRoutingInfoStatus::RoutingInfo[MAX_NODEMASK_LENGTH]

SCommandInitiateShutdown

Public Attributes

void(* [Handle](#))

Public Attribute Documentation

Handle

```
void(* SCommandInitiateShutdown::Handle) (void)
```

SCommandSetSecurityFlags

Public Attributes

<code>node_id_t</code>	<code>nodeID</code>
<code>bool</code>	<code>nodeS2Capable</code>
<code>bool</code>	<code>nodeS2Included</code>
<code>bool</code>	<code>nodeSecureIncluded</code>

Public Attribute Documentation

nodeID

```
node_id_t SCommandSetSecurityFlags::nodeID
```

nodeS2Capable

```
bool SCommandSetSecurityFlags::nodeS2Capable
```

nodeS2Included

```
bool SCommandSetSecurityFlags::nodeS2Included
```

nodeSecureIncluded

```
bool SCommandSetSecurityFlags::nodeSecureIncluded
```

SZWaveGetPriorityRouteStatus

Public Attributes

uint8_t	bAnyRouteFound
uint8_t	repeaters
uint8_t	routeSpeed

Public Attribute Documentation

bAnyRouteFound

```
uint8_t SZWaveGetPriorityRouteStatus::bAnyRouteFound
```

repeaters

```
uint8_t SZWaveGetPriorityRouteStatus::repeaters[MAX_REPEATERS]
```

routeSpeed

```
uint8_t SZWaveGetPriorityRouteStatus::routeSpeed
```

SZWaveSetPriorityRouteStatus

Public Attributes

uint8_t [bRouteUpdated](#)

Public Attribute Documentation

bRouteUpdated

uint8_t SZWaveSetPriorityRouteStatus::bRouteUpdated

SZWaveGetVirtualNodesStatus

Public Attributes

uint8_t [vNodesMask](#)

Public Attribute Documentation

vNodesMask

uint8_t SZWaveGetVirtualNodesStatus::vNodesMask[MAX_NODEMASK_LENGTH]

SZWaveAesEcbStatus

Public Attributes

uint8_t [outputData](#)

Public Attribute Documentation

outputData

uint8_t SZWaveAesEcbStatus::outputData[16]

SZWaveGetBackgroundRssiStatus

Public Attributes

int8_t rssi

Public Attribute Documentation

rssi

int8_t SZWaveGetBackgroundRssiStatus::rssi[ZPAL_RADIO_NUM_CHANNELS_LR_CH_CFG_1_2]

SZWaveNetworkManagementStatus

Public Attributes

void * [pHandle](#)

uint8_t [statusInfo](#)

Public Attribute Documentation

pHandle

```
void* SZWaveNetworkManagementStatus::pHandle
```

statusInfo

```
uint8_t SZWaveNetworkManagementStatus::statusInfo[5+NODEPARM_MAX]
```

SNvmBackupRestoreStatus

Public Attributes

uint8_t [status](#)

Public Attribute Documentation

status

uint8_t SNvmBackupRestoreStatus::status

SCommandGetNodeNLSSStateStatus

Public Attributes

uint8_t	nlsSupport
uint8_t	nlsState

Public Attribute Documentation

nlsSupport

uint8_t SCommandGetNodeNLSSStateStatus::nlsSupport

nlsState

uint8_t SCommandGetNodeNLSSStateStatus::nlsState

SCommandEnableNodeNLSStatus

Public Attributes

uint8_t status

Public Attribute Documentation

status

uint8_t SCommandEnableNodeNLSStatus::status

SZWaveGetIncludedNodes

Public Attributes

NODE_MASK_TY
PE node_id_list

Public Attribute Documentation

node_id_list

NODE_MASK_TYPE SZWaveGetIncludedNodes::node_id_list

SZWaveGetIncludedNodesLR

Public Attributes

LR_NODE_MASK_ node_id_list
TYPE

Public Attribute Documentation

node_id_list

LR_NODE_MASK_TYPE SZWaveGetIncludedNodesLR::node_id_list

SZWaveGetIncludedNodesNLS

Public Attributes

uint8_t	node_id_list
bool	more_nodes
uint8_t	output_length

Public Attribute Documentation

node_id_list

```
uint8_t SZWaveGetIncludedNodesNLS::node_id_list[GET-NLS-NODES-LIST-LENGTH-MAX]
```

more_nodes

```
bool SZWaveGetIncludedNodesNLS::more_nodes
```

output_length

```
uint8_t SZWaveGetIncludedNodesNLS::output_length
```

SZWaveTxPowerMaxSupported

Public Attributes

[zpal_tx_power_t](#) [tx_power_max_supported](#)

Public Attribute Documentation

tx_power_max_supported

`zpal_tx_power_t SZWaveTxPowerMaxSupported::tx_power_max_supported`

SCommandTransferProtocolCC

Public Attributes

<code>node_id_t</code>	<code>srcNodeId</code>
<code>security_key_t</code>	<code>decryptionKey</code>
<code>uint8_t</code>	<code>payloadLength</code>
<code>uint8_t</code>	<code>payload</code>

Public Attribute Documentation

srcNodeId

```
node_id_t SCommandTransferProtocolCC::srcNodeId
```

decryptionKey

```
security_key_t SCommandTransferProtocolCC::decryptionKey
```

payloadLength

```
uint8_t SCommandTransferProtocolCC::payloadLength
```

payload

```
uint8_t SCommandTransferProtocolCC::payload[ZW_MAX_PAYLOAD_SIZE]
```

SReceiveNodeUpdate

The Z Wave protocol MAY notify an application by sending [SReceiveNodeUpdate](#) when a Node Information Frame has been received.

The Z Wave protocol MAY refrain from sending the information if the protocol is currently expecting node information.

A controller application MAY use the information provided by [SReceiveNodeUpdate](#) to update local data structures.

The Z Wave protocol MUST notify a controller application by calling [SReceiveNodeUpdate](#) when a new node has been added or deleted from the controller through the network management features.

The Z Wave protocol MUST pass [SReceiveNodeUpdate](#) to application in response to [SNodeInfoRequest](#) being passed to protocol by the controller application. The Z Wave protocol MAY notify a controller application by sending [SReceiveNodeUpdate](#) when a Node Information Frame has been received. The Z Wave protocol MAY refrain from sending the information if the protocol is currently expecting a Node Information frame.

[SReceiveNodeUpdate](#) MUST be sent in a controller node operating as SIS each time a node is added or deleted by the primary controller. [SReceiveNodeUpdate](#) MUST be sent in a controller node operating as SIS each time a node is added/deleted by an inclusion controller.

A controller application MAY send a ZW_RequestNetWorkUpdate command to a SIS or SIS node. In response, the SIS MUST return update information for each node change since the last update handled by the requesting controller node. The application of the requesting controller node MAY receive multiple instances of [SReceiveNodeUpdate](#) in response to application passing [SNetworkUpdateRequest](#) to protocol.

The Z Wave protocol MUST NOT send [SReceiveNodeUpdate](#) in a controller node acting as primary controller or inclusion controller when a node is added or deleted.

Any controller application MUST implement this function.

serialapi{ZW->HOST: REQ | 0x49 | bStatus | bNodeID | bLen | basic | generic | specific | commandclasses[]}

[SReceiveNodeUpdate](#) via the Serial API also have the possibility for receiving the status UPDATE_STATE_NODE_INFO_REQ_FAILED, which means that a node did not acknowledge a [SNodeInfoRequest](#) .

Public Attributes

uint8_t	Status
node_id_t	NodeId
uint8_t	iLength
uint8_t	aPayload

Public Attribute Documentation

Status

uint8_t SReceiveNodeUpdate::Status

NodeId

node_id_t SReceiveNodeUpdate::NodeId

iLength

uint8_t SReceiveNodeUpdate::iLength

aPayload

uint8_t SReceiveNodeUpdate::aPayload[MAX_NODE_INFO_LENGTH]

SReceiveSecurityEvent

Public Attributes

e_application_security_event_t	Event
uint8_t	iLength
uint8_t	aEventData

Public Attribute Documentation

Event

e_application_security_event_t SReceiveSecurityEvent::Event

iLength

uint8_t SReceiveSecurityEvent::iLength

aEventData

uint8_t SReceiveSecurityEvent::aEventData[48]

UTransmitParameters

Transmit Parameters Union used by [SZwaveTransmitPackage](#).

Public Attributes

SExploreInclusionRequest	ExploreInclusionRequest
SExploreExclusionRequest	ExploreExclusionRequest
SNetworkUpdateRequest	NetworkUpdateRequest
SNodeInfoRequest	NodeInfoRequest
SNodeInfo	NodeInfo
SIncludedNodeInfo	IncludedNodeInfo
STest	Test
SSendData	SendData
SSendDataEx	SendDataEx
SSendDataBridge	SendDataBridge
SSendDataMulti	SendDataMulti
SSendDataMultiEx	SendDataMultiEx
SSendDataMultiBridge	SendDataMultiBridge
SSendProtocolData	SendProtocolData
SSetSucNodeId	SetSucNodeId
SSendSucNodeId	SendSucNodeId
SAssignReturnRoute	AssignReturnRoute
SDeleteReturnRoute	DeleteReturnRoute
SSendSlaveNodeInformation	SendSlaveNodeInformation
SRequestNewRouteDestinations	RequestNewRouteDestinations

Public Attribute Documentation

SExploreInclusionRequest UTransmitParameters::ExploreInclusionRequest

ExploreExclusionRequest

SExploreExclusionRequest UTransmitParameters::ExploreExclusionRequest

NetworkUpdateRequest

SNetworkUpdateRequest UTransmitParameters::NetworkUpdateRequest

NodeInfoRequest

SNodeInfoRequest UTransmitParameters::NodeInfoRequest

NodeInfo

SNodeInfo UTransmitParameters::NodeInfo

IncludedNodeInfo

SIncludedNodeInfo UTransmitParameters::IncludedNodeInfo

Test

STest UTransmitParameters::Test

SendData

SSendData UTransmitParameters::SendData

SendDataEx

SSendDataEx UTransmitParameters::SendDataEx

SendDataBridge

SSendDataBridge UTransmitParameters::SendDataBridge

SendDataMulti

SSendDataMulti UTransmitParameters::SendDataMulti

SendDataMultiEx

SSendDataMultiEx UTransmitParameters::SendDataMultiEx

SendDataMultiBridge

SSendDataMultiBridge UTransmitParameters::SendDataMultiBridge

SendProtocolData

SSendProtocolData UTransmitParameters::SendProtocolData

SetSucNodeId

SSetSucNodeId UTransmitParameters::SetSucNodeId

SendSucNodeId

SSendSucNodeId UTransmitParameters::SendSucNodeId

AssignReturnRoute

SAssignReturnRoute UTransmitParameters::AssignReturnRoute

DeleteReturnRoute

SDeleteReturnRoute UTransmitParameters::DeleteReturnRoute

SendSlaveNodeInformation

SSendSlaveNodeInformation UTransmitParameters::SendSlaveNodeInformation

RequestNewRouteDestinations

SRequestNewRouteDestinations UTransmitParameters::RequestNewRouteDestinations

UCommandStatus

Command Status Union used by [SZWaveCommandStatusPackage](#).

Public Attributes

SZWaveTransmitStatus	TxStatus
SZWaveGenerateRandomStatus	GenerateRandomStatus
SZWaveNodeInfoStatus	NodeInfoStatus
SZWaveLearnModeStatus	LearnModeStatus
SZWaveInvalidTxRequestStatus	InvalidTxRequestStatus
SZWaveInvalidCommandStatus	InvalidCommandStatus
SZWaveGeneric8bStatus	SetRFReceiveModeStatus
SZWaveGeneric8bStatus	IsNodeWithinDirectRange
SZWaveGeneric8bStatus	GetNeighborCountStatus
SZWaveGeneric8bStatus	AreNodesNeighborStatus
SZWaveGeneric8bStatus	IsFailedNodeIDStatus
SZWaveGeneric8bStatus	ReplaceFailedNodeStatus
SZWaveGetRoutingInfoStatus	GetRoutingInfoStatus
SZWaveGeneric8bStatus	SetRoutingInfoStatus
SZWaveGeneric8bStatus	StoreNodeInfoStatus
SZWaveGetPriorityRouteStatus	GetPriorityRouteStatus
SZWaveSetPriorityRouteStatus	SetPriorityRouteStatus
SZWaveGeneric8bStatus	SetSlaveLearnModeStatus
SZWaveGeneric8bStatus	IsVirtualNodeStatus

SZWaveGetVirtualNodesStatus	GetVirtualNodesStatus
SZWaveGeneric8bStatus	GetControllerCapabilitiesStatus
SZWaveGeneric8bStatus	IsPrimaryCtrlStatus
SZWaveAesEcbStatus	AesEcbStatus
SZWaveGetBackgroundRssiStatus	GetBackgroundRssiStatus
SZWaveNetworkManagementStatus	NetworkManagementStatus
SZWaveGeneric8bStatus	FailedNodeIDStatus
SNvmBackupRestoreStatus	NvmBackupRestoreStatus
SZWaveGenericBooStatus	SetPowerDownCallbackStatus
SZWaveGetIncludedNodes	GetIncludedNodes
SZWaveGetIncludedNodesLR	GetIncludedNodesLR
SZWaveGetIncludedNodesNLS	GetIncludedNodesNLS
SZWaveGeneric8bStatus	RequestNodeNeighborUpdateStatus
SZWaveGeneric8bStatus	RequestNodeTypeNeighborUpdateStatus
SZWaveGeneric8bStatus	InitiateShutdownStatus
SZWaveGeneric8bStatus	GetLRChannel
SZWaveGeneric8bStatus	SetLRChannel
SZWaveGeneric8bStatus	GetPTIconfig
SZWaveGenericBooStatus	SetTxAttenuation
SZWaveTxPowerMaxSupported	GetTxPowerMaximumSupported
SCommandGetNodeNLSStateStatus	GetNodeNlsStateStatus

[SCommandEnable](#) [EnableNodeNlsStatus](#)
[NodeNLSStatus](#)

Public Attribute Documentation

TxStatus

SZWaveTransmitStatus UCommandStatus::TxStatus

GenerateRandomStatus

SZWaveGenerateRandomStatus UCommandStatus::GenerateRandomStatus

NodeInfoStatus

SZWaveNodeInfoStatus UCommandStatus::NodeInfoStatus

LearnModeStatus

SZWaveLearnModeStatus UCommandStatus::LearnModeStatus

InvalidTxRequestStatus

SZWaveInvalidTxRequestStatus UCommandStatus::InvalidTxRequestStatus

InvalidCommandStatus

SZWaveInvalidCommandStatus UCommandStatus::InvalidCommandStatus

SetRFReceiveModeStatus

SZWaveGeneric8bStatus UCommandStatus::SetRFReceiveModeStatus

IsNodeWithinDirectRange

SZWaveGeneric8bStatus UCommandStatus::IsNodeWithinDirectRange

GetNeighborCountStatus

SZWaveGeneric8bStatus UCommandStatus::GetNeighborCountStatus

AreNodesNeighborStatus

SZWaveGeneric8bStatus UCommandStatus::AreNodesNeighborStatus

IsFailedNodeIDStatus

SZWaveGeneric8bStatus UCommandStatus::IsFailedNodeIDStatus

ReplaceFailedNodeStatus

SZWaveGeneric8bStatus UCommandStatus::ReplaceFailedNodeStatus

GetRoutingInfoStatus

SZWaveGetRoutingInfoStatus UCommandStatus::GetRoutingInfoStatus

SetRoutingInfoStatus

SZWaveGeneric8bStatus UCommandStatus::SetRoutingInfoStatus

StoreNodeInfoStatus

SZWaveGeneric8bStatus UCommandStatus::StoreNodeInfoStatus

GetPriorityRouteStatus

SZWaveGetPriorityRouteStatus UCommandStatus::GetPriorityRouteStatus

SetPriorityRouteStatus

SZWaveSetPriorityRouteStatus UCommandStatus::SetPriorityRouteStatus

SetSlaveLearnModeStatus

SZWaveGeneric8bStatus UCommandStatus::SetSlaveLearnModeStatus

IsVirtualNodeStatus

SZWaveGeneric8bStatus UCommandStatus::IsVirtualNodeStatus

GetVirtualNodesStatus

SZWaveGetVirtualNodesStatus UCommandStatus::GetVirtualNodesStatus

GetControllerCapabilitiesStatus

SZWaveGeneric8bStatus UCommandStatus::GetControllerCapabilitiesStatus

IsPrimaryCtrlStatus

SZWaveGeneric8bStatus UCommandStatus::IsPrimaryCtrlStatus

AesEcbStatus

SZWaveAesEcbStatus UCommandStatus::AesEcbStatus

GetBackgroundRssiStatus

SZWaveGetBackgroundRssiStatus UCommandStatus::GetBackgroundRssiStatus

NetworkManagementStatus

SZWaveNetworkManagementStatus UCommandStatus::NetworkManagementStatus

FailedNodeIDStatus

SZWaveGeneric8bStatus UCommandStatus::FailedNodeIDStatus

NvmBackupRestoreStatus

SNvmBackupRestoreStatus UCommandStatus::NvmBackupRestoreStatus

SetPowerDownCallbackStatus

SZWaveGenericBoolStatus UCommandStatus::SetPowerDownCallbackStatus

GetIncludedNodes

SZWaveGetIncludedNodes UCommandStatus::GetIncludedNodes

GetIncludedNodesLR

SZWaveGetIncludedNodesLR UCommandStatus::GetIncludedNodesLR

GetIncludedNodesNLS

SZWaveGetIncludedNodesNLS UCommandStatus::GetIncludedNodesNLS

RequestNodeNeighborUpdateStatus

SZWaveGeneric8bStatus UCommandStatus::RequestNodeNeighborUpdateStatus

RequestNodeTypeNeighborUpdateStatus

SZWaveGeneric8bStatus UCommandStatus::RequestNodeTypeNeighborUpdateStatus

InitiateShutdownStatus

SZWaveGeneric8bStatus UCommandStatus::InitiateShutdownStatus

GetLRChannel

SZWaveGeneric8bStatus UCommandStatus::GetLRChannel

SetLRChannel

SZWaveGeneric8bStatus UCommandStatus::SetLRChannel

GetPTIconfig

SZWaveGeneric8bStatus UCommandStatus::GetPTIconfig

SetTxAttenuation

SZWaveGenericBoolStatus UCommandStatus::SetTxAttenuation

GetTxPowerMaximumSupported

SZWaveTxPowerMaxSupported UCommandStatus::GetTxPowerMaximumSupported

GetNodeNlsStateStatus

SCommandGetNodeNLSSStateStatus UCommandStatus::GetNodeNlsStateStatus

EnableNodeNlsStatus

SCommandEnableNodeNLSSStatus UCommandStatus::EnableNodeNlsStatus

UReceiveCmdPayload

Public Attributes

ZW_APPLICATION _TX_BUFFER	rxBuffer
uint8_t	padding

Public Attribute Documentation

rxBuffer

ZW_APPLICATION_TX_BUFFER UReceiveCmdPayload::rxBuffer

padding

uint8_t UReceiveCmdPayload::padding[ZW_MAX_PAYLOAD_SIZE]

UCommandParameters

Public Attributes

SCommandGenerateRandom	GenerateRandom
SCommandNodeInfo	NodeInfo
SCommandClearNetworkStatistics	ClearNetworkStatistics
SCommandSetLearnMode	SetLearnMode
SCommandSetSmartStartLearnMode	SetSmartStartLearnMode
SCommandSetPromiscuousMode	SetPromiscuousMode
SCommandSetRfReceiveMode	SetRfReceiveMode
SCommandGeneric8bParameter	IsNodeWithinDirectRange
SCommandGeneric8bParameter	GetNeighborCount
SCommandAreNodesNeighbours	AreNodesNeighbours
SCommandNetworkManagement	IsFailedNodeID
SCommandNetworkManagement	SetSlaveLearnMode
SCommandGetRoutingInfo	GetRoutingInfo
SCommandSetRoutingInfo	SetRoutingInfo
SCommandStoreNodeInfo	StoreNodeInfo
SCommandStoreHomeID	StoreHomeID
SCommandGeneric8bParameter	LockRouteResponse
SCommandGetPriorityRoute	GetPriorityRoute
SCommandSetPriorityRoute	SetPriorityRoute

SCommandGeneric 16bParameter	IsVirtualNode
SCommandGeneric 8bParameter	SetRoutingMax
SCommandNetwork Management	NetworkManagement
SCommandNetwork ManagementDSK	NetworkManagementDSK
SCommandFailed NodeIDCmd	FailedNodeIDCmd
SCommandAesEc b	AesEcb
SCommandPMSet PowerDownCallb ack	PMSetPowerDownCallback
SCommandGeneric 8bParameter	UpdateCtrlNodeInformation
SCommandSetLB TThreshold	SetLBTThreshold
SCommandSetRf PowerLevel	SetRfPowerLevel
SCommandSetMa xInclReqInterval	SetMaxInclReqInterval
SCommandNvmB ackupRestore	NvmBackupRestore
SCommandSetSe curityKeys	SetSecurityKeys
SCommandReque stNodeNeighborU pdate	RequestNodeNeighborUpdate
SCommandReque stNodeTypeNeigh borUpdate	RequestNodeTypeNeighborUpdate
SCommandInitiate Shutdown	InitiateShutdown
SCommandGeneric 8bParameter	SetLRChannel
SCommandGeneric 8bParameter	SetLRVirtualNodeIDs
SCommandGeneric 8bParameter	SetTxAttenuation
SCommandSetSe curityFlags	SetSecurityFlags

SCommandTransferProtocolCC
TransferProtocolCC

SCommandEnableNodeNLS
EnableNodeNLS

SCommandGetNodeNLSState
GetNodeNLSState

SCommandSendProtocolDataCb
SendProtocolDataCb

SCommandGetNLSNodes
GetNLSNodes

Public Attribute Documentation

GenerateRandom

SCommandGenerateRandom UCommandParameters::GenerateRandom

NodeInfo

SCommandNodeInfo UCommandParameters::NodeInfo

ClearNetworkStatistics

SCommandClearNetworkStatistics UCommandParameters::ClearNetworkStatistics

SetLearnMode

SCommandSetLearnMode UCommandParameters::SetLearnMode

SetSmartStartLearnMode

SCommandSetSmartStartLearnMode UCommandParameters::SetSmartStartLearnMode

SetPromiscuousMode

SCommandSetPromiscuousMode UCommandParameters::SetPromiscuousMode

SetRfReceiveMode

SCommandSetRfReceiveMode UCommandParameters::SetRfReceiveMode

IsNodeWithinDirectRange

SCommandGeniric8bParameter UCommandParameters::IsNodeWithinDirectRange

GetNeighborCount

SCommandGeniric8bParameter UCommandParameters::GetNeighborCount

AreNodesNeighbours

SCommandAreNodesNeighbours UCommandParameters::AreNodesNeighbours

IsFailedNodeID

SCommandNetworkManagement UCommandParameters::IsFailedNodeID

SetSlaveLearnMode

SCommandNetworkManagement UCommandParameters::SetSlaveLearnMode

GetRoutingInfo

SCommandGetRoutingInfo UCommandParameters::GetRoutingInfo

SetRoutingInfo

SCommandSetRoutingInfo UCommandParameters::SetRoutingInfo

StoreNodeInfo

SCommandStoreNodeInfo UCommandParameters::StoreNodeInfo

StoreHomeID

SCommandStoreHomeID UCommandParameters::StoreHomeID

LockRouteResponse

SCommandGeniric8bParameter UCommandParameters::LockRouteResponse

GetPriorityRoute

SCommandGetPriorityRoute UCommandParameters::GetPriorityRoute

SetPriorityRoute

SCommandSetPriorityRoute UCommandParameters::SetPriorityRoute

IsVirtualNode

SCommandGeniric16bParameter UCommandParameters::IsVirtualNode

SetRoutingMax

SCommandGeniric8bParameter UCommandParameters::SetRoutingMax

NetworkManagement

SCommandNetworkManagement UCommandParameters::NetworkManagement

NetworkManagementDSK

SCommandNetworkManagementDSK UCommandParameters::NetworkManagementDSK

FailedNodeIDCmd

SCommandFailedNodeIDCmd UCommandParameters::FailedNodeIDCmd

AesEcb

SCommandAesEcb UCommandParameters::AesEcb

PMSetPowerDownCallback

SCommandPMSetPowerDownCallback UCommandParameters::PMSetPowerDownCallback

UpdateCtrlNodeInformation

SCommandGeneric8bParameter UCommandParameters::UpdateCtrlNodeInformation

SetLBTThreshold

SCommandSetLBTThreshold UCommandParameters::SetLBTThreshold

SetRfPowerLevel

SCommandSetRfPowerLevel UCommandParameters::SetRfPowerLevel

SetMaxInclReqInterval

SCommandSetMaxInclReqInterval UCommandParameters::SetMaxInclReqInterval

NvmBackupRestore

SCommandNvmBackupRestore UCommandParameters::NvmBackupRestore

SetSecurityKeys

SCommandSetSecurityKeys UCommandParameters::SetSecurityKeys

RequestNodeNeighborUpdate

SCommandRequestNodeNeighborUpdate UCommandParameters::RequestNodeNeighborUpdate

RequestNodeTypeNeighborUpdate

SCommandRequestNodeTypeNeighborUpdate UCommandParameters::RequestNodeTypeNeighborUpdate

InitiateShutdown

SCommandInitiateShutdown UCommandParameters::InitiateShutdown

SetLRChannel

SCommandGeneric8bParameter UCommandParameters::SetLRChannel

SetLRVirtualNodeIDs

SCommandGeneric8bParameter UCommandParameters::SetLRVirtualNodeIDs

SetTxAttenuation

SCommandGeneric8bParameter UCommandParameters::SetTxAttenuation

SetSecurityFlags

SCommandSetSecurityFlags UCommandParameters::SetSecurityFlags

TransferProtocolCC

SCommandTransferProtocolCC UCommandParameters::TransferProtocolCC

EnableNodeNIs

SCommandEnableNodeNLS UCommandParameters::EnableNodeNIs

GetNodeNIsState

SCommandGetNodeNLSState UCommandParameters::GetNodeNIsState

SendProtocolDataCb

SCommandSendProtocolDataCb UCommandParameters::SendProtocolDataCb

GetNLSNodes

SCommandGetNLSNodes UCommandParameters::GetNLSNodes

SZwaveTransmitPackage

Public Attributes

EZwaveTransmitType	eTransmitType
UTransmitParameters	uTransmitParams

Public Attribute Documentation

eTransmitType

EZwaveTransmitType	SZwaveTransmitPackage::eTransmitType

uTransmitParams

UTransmitParameters	SZwaveTransmitPackage::uTransmitParams

SZwaveCommandStatusPackage

Z-Wave Command Status Package.

Public Attributes

- EZwaveCommand
StatusType
- eStatusType
- UCommandStatus
- Content

Public Attribute Documentation

eStatusType

EZwaveCommandStatusType SZwaveCommandStatusPackage::eStatusType

Content

UCommandStatus SZwaveCommandStatusPackage::Content

SZwaveCommandPackage

Public Attributes

EZwaveCommandType	eCommandType
UCommandParameters	uCommandParams

Public Attribute Documentation

eCommandType

EZwaveCommandType SZwaveCommandPackage::eCommandType

uCommandParams

UCommandParameters SZwaveCommandPackage::uCommandParams

SReceiveSingle

Public Attributes

UReceiveCmdPayload	Payload
uint8_t	iLength
RECEIVE_OPTION_S_TYPE	RxOptions

Public Attribute Documentation

Payload

UReceiveCmdPayload SReceiveSingle::Payload

iLength

uint8_t SReceiveSingle::iLength

RxOptions

RECEIVE_OPTIONS_TYPE SReceiveSingle::RxOptions

SReceiveMulti

Public Attributes

<code>NODE_MASK_TYPE</code>	<code>NodeMask</code>
<code>UReceiveCmdPayload</code>	<code>Payload</code>
<code>uint8_t</code>	<code>iCommandLength</code>
<code>RECEIVE_OPTIONS_TYPE</code>	<code>RxOptions</code>

Public Attribute Documentation

NodeMask

<code>NODE_MASK_TYPE</code> SReceiveMulti::NodeMask

Payload

<code>UReceiveCmdPayload</code> SReceiveMulti::Payload

iCommandLength

<code>uint8_t</code> SReceiveMulti::iCommandLength

RxOptions

<code>RECEIVE_OPTIONS_TYPE</code> SReceiveMulti::RxOptions

SProtocolTxOptions

Public Attributes

uint8_t	rfSpeed
uint8_t	txPower
TxOptions_t	txOptions

Public Attribute Documentation

rfSpeed

uint8_t SProtocolTxOptions::rfSpeed

txPower

uint8_t SProtocolTxOptions::txPower

txOptions

TxOptions_t SProtocolTxOptions::txOptions

SRequestEncryption

Public Attributes

node_id_t	destNodeID
uint8_t	payloadLength
UReceiveCmdPayload	Payload
uint8_t	protocolMetadataLength
uint8_t	protocolMetadata
uint8_t	useSupervision
uint8_t	reserved

Public Attribute Documentation

destNodeID

```
node_id_t SRequestEncryption::destNodeID
```

payloadLength

```
uint8_t SRequestEncryption::payloadLength
```

Payload

```
UReceiveCmdPayload SRequestEncryption::Payload
```

protocolMetadataLength

```
uint8_t SRequestEncryption::protocolMetadataLength
```

protocolMetadata

```
uint8_t SRequestEncryption::protocolMetadata[PROTOCOL_METADATA_LENGTH]
```

useSupervision

uint8_t SRequestEncryption::useSupervision

reserved

uint8_t SRequestEncryption::reserved

UReceiveParameters

Public Attributes

SReceiveSingle	Rx
SReceiveMulti	RxMulti
SReceiveNodeUpdate	RxNodeUpdate
SReceiveSecurityEvent	RxSecurityEvent
SRequestEncryption	RequestEncryption

Public Attribute Documentation

Rx

`SReceiveSingle UReceiveParameters::Rx`

RxMulti

`SReceiveMulti UReceiveParameters::RxMulti`

RxNodeUpdate

`SReceiveNodeUpdate UReceiveParameters::RxNodeUpdate`

RxSecurityEvent

`SReceiveSecurityEvent UReceiveParameters::RxSecurityEvent`

RequestEncryption

`SRequestEncryption UReceiveParameters::RequestEncryption`

SZwaveReceivePackage

Public Attributes

EZwaveReceiveType	eReceiveType
UReceiveParameters	uReceiveParams

Public Attribute Documentation

eReceiveType

EZwaveReceiveType SZwaveReceivePackage::eReceiveType

uReceiveParams

UReceiveParameters SZwaveReceivePackage::uReceiveParams

Z-Wave Basis API

Z-Wave Basis API

This section defines functions that are implemented in all Z Wave nodes.

Modules

[_APPL_NODE_TYPE_](#)

[_NODE_TYPE_](#)

[_NODEINFO_](#)

[RSSI_LEVELS](#)

[Application Node Info](#)

Enumerations

```
enum \_APPLICATION\_STATUS\_ {
    APPLICATION_RUNNING
    APPLICATION_POWER_DOWN
    APPLICATION_TEST
}
Return values of ApplicationInit.

enum \_E\_NETWORK\_LEARN\_MODE\_ACTION\_ {
    E_NETWORK_LEARN_MODE_DISABLE = 0
    E_NETWORK_LEARN_MODE_INCLUSION = 1
    E_NETWORK_LEARN_MODE_EXCLUSION = 2
    E_NETWORK_LEARN_MODE_EXCLUSION_NWE = 3
    E_NETWORK_LEARN_MODE_INCLUSION_SMARTSTART = 4
    E_NETWORK_LEARN_MODE_COUNT
}
Network modes that can be set by ZAF\_setNetworkLearnMode.

enum \_E\_SYSTEM\_TYPE\_ {
    E_SYSTEM_TYPE_NON_LISTENING = APPLICATION_NODEINFO_NOT_LISTENING
    E_SYSTEM_TYPE_LISTENING = APPLICATION_NODEINFO_LISTENING
    E_SYSTEM_TYPE_FLIRS = 0x02
}
```

Typedefs

```
typedef enum ZW\_APPLICATION\_STATUS
\_APPLICATION\_STATUS\_
Return values of ApplicationInit.

typedef struct APPL\_NODE\_TYPE
\_APPL\_NODE\_TYPE\_
Node type structure used in ApplicationNodeInformation.

typedef struct NODE\_TYPE
\_NODE\_TYPE\_
```

<code>typedef struct</code> <code>_NODEINFO_</code>	NODEINFO Node info stored within the non-volatile memory This are the first (protocol part) payload bytes from the Node Infomation frame.
<code>typedef enum</code> <code>_E_NETWORK_LEARN_MODE_ACTION_</code> <code>ARN_MODE_ACTION_</code> <code>ON_</code>	E_NETWORK_LEARN_MODE_ACTION Network modes that can be set by ZAF_setNetworkLearnMode .
<code>typedef enum</code> <code>_E_SYSTEM_TYPE_</code> <code>E_</code>	E_SYSTEM_TYPE

Functions

<code>ZW_APPLICATION</code> <code>_STATUS</code>	ApplicationInit (zpal_reset_reason_t eResetReason) ApplicationInit () MUST be defined by the application.
<code>void</code>	sl_zwave_protocol_startup (void) Initialize the Z-Wave protocol stack.
<code>void</code>	sl_zwave_platform_startup (void) Early startup of Z-Wave platform features.
<code>uint16_t</code>	ZW_GetProtocolBuildNumber (void) Gets Z-Wave Protocol Build number.
<code>const uint8_t *</code>	ZW_GetProtocolGitHash (void) Gets Z-Wave Protocol git hash.

Macros

<code>#define</code>	<code>SYSTEM_TYPE_LISTENING_MASK</code> E_SYSTEM_TYPE_LISTENING
<code>#define</code>	<code>SYSTEM_TYPE_FLIRS_MASK</code> E_SYSTEM_TYPE_FLIRS

Enumeration Documentation

`_APPLICATION_STATUS_`

<code>_APPLICATION_STATUS_</code>

Return values of [ApplicationInit](#).

Enumerator	
<code>APPLICATION_RUNNING</code>	Application started normally.
<code>APPLICATION_POWER_DOWN</code>	Application started in special mode: Scheduler started, but Z-Wave protocol task did not.
<code>APPLICATION_TEST</code>	Test only, not to be used in production.

`_E_NETWORK_LEARN_MODE_ACTION_`

<code>_E_NETWORK_LEARN_MODE_ACTION_</code>

Network modes that can be set by [ZAF_setNetworkLearnMode](#).

Enumerator

E_NETWORK_LEARN_MODE_DISABLE	Disable learn process.
E_NETWORK_LEARN_MODE_INCLUSION	Enable the learn process to do an inclusion.
E_NETWORK_LEARN_MODE_EXCLUSION	Enable the learn process to do an exclusion.
E_NETWORK_LEARN_MODE_EXCLUSION_NWE	Enable the learn process to do an network wide exclusion.
E_NETWORK_LEARN_MODE_INCLUSION_SMARTSTART	Enable the learn process to do an SMARTSTART inclusion.
E_NETWORK_LEARN_MODE_COUNT	

_E_SYSTEM_TYPE_

_E_SYSTEM_TYPE_

DeprecatedUse [Application Node Info](#) instead

Enumerator

E_SYSTEM_TYPE_NON_LISTENING	
E_SYSTEM_TYPE_LISTENING	
E_SYSTEM_TYPE_FLIRS	

Typedef Documentation

ZW_APPLICATION_STATUS

typedef enum _APPLICATION_STATUS_ ZW_APPLICATION_STATUS

Return values of [ApplicationInit](#).

APPL_NODE_TYPE

typedef struct _APPL_NODE_TYPE_ APPL_NODE_TYPE

Node type structure used in ApplicationNodeInformation.

NODE_TYPE

typedef struct _NODE_TYPE_ NODE_TYPE

NODEINFO

typedef struct _NODEINFO_ NODEINFO

Node info stored within the non-volatile memory This are the first (protocol part) payload bytes from the Node Infomation frame.

E_NETWORK_LEARN_MODE_ACTION

```
typedef enum _E_NETWORK_LEARN_MODE_ACTION_ E_NETWORK_LEARN_MODE_ACTION
```

Network modes that can be set by [ZAF_setNetworkLearnMode](#).

E_SYSTEM_TYPE

```
typedef enum _E_SYSTEM_TYPE_ E_SYSTEM_TYPE
```

DeprecatedUse [Application Node Info](#) instead

DeprecatedUse [APPLICATION_NODEINFO_LISTENING](#) instead

Function Documentation

ApplicationInit

```
ZW_APPLICATION_STATUS ApplicationInit (zpal_reset_reason_t eResetReason)
```

[ApplicationInit\(\)](#) MUST be defined by the application.

Parameters

Type	Direction	Argument Name	Description
zpal_reset_reason_t	[in]	eResetReason	The reason that triggered a reset/boot of the firmware. For instance, external pin, wake up timer, power on, etc. Please see zpal_reset_reason_t .

This function will be invoked by the Z-Wave stack during boot.

[ApplicationInit\(\)](#) is invoked before the scheduler starts.

[ApplicationInit\(\)](#) MUST invoke [ZW_ApplicationRegisterTask\(\)](#) as a minimum.

Returns

- The Z-Wave stack acts differently depending on the returned value. If [APPLICATION_RUNNING](#) is returned, everything will boot normally. If [APPLICATION_POWER_DOWN](#) is returned, the Z-Wave stack will NOT start, but the scheduler will start.

sl_zwave_protocol_startup

```
void sl_zwave_protocol_startup (void )
```

Initialize the Z-Wave protocol stack.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

sl_zwave_platform_startup

```
void sl_zwave_platform_startup (void )
```

Early startup of Z-Wave platform features.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZW_GetProtocolBuildNumber

```
uint16_t ZW_GetProtocolBuildNumber (void )
```

Gets Z-Wave Protocol Build number.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Z-Wave Protocol Build number

ZW_GetProtocolGitHash

```
const uint8_t * ZW_GetProtocolGitHash (void )
```

Gets Z-Wave Protocol git hash.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- uint8_t* Z-Wave Protocol git hash

_APPL_NODE_TYPE_

Node type structure used in ApplicationNodeInformation.

Public Attributes

uint8_t	generic
	Generic Device Type.
uint8_t	specific
	Specific Device Type.

Public Attribute Documentation

generic

uint8_t _APPL_NODE_TYPE_::generic

Generic Device Type.

specific

uint8_t _APPL_NODE_TYPE_::specific

Specific Device Type.

_NODE_TYPE_

Public Attributes

uint8_t	basic	Basic Device Type, is it a Controller, Controller_Static, Slave or a Slave_Routing Device Type.
uint8_t	generic	Generic Device Type.
uint8_t	specific	Specific Device Type.

Public Attribute Documentation

basic

uint8_t _NODE_TYPE_::basic

Basic Device Type, is it a Controller, Controller_Static, Slave or a Slave_Routing Device Type.

generic

uint8_t _NODE_TYPE_::generic

Generic Device Type.

specific

uint8_t _NODE_TYPE_::specific

Specific Device Type.

NODEINFO

Node info stored within the non-volatile memory This are the first (protocol part) payload bytes from the Node Infomation frame.

Public Attributes

uint8_t	capability	Network capabilities.
uint8_t	security	Network security.
uint8_t	reserved	
NODE_TYPE	nodeType	Basic, Generic and Specific Device types - Basic is generated...

Public Attribute Documentation

capability

uint8_t _NODEINFO::capability

Network capabilities.

security

uint8_t _NODEINFO::security

Network security.

reserved

uint8_t _NODEINFO::reserved

nodeType

NODE_TYPE _NODEINFO::nodeType

Basic, Generic and Specific Device types - Basic is generated...

RSSI_LEVELS

Public Attributes

signed char [rssi_dBm](#)

Public Attribute Documentation

rssi_dBm

signed char RSSI_LEVELS::rssi_dBm[ZPAL_RADIO_NUM_CHANNELS_LR_CH_CFG_1_2]

Application Node Info

Application Node Info

Defines for ApplicationNodeInformation field deviceOptionMask.

Macros

```
#define APPLICATION_NODEINFO_NOT_LISTENING 0x00
    < Not listening Node, corresponds to Reporting Sleeping End Device role type

#define APPLICATION_NODEINFO_LISTENING 0x01
    Frequently Listening, corresponds to FLIRS role type. Wakes up every 1000ms.

#define APPLICATION_FREQ_LISTENING_MODE_1000ms 0x10
    Frequently Listening, corresponds to FLIRS role type. Wakes up every 250ms.

#define APPLICATION_FREQ_LISTENING_MODE_250ms 0x20
```

Z-Wave Security API

Z-Wave Security API

Modules

[_S_SECURITY_S2_INCLUSION_CSA_PUBLIC_DSK_](#)

[_S_APPLICATION_SECURITY_EVENT_DATA_](#)

[Security Keys Bitmask](#)

Enumerations

```
enum SECURITY_KEY {
    SECURITY_KEY_NONE = 0x00
    SECURITY_KEY_S2_UNAUTHENTICATED = 0x01
    SECURITY_KEY_S2_AUTHENTICATED = 0x02
    SECURITY_KEY_S2_ACCESS = 0x03
    SECURITY_KEY_S0 = 0x04
}
```

The security key a frame was received with or should be sent with.

```
enum _E_APPLICATION_SECURITY_EVENT_ {
    E_APPLICATION_SECURITY_EVENT_S2_INCLUSION_REQUEST_DSK_CSA
}
```

Definitions for Application bound Security events Delivered from protocol to Application through the [SReceiveSecurityEvent](#) notification being passed to application.

Typedefs

```
typedef enum SECURITY_KEY security_key_t
    The security key a frame was received with or should be sent with.

typedef struct _S_SECURITY_S2_INCLUSION_CSA_PUBLIC_DSK_ s_SecurityS2InclusionCSAPublicDSK_t

typedef enum _E_APPLICATION_SECURITY_EVENT_T_ e_application_security_event_t
    Definitions for Application bound Security events Delivered from protocol to Application through the
    SReceiveSecurityEvent notification being passed to application.

typedef struct _S_APPLICATION_SECURITY_EVENT_T_DATA_ s_application_security_event_data_t
    Security Event Data.
```

Macros

```
#define SECURITY_KEY_S2_PUBLIC_DSK_LENGTH 16
    Security S2 Public DSK Key length.
```

```
#define SECURITY_KEY_S2_PUBLIC_CSA_DSK_LENGTH 4
Security S2 Public CSA DSK Key length.
```

Enumeration Documentation

SECURITY_KEY

```
SECURITY_KEY
```

The security key a frame was received with or should be sent with.

Special values:

Enumerator	
SECURITY_KEY_NONE	SECURITY_KEY_NONE.
SECURITY_KEY_S2_UNAUTHENTICATED	SECURITY_KEY_S2_UNAUTHENTICATED.
SECURITY_KEY_S2_AUTHENTICATED	SECURITY_KEY_S2_AUTHENTICATED.
SECURITY_KEY_S2_ACCESS	SECURITY_KEY_S2_ACCESS.
SECURITY_KEY_S0	SECURITY_KEY_S0.

_E_APPLICATION_SECURITY_EVENT_

```
_E_APPLICATION_SECURITY_EVENT_
```

Definitions for Application bound Security events Delivered from protocol to Application through the [SReceiveSecurityEvent](#) notification being passed to application.

Enumerator	
E_APPLICATION_SECURITY_EVENT_S2_INCLUSION_REQUEST_DSK_CSA	

Typedef Documentation

security_key_t

```
typedef enum SECURITY_KEY security_key_t
```

The security key a frame was received with or should be sent with.

Special values:

s_SecurityS2InclusionCSAPublicDSK_t

```
typedef struct _S_SECURITY_S2_INCLUSION_CSA_PUBLIC_DSK_ s_SecurityS2InclusionCSAPublicDSK_t
```

e_application_security_event_t

```
typedef enum _E_APPLICATION_SECURITY_EVENT_ e_application_security_event_t
```


Definitions for Application bound Security events Delivered from protocol to Application through the [SReceiveSecurityEvent](#) notification being passed to application.

s_application_security_event_data_t

```
typedef struct _S_APPLICATION_SECURITY_EVENT_DATA_ s_application_security_event_data_t
```

Security Event Data.

_S_SECURITY_S2_INCLUSION_CSA_PUBLIC_DSK_

Public Attributes

uint8_t [aCSA_DSK](#)

Public Attribute Documentation

aCSA_DSK

uint8_t _S_SECURITY_S2_INCLUSION_CSA_PUBLIC_DSK_:aCSA_DSK[SECURITY_KEY_S2_PUBLIC_CSA_DSK_LENGTH]

_S_APPLICATION_SECURITY_EVENT_DATA_

Security Event Data.

Public Attributes

e_application_security_event_t	event
uint8_t	eventDataLength
uint8_t *	eventData

Public Attribute Documentation

event

e_application_security_event_t _S_APPLICATION_SECURITY_EVENT_DATA_::event

eventDataLength

uint8_t _S_APPLICATION_SECURITY_EVENT_DATA_::eventDataLength

eventData

uint8_t* _S_APPLICATION_SECURITY_EVENT_DATA_::eventData

Security Keys Bitmask

Security Keys Bitmask

Bitmask for security keys.

Macros

```
#define SECURITY_KEY_S2_UNAUTHENTICATED_BIT 0x01
#define SECURITY_KEY_S2_AUTHENTICATED_BIT 0x02
#define SECURITY_KEY_S2_ACCESS_BIT 0x04
#define SECURITY_KEY_S0_BIT 0x80
#define SECURITY_KEY_S2_MASK undefined
#define SECURITY_KEY_NONE_MASK 0x00
```

Z-Wave System Startup API

Z-Wave System Startup API

Functions

bool	IsWakeupCausedByRtccTimeout (void) Was the device woken up by an RTCC timer interrupt?
uint32_t	GetCompletedSleepDurationMs (void) Get number of milliseconds the device has spent sleeping in deep sleep before it was woken up.
uint32_t	GetDeepSleepWakeupTick (void) Get the sleeptimer tick value of the device when it woke up from deep sleep.
uint32_t	GetLastTickBeforeDeepSleep (void) Get the sleeptimer tick value of the device when it last went to Deep Sleep.
zpal_reset_reason_t	GetResetReason (void) Get the reset reason of the device when it was last reset or woke up from EM4 sleep.
bool	ZW_ApplicationRegisterTask (void(*appTaskFunc)(SApplicationHandles *), uint8_t iZwRxQueueTaskNotificationBitNumber, uint8_t iZwCommandStatusQueueTaskNotificationBitNumber, const SProtocolConfig_t *pProtocolConfig) Please.
SApplicationHandles *	ZW_system_startup_getAppHandles (void) Used to get the pointer to the application handles structure! Needed to insert task related data when creating user task.
void	ZW_system_startup_SetEventNotificationBitNumbers (uint8_t iZwRxQueueTaskNotificationBitNumber, uint8_t iZwCommandStatusQueueTaskNotificationBitNumber, const SProtocolConfig_t *pProtocolConfig) Used by ZW_UserTask_ApplicationRegisterTask() in ZW_UserTask.h to set the Event Notification Bit Numbers in the Application Interface structure!
void	ZW_system_startup_SetMainApplicationTaskHandle (TaskHandle_t xHandle) Used by ZW_UserTask_ApplicationRegisterTask() in ZW_UserTask.h to set the Main Application Task Handle in the Application Interface structure, after ZW_UserTask_ApplicationRegisterTask() has created the task!
void	ZW_system_startup_SetCCSet (SCommandClassSet_t *CCSet) Passed the Command Class set from the Application to the Protocol using a shared variable.
SCommandClassSet_t *	ZW_system_startup_GetCCSet (void) Gets the Command Class Set.
bool	ZW_system_startup_IsSchedulerStarted (void) INCLUDE_xTaskGetSchedulerState did not include xTaskGetSchedulerState in the build.
const SAppNodeInfo_t *	ZW_system_startup_getAppNodeInfo (void) Used to get a pointer to The Application node information SAppNodeInfo_t structure.

Function Documentation

IsWakeupCausedByRtccTimeout

```
bool IsWakeupCausedByRtccTimeout (void )
```

Was the device woken up by an RTCC timer interrupt?

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function can safely be called from multiple threads.

Returns

- true if woken up by RTCC timeout, false otherwise.

GetCompletedSleepDurationMs

```
uint32_t GetCompletedSleepDurationMs (void )
```

Get number of milliseconds the device has spent sleeping in deep sleep before it was woken up.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function can safely be called from multiple threads.

Returns

- Number of milliseconds.

GetDeepSleepWakeupTick

```
uint32_t GetDeepSleepWakeupTick (void )
```

Get the sleeptimer tick value of the device when it woke up from deep sleep.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function can safely be called from multiple threads.

Returns

- RTCC ticks.

GetLastTickBeforeDeepSleep

```
uint32_t GetLastTickBeforeDeepSleep (void )
```

Get the sleeptimer tick value of the device when it last went to Deep Sleep.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

(It is stored in retention RAM)

This function can safely be called from multiple threads.

Returns

- RTCC ticks.

GetResetReason

```
zpal_reset_reason_t GetResetReason (void )
```

Get the reset reason of the device when it was last reset or woke up from EM4 sleep.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function can safely be called from multiple threads.

Returns

- reset reason.

ZW_ApplicationRegisterTask

```
bool ZW_ApplicationRegisterTask (void*)(SApplicationHandles *) appTaskFunc, uint8_t iZwRxQueueTaskNotificationBitNumber, uint8_t iZwCommandStatusQueueTaskNotificationBitNumber, const SProtocolConfig_t * pProtocolConfig)
```

Please.

Parameters

Type	Direction	Argument Name	Description
void*)(SApplicationHandles *)	N/A	appTaskFunc	
uint8_t	N/A	iZwRxQueueTaskNotificationBitNumber	
uint8_t	N/A	iZwCommandStatusQueueTaskNotificationBitNumber	
const SProtocolConfig_t *	N/A	pProtocolConfig	

See Also

- [ZW_UserTask_ApplicationRegisterTask\(\)](#) in [ZW_UserTask.h](#) for more info.

ZW_system_startup_getAppHandles

```
SApplicationHandles * ZW_system_startup_getAppHandles (void )
```

Used to get the pointer to the application handles structure! Needed to insert task related data when creating user task.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZW_system_startup_SetEventNotificationBitNumbers

```
void ZW_system_startup_SetEventNotificationBitNumbers (uint8_t iZwRxQueueTaskNotificationBitNumber, uint8_t iZwCommandStatusQueueTaskNotificationBitNumber, const SProtocolConfig_t * pProtocolConfig)
```

Used by [ZW_UserTask_ApplicationRegisterTask\(\)](#) in [ZW_UserTask.h](#) to set the Event Notification Bit Numbers in the Application Interface structure!

Parameters

Type	Direction	Argument Name	Description
uint8_t	N/A	iZwRxQueueTaskNotificationBitNumber	
uint8_t	N/A	iZwCommandStatusQueueTaskNotificationBitNumber	
const SProtocolConfig_t *	N/A	pProtocolConfig	

ZW_system_startup_SetMainApplicationTaskHandle

```
void ZW_system_startup_SetMainApplicationTaskHandle (TaskHandle_t xHandle)
```

Used by [ZW_UserTask_ApplicationRegisterTask\(\)](#) in [ZW_UserTask.h](#) to set the Main Application Task Handle in the Application Interface structure, after [ZW_UserTask_ApplicationRegisterTask\(\)](#) has created the task!

Parameters

Type	Direction	Argument Name	Description
TaskHandle_t	N/A	xHandle	

ZW_system_startup_SetCCSet

```
void ZW_system_startup_SetCCSet (SCommandClassSet\_t * CCSet)
```

Passed the Command Class set from the Application to the Protocol using a shared variable.

Parameters

Type	Direction	Argument Name	Description
SCommandClassSet_t *	[in]	CCSet	Command Class Set

ZW_system_startup_GetCCSet

```
SCommandClassSet_t * ZW_system_startup_GetCCSet (void )
```


Gets the Command Class Set.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- SCommandClassSet_t* A pointer to the Command Class Set

ZW_system_startup_IsSchedulerStarted

```
bool ZW_system_startup_IsSchedulerStarted (void )
```

INCLUDE_xTaskGetSchedulerState did not include xTaskGetSchedulerState in the build.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This is a workaround!

This function indicated whether the Scheduler has ever been started. This will not indicate the current state, e.g. whether the scheduler is in suspended state or running!

ZW_system_startup_getAppNodeInfo

```
const SAppNodeInfo_t * ZW_system_startup_getAppNodeInfo (void )
```

Used to get a pointer to The Application node information [SAppNodeInfo_t](#) structure.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- const SAppNodeInfo_t*

Z-Wave Transport API

Z-Wave Transport API

Z-Wave Transport Application layer interface.

The Z Wave transport layer controls transfer of data between Z Wave nodes including retransmission, frame check and acknowledgement. The Z Wave transport interface includes functions for transfer of data to other Z Wave nodes. Application data received from other nodes is handed over to the application via the [ApplicationCommandHandler](#) function. The ZW_MAX_NODES define defines the maximum of nodes possible in a Z Wave network.

Modules

- [s_rssi_val](#)
- [_S_ROUTE_LINK_T_](#)
- [_TX_STATUS_TYPE_](#)
- [_RECEIVE_OPTIONS_TYPE](#)
- [_TRANSMIT_OPTIONS_TYPE](#)
- [_TRANSMIT_MULTI_OPTIONS_TYPE](#)
- [_RECEIVE_OPTIONS_TYPE.homelId](#)
- [Transmit Options](#)
- [Receive Status](#)
- [Transmit complete codes](#)

Typedefs

typedef uint32_t	TxOptions_t The universal TX-Option flag count.
typedef struct _S_ROUTE_LINK_T_	S_ROUTE_LINK_T
typedef struct _TX_STATUS_TYP E_	TX_STATUS_TYPE
typedef struct _RECEIVE_OPTIO NS_TYPE	RECEIVE_OPTIONS_TYPE
typedef struct _TRANSMIT_OPTI ONS_TYPE	TRANSMIT_OPTIONS_TYPE Transmit options for ZW_SendDataEx.
typedef struct _TRANSMIT_MUL TI_OPTIONS_TYP E	TRANSMIT_MULTI_OPTIONS_TYPE Transmit options for ZW_SendDataMultiEx.

#define	ZW_MAX_NODES	232
#define	ZW_MAX_NODES_LR	1024
		Max number of Long Range nodes in a Z-wave system Must not be higher than 4000 according to specs.
#define	LOWEST_LONG_RANGE_NODE_ID	(0×0100)
#define	HIGHEST_LONG_RANGE_NODE_ID	(LOWEST_LONG_RANGE_NODE_ID + ZW_MAX_NODES_LR - 1)
#define	NODEPARAM_MAX	35
		max. number of parameters
#define	HOMEID_LENGTH	4
		Number of bytes in a homeID.
#define	GET-NLS_NODES_LIST_LENGTH_MAX	(128)
		Frames should not exceed 128 bytes long.
#define	NODE_BROADCAST	0xFF
		Predefined Node ID's.
#define	ZW_TEST_NOT_A_NODEID	0×00
#define	ZW_MAX_CACHED_RETURN_ROUTE_DESTINATIONS	5
#define	MAX_REPEATERS	4
		Max hops in route.
#define	LAST_USED_ROUTE_CONF_SIZE	1
		TX_STATUS_TYPE Last Used Route array size definitions.
#define	LAST_USED_ROUTE_SIZE	(MAX_REPEATERS + LAST_USED_ROUTE_CONF_SIZE)
#define	S2_TXOPTION_VERIFY_DELIVERY	1
		This flag will activate frame delivery.
#define	S2_TXOPTION_SINGLECAST_FOLLOWUP	2
		This flag must be present on all single cast followup messages.
#define	S2_TXOPTION_FIRST_SINGLECAST_FOLLOWUP	4
		This flag must be present on the first, and only the first single cast followup message in a S2 multicast transmission.

Typedef Documentation

TxOptions_t

```
typedef uint32_t TxOptions_t
```

The universal TX-Option flag count.

S_ROUTE_LINK_T

```
typedef struct _S_ROUTE_LINK_T S_ROUTE_LINK_T
```

TX_STATUS_TYPE

```
typedef struct _TX_STATUS_TYPE_ TX_STATUS_TYPE
```

RECEIVE_OPTIONS_TYPE

```
typedef struct _RECEIVE_OPTIONS_TYPE RECEIVE_OPTIONS_TYPE
```

TRANSMIT_OPTIONS_TYPE

```
typedef struct _TRANSMIT_OPTIONS_TYPE TRANSMIT_OPTIONS_TYPE
```

Transmit options for ZW_SendDataEx.

TRANSMIT_MULTI_OPTIONS_TYPE

```
typedef struct _TRANSMIT_MULTI_OPTIONS_TYPE TRANSMIT_MULTI_OPTIONS_TYPE
```

Transmit options for ZW_SendDataMultiEx.

s_rssi_val

RSSI value array used in TX_STATUS_TYPE.

Each value is an RSSI feedback constant defined above.

Public Attributes

signed char [incoming](#)

Public Attribute Documentation

incoming

signed char s_rssi_val::incoming[MAX_REPEATERS+1]

_S_ROUTE_LINK_T_

Public Attributes

```
uint8_t  from
uint8_t  to
```

Public Attribute Documentation

from

```
uint8_t _S_ROUTE_LINK_T::from
```

to

```
uint8_t _S_ROUTE_LINK_T::to
```

_TX_STATUS_TYPE_

Public Attributes

uint32_t	TransmitTicks Passed 1ms ticks.
uint8_t	bRepeaters Repeaters in route, zero for direct range.
struct s_rssi_val	rssi_values rssi_values per hop for direct and routed frames.
uint8_t	bACKChannelNo
uint8_t	bLastTxChannelNo
uint8_t	bRouteSchemeState
uint8_t	pLastUsedRoute
uint8_t	bRouteTries
S_ROUTE_LINK_T	bLastFailedLink
int8_t	bUsedTxpower
int8_t	bMeasuredNoiseFloor
int8_t	bDestinationAckUsedTxPower
int8_t	bDestinationAckMeasuredRSSI
int8_t	bDestinationAckMeasuredNoiseFloor

Public Attribute Documentation

TransmitTicks

uint32_t _TX_STATUS_TYPE_:TransmitTicks

Passed 1ms ticks.

bRepeaters

uint8_t _TX_STATUS_TYPE_:bRepeaters

Repeaters in route, zero for direct range.

rssi_values

```
struct s_rssi_val _TX_STATUS_TYPE_::rssi_values
```

rssi_values per hop for direct and routed frames.

Contains repeaters + 1 values.

bACKChannelNo

```
uint8_t _TX_STATUS_TYPE_::bACKChannelNo
```

bLastTxChannelNo

```
uint8_t _TX_STATUS_TYPE_::bLastTxChannelNo
```

bRouteSchemeState

```
uint8_t _TX_STATUS_TYPE_::bRouteSchemeState
```

pLastUsedRoute

```
uint8_t _TX_STATUS_TYPE_::pLastUsedRoute[LAST_USED_ROUTE_SIZE]
```

bRouteTries

```
uint8_t _TX_STATUS_TYPE_::bRouteTries
```

bLastFailedLink

```
S_ROUTE_LINK_T _TX_STATUS_TYPE_::bLastFailedLink
```

bUsedTxpower

```
int8_t _TX_STATUS_TYPE_::bUsedTxpower
```

bMeasuredNoiseFloor

int8_t _TX_STATUS_TYPE_::bMeasuredNoiseFloor

bDestinationAckUsedTxPower

int8_t _TX_STATUS_TYPE_::bDestinationAckUsedTxPower

bDestinationAckMeasuredRSSI

int8_t _TX_STATUS_TYPE_::bDestinationAckMeasuredRSSI

bDestinationAckMeasuredNoiseFloor

int8_t _TX_STATUS_TYPE_::bDestinationAckMeasuredNoiseFloor

_RECEIVE_OPTIONS_TYPE

Public Attributes

uint8_t	rxStatus	< Frame header info
union _RECEIVE_OPTIO NS_TYPE:@2	homeId	Command sender Node ID.
node_id_t	sourceNode	Frame destination ID, only valid when frame is not Multicast.
node_id_t	destNode	Average RSSI val in dBm as defined in RSSI feedback constants above.
signed char	rxRSSIVal	Security key frame was received with.
enum SECURITY_KEY	securityKey	CommunicationProfile used on receive.
uint8_t	rxSpeed	Channel on which frame was received is a Z-Wave Long Range channel.
bool	isLongRangeChannel	Transmit power used by the source to transmit the frame.
int8_t	bSourceTxPower	Noise floor measured by the source node before transmitting.
int8_t	bSourceNoiseFloor	

Public Attribute Documentation

rxStatus

uint8_t _RECEIVE_OPTIONS_TYPE::rxStatus

< Frame header info

Command sender Home ID

homeId

union _RECEIVE_OPTIONS_TYPE:@2 _RECEIVE_OPTIONS_TYPE::homeId

Command sender Node ID.

sourceNode

```
node_id_t _RECEIVE_OPTIONS_TYPE::sourceNode
```

Frame destination ID, only valid when frame is not Multicast.

destNode

```
node_id_t _RECEIVE_OPTIONS_TYPE::destNode
```

Average RSSI val in dBm as defined in RSSI feedback constants above.

rxRSSIVal

```
signed char _RECEIVE_OPTIONS_TYPE::rxRSSIVal
```

Security key frame was received with.

securityKey

```
enum SECURITY_KEY _RECEIVE_OPTIONS_TYPE::securityKey
```

CommunicationProfile used on receive.

rxSpeed

```
uint8_t _RECEIVE_OPTIONS_TYPE::rxSpeed
```

Channel on which frame was received is a Z-Wave Long Range channel.

isLongRangeChannel

```
bool _RECEIVE_OPTIONS_TYPE::isLongRangeChannel
```

Transmit power used by the source to transmit the frame.

bSourceTxPower

```
int8_t _RECEIVE_OPTIONS_TYPE::bSourceTxPower
```

Noise floor measured by the source node before transmitting.

bSourceNoiseFloor

int8_t _RECEIVE_OPTIONS_TYPE::bSourceNoiseFloor

_TRANSMIT_OPTIONS_TYPE

Transmit options for ZW_SendDataEx.

Public Attributes

<code>node_id_t</code>	<code>destNode</code> < Destination node ID - 0xFF == all nodes
<code>node_id_t</code>	<code>bSrcNode</code> Transmit options.
<code>TxOptions_t</code>	<code>txOptions</code> Options for enabling specific Security scheme functionality.
<code>uint8_t</code>	<code>txSecOptions</code> Security key to use for sending.
<code>enum SECURITY_KEY</code>	<code>securityKey</code> More transmit options.
<code>uint8_t</code>	<code>txOptions2</code>

Public Attribute Documentation

destNode

`node_id_t _TRANSMIT_OPTIONS_TYPE::destNode`

< Destination node ID - 0xFF == all nodes

Reserved

bSrcNode

`node_id_t _TRANSMIT_OPTIONS_TYPE::bSrcNode`

Transmit options.

txOptions

`TxOptions_t _TRANSMIT_OPTIONS_TYPE::txOptions`

Options for enabling specific Security scheme functionality.

txSecOptions

`uint8_t _TRANSMIT_OPTIONS_TYPE::txSecOptions`

Security key to use for sending.

securityKey

```
enum SECURITY_KEY _TRANSMIT_OPTIONS_TYPE::securityKey
```

More transmit options.

txOptions2

```
uint8_t _TRANSMIT_OPTIONS_TYPE::txOptions2
```

_TRANSMIT_MULTI_OPTIONS_TYPE

Transmit options for ZW_SendDataMultiEx.

Public Attributes

uint8_t	groupID	< Destination group ID
node_id_t	bSrcNode	Transmit options.
TxOptions_t	txOptions	Security key to use for sending - only S2 keys are valid.
enum SECURITY_KEY	securityKey	

Public Attribute Documentation

groupID

uint8_t _TRANSMIT_MULTI_OPTIONS_TYPE::groupID

< Destination group ID

Reserved

bSrcNode

node_id_t _TRANSMIT_MULTI_OPTIONS_TYPE::bSrcNode

Transmit options.

txOptions

TxOptions_t _TRANSMIT_MULTI_OPTIONS_TYPE::txOptions

Security key to use for sending - only S2 keys are valid.

securityKey

enum SECURITY_KEY _TRANSMIT_MULTI_OPTIONS_TYPE::securityKey

_RECEIVE_OPTIONS_TYPE.homeld

Public Attributes

uint8_t [array](#)

uint32_t [word](#)

Public Attribute Documentation

array

word

Transmit Options

Transmit Options

Transmit frame option flags.

Macros

```
#define TRANSMIT_OPTION_ACK 0x01
Request acknowledge from destination node.

#define TRANSMIT_OPTION_LOW_POWER 0x02
Rransmit at low output power level (1/3 of normal RF range)

#define TRANSMIT_OPTION_MULTICAST_AS_BROADCAST 0x02
The multicast frame should be send as a broadcast.

#define TRANSMIT_OPTION_AUTO_ROUTE 0x04
Request retransmission via repeater nodes.

#define TRANSMIT_OPTION_NO_ROUTE 0x10
Do not use response route - Even if available.

#define TRANSMIT_OPTION_EXPLORE 0x20
Use explore frame if needed.
```

Receive Status

Receive Status

Status codes for receiving frames.

Macros

```
#define RECEIVE_STATUS_ROUTED_BUSY 0x01
    < A response route is locked by the application

#define RECEIVE_STATUS_LOW_POWER 0x02
    Received at low output power level, this must have the same value as TRANSMIT_OPTION_LOW_POWER.

#define RECEIVE_STATUS_TYPE_MASK 0x0C
    Mask for masking out the received frametype bits.

#define RECEIVE_STATUS_TYPE_SINGLE 0x00
    Received frame is singlecast frame (rxOptions == xxxx00xx)

#define RECEIVE_STATUS_TYPE_BROAD 0x04
    Received frame is broadcast frame (rxOptions == xxxx01xx)

#define RECEIVE_STATUS_TYPE_MULTI 0x08
    Received frame is multicast frame (rxOptions == xxxx10xx)

#define RECEIVE_STATUS_TYPE_EXPLORE 0x10
    Received frame is an explore frame (rxOptions == xxx1xxxx) Only TYPE_BROAD can be active at the same time as
    TYPE_EXPLORE.

#define RECEIVE_STATUS_FOREIGN_FRAME 0x40
    Received frame is not send to me (rxOptions == x1xxxxxx)
```

Transmit complete codes

Transmit complete codes

Macros

```
#define TRANSMIT_COMPLETE_OK 0x00
Transmit OK.

#define TRANSMIT_COMPLETE_NO_ACK 0x01
Retransmission error.

#define TRANSMIT_COMPLETE_FAIL 0x02
Transmit error.

#define TRANSMIT_ROUTING_NOT_IDLE 0x03
Transmit error.

#define TRANSMIT_COMPLETE_VERIFIED 0x05
Security S2. Receiving node has decrypted the frame.
```

Z-Wave Typedefs

Z-Wave Typedefs

Functions

```
uint32_t  minimum(uint32_t x, uint32_t y)
```

Macros

```
#define ZW_WEAK __attribute__((weak))

#define EOF (-1)

#define NULL ((void*)0)

#define NO_RETURN
    This definition is used for task-functions and allows the compiler to know this detail and therefore do optimizations on the function.

#define VOID_CALLBACKFUNC (completedFunc)
    Define for making easy and consistent callback definitions.

#define IS_NULL (x)
    Safe null pointer check.

#define NON_NULL (x)

#define UIP_HTONL (x)
```

Function Documentation

minimum

```
static uint32_t minimum (uint32_t x, uint32_t y)
```

Parameters

Type	Direction	Argument Name	Description
uint32_t	N/A	x	
uint32_t	N/A	y	

Z-Wave User Task API

Z-Wave User Task API

Modules

- [ZW_UserTask_Buffer_t](#)
- [ZW_UserTask_t](#)

Enumerations

```
enum ZW\_UserTask\_Priority\_t {  
    USERTASK_PRIORITY_BACKGROUND = 0  
    USERTASK_PRIORITY_NORMAL  
    USERTASK_PRIORITY_HIGHEST  
}  
  
enum ReturnCode\_t {  
    Code_Success = 0  
    Code_Fail_Unknown  
    Code_Fail_NoMemory  
    Code_Fail_LimitReached  
    Code_Fail_NotFound  
    Code_Fail_Suspended  
    Code_Fail_InvalidState  
    Code_Fail_InvalidOperation  
    Code_Fail_InvalidParameter  
    Code_DependencyUnavailable  
    Code_Fail_Busy  
    Code_Fail_Timeout  
    Code_Fail_Driver  
    Code_Fail_NotImplemented  
    Code_Fail_NotAllowed  
}  
Return code used by the functions in this module.
```

Functions

- [ReturnCode_t](#) [ZW_UserTask_Init](#)(void)
Initializes this module.
- [ReturnCode_t](#) [ZW_UserTask_CreateTask](#)(ZW_UserTask_t *task, TaskHandle_t *xHandle)
Used to create user application level tasks.
- [bool](#) [ZW_UserTask_ApplicationRegisterTask](#)(void(*appTaskFunc)(SApplicationHandles *), uint8_t iZwRxQueueTaskNotificationBitNumber, uint8_t iZwCommandStatusQueueTaskNotificationBitNumber, const SProtocolConfig_t *pProtocolConfig)
Used to create the main application task at highest UserTask priority.

Enumeration Documentation

ZW_UserTask_Priority_t

ZW_UserTask_Priority_t

Enumerator

USERTASK_PRIORITY_BACKGROUND	Right above idle priority. Good for HMI and worker threads.
USERTASK_PRIORITY_NORMAL	The default 'APP'-task priority.
USERTASK_PRIORITY_HIGHEST	High priority UserTask. Good for handling interrupts requests.

ReturnCode_t

ReturnCode_t

Return code used by the functions in this module.

Enumerator

Code_Success	Success.
Code_Fail_Unknown	Default failed code with no helping indication.
Code_Fail_NoMemory	Failed due to lack of memory.
Code_Fail_LimitReached	A set limit is reached for this operation.
Code_Fail_NotFound	A specific entity could not be found.
Code_Fail_Suspended	The operation or the entire module is globally suspended and cannot be utilized at this point in time.
Code_Fail_InvalidState	Not in an appropriate state for this call.
Code_Fail_InvalidOperation	This operation can not be allowed at this point.
Code_Fail_InvalidParameter	Wrong/invalid input parameters.
Code_DependencyUnavailable	A required resource or dependency that this operation relies on was not setup or available. Perhaps wrong initialization order.
Code_Fail_Busy	A resource needed by this operation or call is busy. Try again later. Potential racing condition or untimely action.
Code_Fail_Timeout	Failed due to timeout.
Code_Fail_Driver	Error occurred at the driver level.
Code_Fail_NotImplemented	This operation is not supported due to limited implementation or need.
Code_Fail_NotAllowed	This operation cannot be performed, since it is blocked or not allowed.

Function Documentation

ZW_UserTask_Init

ReturnCode_t ZW_UserTask_Init (void)

Initializes this module.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

- Should not be used directly.

Return Returns ReturnCode_t. **See Also**

- See [ReturnCode_t](#) for more info.

ZW_UserTask_CreateTask

```
ReturnCode_t ZW_UserTask_CreateTask (ZW_UserTask_t * task, TaskHandle_t * xHandle)
```

Used to create user application level tasks.

Parameters

Type	Direction	Argument Name	Description
ZW_UserTask_t *	[out]	task	Used to return the task handle. Can be NULL Return Returns ReturnCode_t. See ReturnCode_t for more info.
TaskHandle_t *	N/A	xHandle	

See Also

- See [ZW_UserTask_t](#) for more info.

ZW_UserTask_ApplicationRegisterTask

```
bool ZW_UserTask_ApplicationRegisterTask (void(*)(SApplicationHandles *) appTaskFunc, uint8_t iZwRxQueueTaskNotificationBitNumber, uint8_t iZwCommandStatusQueueTaskNotificationBitNumber, const SProtocolConfig\_t * pProtocolConfig)
```

Used to create the main application task at highest UserTask priority.

Parameters

Type	Direction	Argument Name	Description
void(*)(SApplicationHandles *)	N/A	appTaskFunc	Application task that will be triggered by FreeRTOS. (function pointer)
uint8_t	N/A	iZwRxQueueTaskNotificationBitNumber	The bit number set by Protocol when a message has been put on the ZW Rx queue. (Receiving EAPPLICATIONEVENT_ZWRX events)
uint8_t	N/A	iZwCommandStatusQueueTaskNotificationBitNumber	The bit number Set by Protocol when a message has been put on the ZW command status queue. (Receiving EAPPLICATIONEVENT_ZWCOMMANDSTATUS events)
const SProtocolConfig_t *	N/A	pProtocolConfig	Pointer to a statically allocated Protocol Config structure.

This function further registers Queue Notification Bit Numbers so that an association is created between the bit number and the event handle to invoke for that event, when it occurs.

This function should only be called once, or false is returned

- This task is the only UserTask that should use the ZAF API functions. Accessing the ZAF API functions from any other [ZW_UserTask.h](#) generated tasks will lead to undefined behavior

See Also

- [SProtocolConfig_t](#)

Returns

- true if the main APP task was created successfully, false if not.

ZW_UserTask_Buffer_t

A buffer structure to allocate the needed memory for a task in user space.

The stack size is not fixed, but limited to between [TASK_STACK_SIZE_MAX](#) and [TASK_STACK_SIZE_MIN](#). The actual size chosen must be set in `stackBufferLength`. The buffers must be allocated elsewhere, since this structure does not allocate any space, but holds the pointers to the buffers.

Public Attributes

StaticTask_t *	taskBuffer	Memory needed by the Task for internal states. Pointer to buffer.
uint8_t *	stackBuffer	Pointer to a statically allocated stack-buffer-array for the task. Pointer to buffer.
uint16_t	stackBufferLength	This is measured in bytes and is preferably a multiple of 4, or it will be rounded down.

Public Attribute Documentation

taskBuffer

StaticTask_t* ZW_UserTask_Buffer_t::taskBuffer

Memory needed by the Task for internal states. Pointer to buffer.

stackBuffer

uint8_t* ZW_UserTask_Buffer_t::stackBuffer

Pointer to a statically allocated stack-buffer-array for the task. Pointer to buffer.

stackBufferLength

uint16_t ZW_UserTask_Buffer_t::stackBufferLength

This is measured in bytes and is preferably a multiple of 4, or it will be rounded down.

ZW_UserTask_t

Public Attributes

TaskFunction_t	pTaskFunc	The function of the task (Invalid parameter if NULL)
char *	pTaskName	The name of the task. (Max characters = configMAX_TASK_NAME_LEN). NULL allowed.
void *	pUserTaskParam	Any parameter variable of own choice that is casted as (void*). Can be NULL.
ZW_UserTask_Priority_t	priority	The desired priority of the task.
ZW_UserTask_Buffer_t *	taskBuffer	The buffer collection for the task.

Public Attribute Documentation

pTaskFunc

TaskFunction_t ZW_UserTask_t::pTaskFunc

The function of the task (Invalid parameter if NULL)

pTaskName

char* ZW_UserTask_t::pTaskName

The name of the task. (Max characters = configMAX_TASK_NAME_LEN). NULL allowed.

pUserTaskParam

void* ZW_UserTask_t::pUserTaskParam

Any parameter variable of own choice that is casted as (void*). Can be NULL.

priority

ZW_UserTask_Priority_t ZW_UserTask_t::priority

The desired priority of the task.

taskBuffer

ZW_UserTask_Buffer_t* ZW_UserTask_t::taskBuffer

The buffer collection for the task.

See Also

- [ZW_UserTask_Buffer_t](#) for more info.

Components

Components

Modules

[Events](#)[MfgTokens](#)[NodeMask](#)[Queue](#)[Timer](#)[SyncEvent](#)

Events

Events

Modules

- EventDistributor
- ZAF Event Handling

EventDistributor

EventDistributor

Modules

- [event_cc_t](#)
- [zaf_event_distributor_cc_event_handler_map_v1_t](#)
- [SEventDistributor](#)

Enumerations

```
enum EEventDistributorStatus {  
    EEVENTDISTRIBUTOR_STATUS_SUCCESS = 0  
    EEVENTDISTRIBUTOR_STATUS_TABLE_TOO_LARGE  
}  
Defines return values from EventDistributorConfigure.
```

Typedefs

```
typedef void(*) zaf_event_distributor_cc_event_handler_handler_v1_t)(const uint8_t, const void *)  
  
typedef zaf_event_distributor_cc_event_handler_map_latest_t  
zaf_event_distributor_cc_event_handler_map_v1_t  
Redefine latest version to a common define called latest.  
  
typedef enum EEventDistributorStatus  
EEventDistributorStatus  
Defines return values from EventDistributorConfigure.  
  
typedef void(*) EventDistributorEventHandler)(void)  
Defines callback pointer for Event handler.  
  
typedef struct SEventDistributor  
SEventDistributor  
EventDistributor object.
```

Functions

```
void zaf_event_distributor_init(void)  
Initializes the ZAF Event Distributor component.  
  
const SEventDistributor* zaf_event_distributor_get(void)  
Gets the events distributor.  
  
uint32_t zaf_event_distributor_distribute(void)  
Distributes events.  
  
void zaf_event_distributor_app_zw_rx(SZwaveReceivePackage *RxPackage)  
Used by the application to handle protocol received events.  
  
void zaf_event_distributor_app_zw_command_status(SZwaveCommandStatusPackage *Status)
```

Used by the application to handle protocol command status.

void	zaf_event_distributor_app_state_change (void) Used by the application to handle state changes in the NCP state machine.
void	zaf_event_distributor_app_serial_data_rx (void) Used by the application to handle serial data received.
void	zaf_event_distributor_app_serial_timeout (void) Used by the application to handle serial timeout.
void	zaf_event_distributor_app_event_manager (const uint8_t event) Used by the application to handle application/system received events.
uint8_t	zaf_event_distributor_is_primary_controller (void) Used by application to verify if it is a primary controller.
bool	zaf_event_distributor_enqueue_app_event (const uint8_t event) Enqueues an event in the application queue.
bool	zaf_event_distributor_enqueue_app_event_from_isr (const uint8_t event) Enqueues an event in the application queue.
bool	zaf_event_distributor_enqueue_cc_event (const uint16_t command_class, const uint8_t event, const void *data) Enqueues an event in the command class event queue.
bool	zaf_event_distributor_enqueue_cc_event_from_isr (const uint16_t command_class, const uint8_t event, const void *data) Enqueues an event in the command class event queue.
EEEventDistributorStatus	EventDistributorConfig (SEventDistributor *pThis, uint8_t iEventHandlerTableSize, const EventDistributorEventHandler *pEventHandlerTable, void(*pNoEvent)(void)) Configures the EventDistributor.
uint32_t	EventDistributorDistribute (const SEventDistributor *pThis, uint32_t iEventWait, uint32_t NotificationClearMask) Waits for Task notification and calls event handlers matching the set notification bits.

Macros

#define	ZAF_EVENT_DISTRIBUTOR_CC_EVENT_HANDLER_SECTION "zw_zaf_event_distributor_cc_event_handler" Section name.
#define	ZAF_EVENT_DISTRIBUTOR_REGISTER_CC_EVENT_HANDLER (command_class, handler) Registers a custom event handler for a specific command class.

Enumeration Documentation

EEEventDistributorStatus

EEEventDistributorStatus	
Defines return values from EventDistributorConfigure.	
Enumerator	
EEVENTDISTRIBUTOR_STATUS_SUCCESS	
EEVENTDISTRIBUTOR_STATUS_TABLE_TOO_LARGE	

Typedef Documentation

zaf_event_distributor_cc_event_handler_handler_v1_t

```
typedef void(* zaf_event_distributor_cc_event_handler_handler_v1_t) (const uint8_t, const void *) )(const uint8_t, const void *)
```

zaf_event_distributor_cc_event_handler_map_latest_t

```
typedef zaf_event_distributor_cc_event_handler_map_v1_t zaf_event_distributor_cc_event_handler_map_latest_t
```

Redefine latest version to a common define called latest.

EEventDistributorStatus

```
typedef enum EEventDistributorStatus EEventDistributorStatus
```

Defines return values from EventDistributorConfigure.

EventDistributorEventHandler

```
typedef void(* EventDistributorEventHandler) (void) )(void)
```

Defines callback pointer for Event handler.

Typedef to simplify syntax.

SEventDistributor

```
typedef struct SEventDistributor SEventDistributor
```

EventDistributor object.

All content is private.

Function Documentation

zaf_event_distributor_init

```
void zaf_event_distributor_init (void )
```

Initializes the ZAF Event Distributor component.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

This function invokes [EventDistributorConfig\(\)](#).

zaf_event_distributor_get

```
const SEventDistributor * zaf_event_distributor_get (void )
```

Gets the events distributor.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Each variation (NCP,SOC) must implement it's own event distributor table

Returns

- const [SEventDistributor](#)

zaf_event_distributor_distribute

```
uint32_t zaf_event_distributor_distribute (void )
```

Distributes events.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Wraps [EventDistributorDistribute\(\)](#).

Returns

- bitfield with unhandled events

zaf_event_distributor_app_zw_rx

```
void zaf_event_distributor_app_zw_rx (SZwaveReceivePackage * RxPackage)
```

Used by the application to handle protocol received events.

Parameters

Type	Direction	Argument Name	Description
SZwaveReceivePackage *	N/A	RxPackage	ZW Receive Package

zaf_event_distributor_app_zw_command_status

```
void zaf_event_distributor_app_zw_command_status (SZwaveCommandStatusPackage * Status)
```

Used by the application to handle protocol command status.

Parameters

Type	Direction	Argument Name	Description
SZwaveCommandStatusPackage *	N/A	Status	ZW Command Status Package

zaf_event_distributor_app_state_change

```
void zaf_event_distributor_app_state_change (void )
```

Used by the application to handle state changes in the NCP state machine.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zaf_event_distributor_app_serial_data_rx

```
void zaf_event_distributor_app_serial_data_rx (void )
```

Used by the application to handle serial data received.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zaf_event_distributor_app_serial_timeout

```
void zaf_event_distributor_app_serial_timeout (void )
```

Used by the application to handle serial timeout.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zaf_event_distributor_app_event_manager

```
void zaf_event_distributor_app_event_manager (const uint8_t event)
```

Used by the application to handle application/system received events.

Parameters

Type	Direction	Argument Name	Description
const uint8_t	N/A	event	application/system received event

zaf_event_distributor_is_primary_controller

```
uint8_t zaf_event_distributor_is_primary_controller (void )
```

Used by application to verify if it is a primary controller.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- uint8_t 1 if it is primary controller else 0

zaf_event_distributor_enqueue_app_event

```
bool zaf_event_distributor_enqueue_app_event (const uint8_t event)
```

Enqueues an event in the application queue.

Parameters

Type	Direction	Argument Name	Description
const uint8_t	N/A	event	The event to enqueue.

Returns

- Returns true if the event was enqueued and false otherwise.

zaf_event_distributor_enqueue_app_event_from_isr

```
bool zaf_event_distributor_enqueue_app_event_from_isr (const uint8_t event)
```

Enqueues an event in the application queue.

Parameters

Type	Direction	Argument Name	Description
const uint8_t	N/A	event	The event to enqueue.

Returns

- Returns true if the event was enqueued and false otherwise.

zaf_event_distributor_enqueue_cc_event

```
bool zaf_event_distributor_enqueue_cc_event (const uint16_t command_class, const uint8_t event, const void * data)
```

Enqueues an event in the command class event queue.

Parameters

Type	Direction	Argument Name	Description
const uint16_t	N/A	command_class	The command class to receive the event
const uint8_t	N/A	event	The event to enqueue.
const void *	N/A	data	A data pointer. It must point to a global variable

Returns

- Returns true if the event was enqueued and false otherwise.

zaf_event_distributor_enqueue_cc_event_from_isr

```
bool zaf_event_distributor_enqueue_cc_event_from_isr (const uint16_t command_class, const uint8_t event, const void * data)
```

Enqueues an event in the command class event queue.

Parameters

Type	Direction	Argument Name	Description
const uint16_t	N/A	command_class	The command class to receive the event
const uint8_t	N/A	event	The event to enqueue.
const void *	N/A	data	A data pointer. It must point to a global variable

Returns

- Returns true if the event was enqueued and false otherwise.

EventDistributorConfig

```
EEEventDistributorStatus EventDistributorConfig (SEventDistributor * pThis, uint8_t iEventHandlerTableSize, const EventDistributorEventHandler * pEventHandlerTable, void(*) (void) pNoEvent)
```

Configures the EventDistributor.

Parameters

Type	Direction	Argument Name	Description
SEventDistributor *	[in]	pThis	Pointer to the EventDistributor object.
uint8_t	[in]	iEventHandlerTableSize	Number of entries in EventHandlerTable pointed to by argument pEventHandlerTable Table size range is 0 - 32)
const EventDistributorEventHandler *	[in]	pEventHandlerTable	Pointer to table of EventHandler function pointers
void(*) (void)	[in]	pNoEvent	Function pointer that will be called on notifications without event or wait for event timeout

Configure must be called before calling any other EventDistributor methods. EventDistributor supports reconfiguration at all times.

EventDistributorDistribute

```
uint32_t EventDistributorDistribute (const SEventDistributor * pThis, uint32_t iEventWait, uint32_t NotificationClearMask)
```

Waits for Task notification and calls event handlers matching the set notification bits.

Parameters

Type	Direction	Argument Name	Description
const SEventDistributor *	[in]	pThis	Pointer to the EventDistributor object
uint32_t	[in]	iEventWait	Timeout (in mS) when waiting for event. Value must NOT be zero.
uint32_t	[in]	NotificationClearMask	Mask of additional notification bits to clear, which is not handled by EventDistributor. EventDistributor will always clear notification bits assigned to events. This mask is additional bits to clear.

Method returns when a notification has arrived (or waiting timed out) and any pending events has been handled.

Returns

- uint32_t Task notification value. All bits handled by EventDistributor are set to zero.

event_cc_t

Public Attributes

const void *	data
This must point to a global data.	
uint16_t	command_class
uint8_t	event

Public Attribute Documentation

data

```
const void* event_cc_t::data
```

This must point to a global data.

command_class

```
uint16_t event_cc_t::command_class
```

event

```
uint8_t event_cc_t::event
```

zaf_event_distributor_cc_event_handler_map_v1_t

ZAF Event Distributor Handler Map.

Public Attributes

<code>zaf_event_distributor_cc_event_handler_handler_v1_t</code>	<code>handler</code> Custom handler.
<code>uint16_t</code>	<code>command_class</code> Table entry.
<code>uint8_t</code>	<code>version</code> Map version.

Public Attribute Documentation

handler

```
zaf_event_distributor_cc_event_handler_handler_v1_t zaf_event_distributor_cc_event_handler_map_v1_t::handler
```

Custom handler.

command_class

```
uint16_t zaf_event_distributor_cc_event_handler_map_v1_t::command_class
```

Table entry.

version

```
uint8_t zaf_event_distributor_cc_event_handler_map_v1_t::version
```

Map version.

SEventDistributor

EventDistributor object.
All content is private.

Public Attributes

uint8_t	iEventHandlerTableSize	Number of entries in pEventHandlerTable.
const EventDistributorEventHandler *	pEventHandlerTable	pointer to const array of Callback function pointers - Event handlers
void(*)	pNoEvent	Callback to handler of Notification with no event and wait for event timeout

Public Attribute Documentation

iEventHandlerTableSize

```
uint8_t SEventDistributor::iEventHandlerTableSize
```

Number of entries in pEventHandlerTable.

pEventHandlerTable

```
const EventDistributorEventHandler* SEventDistributor::pEventHandlerTable
```

pointer to const array of Callback function pointers - Event handlers

pNoEvent

```
void(*) SEventDistributor::pNoEvent) (void)
```

Callback to handler of Notification with no event and wait for event timeout

ZAF Event Handling

ZAF Event Handling

Functions

- void

[ZAF_JobHelperInit](#)(void)

Initializes the job helper.
- bool

[ZAF_JobHelperJobEnqueue](#)(uint8_t event)

Adds a given event to the job queue.
- bool

[ZAF_JobHelperJobDequeue](#)(uint8_t *pEvent)

Dequeues a job from the job queue.
- uint8_t

[ZAF_JobHelperGetNumberOfRemainingJobs](#)(void)

Returns the current number of jobs on the job queue.

Macros

- #define

JOB_QUEUE_BUFFER_SIZE 4

Defines the maximum number of jobs to queue for this module.

Function Documentation

ZAF_JobHelperInit

void ZAF_JobHelperInit (void)

Initializes the job helper.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

ZAF_JobHelperJobEnqueue

bool ZAF_JobHelperJobEnqueue (uint8_t event)

Adds a given event to the job queue.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	event	An event.

Returns

- true if the event was added to the job queue, false otherwise.

ZAF_JobHelperJobDequeue

```
bool ZAF_JobHelperJobDequeue (uint8_t * pEvent)
```

Dequeues a job from the job queue.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	pEvent	return event from the queue.

Returns

- true if a job was dequeued, false otherwise.

ZAF_JobHelperGetNumberOfRemainingJobs

```
uint8_t ZAF_JobHelperGetNumberOfRemainingJobs (void )
```

Returns the current number of jobs on the job queue.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Number of jobs on the job queue.

MfgTokens

Functions

- void

[ZW_MfgTokenModuleInit](#)(void)

Function for initializing manufacturing module.
- void

[ZW_SetMfgTokenData](#)(uint16_t token, void *data, uint8_t len)

Function for setting the manufacturing token data.
- void

[ZW_GetMfgTokenData](#)(void *data, uint16_t token, uint8_t len)

Function for Getting the manufacturing token data.
- void

[ZW_GetMfgTokenDataCountryFreq](#)(void *data)

Wrapper function for getting the manufacturing "country freq" token.
- void

[ZW_SetMfgTokenDataCountryRegion](#)(void *region)

Writes Rf Region to manufacturing token.
- void

[ZW_LockMfgTokenData](#)(void)

Lock manufacturing token storage.

Macros

- #define

TOKEN_MFG_ZWAVE_COUNTRY_FREQ_ID

0x0001
- #define

TOKEN_MFG_ZW_PRK_ID

0x0002
- #define

TOKEN_MFG_ZW_PUK_ID

0x0003
- #define

TOKEN_MFG_ZW_INITIALIZED_ID

0x0004
- #define

TOKEN_MFG_ZW_QR_CODE_ID

0x0005
- #define

TOKEN_MFG_ZWAVE_COUNTRY_FREQ_SIZE

1
- #define

TOKEN_MFG_ZW_PRK_SIZE

32
- #define

TOKEN_MFG_ZW_PUK_SIZE

32
- #define

TOKEN_MFG_ZW_INITIALIZED_SIZE

1
- #define

TOKEN_MFG_ZW_QR_CODE_SIZE

106

Function Documentation

ZW_MfgTokenModuleInit

```
void ZW_MfgTokenModuleInit (void )
```

Function for initializing manufacturing module.

Parameters

Type	Direction	Argument Name	Description
Type	Direction	Argument Name	Description
void	N/A		

ZW_SetMfgTokenData

```
void ZW_SetMfgTokenData (uint16_t token, void * data, uint8_t len)
```

Function for setting the manufacturing token data.

Parameters

Type	Direction	Argument Name	Description
uint16_t	[in]	token	Token to be set
void *	[in]	data	Pointer to object data to write
uint8_t	[in]	len	Size of object to write in bytes

ZW_GetMfgTokenData

```
void ZW_GetMfgTokenData (void * data, uint16_t token, uint8_t len)
```

Function for Getting the manufacturing token data.

Parameters

Type	Direction	Argument Name	Description
void *	[out]	data	Pointer to object data where the value will be written to.
uint16_t	[in]	token	Type of the token to get
uint8_t	[in]	len	Size of output object

ZW_GetMfgTokenDataCountryFreq

```
void ZW_GetMfgTokenDataCountryFreq (void * data)
```

Wrapper function for getting the manufacturing "country freq" token.

Parameters

Type	Direction	Argument Name	Description
void *	[out]	data	Country frequency token

ZW_SetMfgTokenDataCountryRegion

```
void ZW_SetMfgTokenDataCountryRegion (void * region)
```

Writes Rf Region to manufacturing token.

Parameters

Type	Direction	Argument Name	Description
void *	[in]	region	Region to write

ZW_LockMfgTokenData

```
void ZW_LockMfgTokenData (void )
```

Lock manufacturing token storage.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

NodeMask

NodeMask

Typedefs

```
typedef uint8_t  NODE\_MASK\_TYPE[MAX_NODEMASK_LENGTH]
```

```
typedef uint8_t  LR\_NODE\_MASK\_TYPE[MAX_LR_NODEMASK_LENGTH]
```

Functions

```
void  ZW\_NodeMaskSetBit(uint8_t *pMask, node_id_t bNodeID)
```

```
void  ZW\_NodeMaskClearBit(uint8_t *pMask, node_id_t bNodeID)  
Clear the node bit in a node bitmask.
```

```
void  ZW\_NodeMaskClear(uint8_t *pMask, uint8_t bLength)  
Clear all bits in a nodemask.
```

```
uint8_t  ZW\_NodeMaskBitsIn(const uint8_t *pMask, uint8_t bLength)  
Check and count number of bit that is set in a nodemask.
```

```
uint8_t  ZW\_NodeMaskNodeIn(const uint8_t *pMask, node_id_t bNode)  
Check if a node is in a nodemask.
```

```
node\_id\_t  ZW\_NodeMaskGetNextNodeIndex(node_id_t currentNodeIndex, uint8_t *pMask)  
Find the next NodeIndex that is set in a nodemask WARNING: this function is the only one working with indexes and not  
node IDs.
```

Macros

```
#define  NODEMASK\_NODES\_PER\_BYTE 8  
Number of nodes per node mask byte.
```

```
#define  MAX\_NODEMASK\_LENGTH (ZW_MAX_NODES / NODEMASK_NODES_PER_BYTE)  
Max length of a node mask.
```

```
#define  MAX\_LR\_NODEMASK\_LENGTH (ZW_MAX_NODES_LR / NODEMASK_NODES_PER_BYTE)  
Max length of Long Range node mask.
```

Typedef Documentation

NODE_MASK_TYPE

```
typedef uint8_t NODE\_MASK\_TYPE[MAX_NODEMASK_LENGTH] [MAX_NODEMASK_LENGTH]
```

LR_NODE_MASK_TYPE

```
typedef uint8_t LR\_NODE\_MASK\_TYPE[MAX_LR_NODEMASK_LENGTH] [MAX_LR_NODEMASK_LENGTH]
```

Function Documentation

ZW_NodeMaskSetBit

```
void ZW_NodeMaskSetBit (uint8_t * pMask, node\_id\_t bNodeID)
```

Parameters

Type	Direction	Argument Name	Description
uint8_t *	N/A	pMask	
node_id_t	N/A	bNodeID	

ZW_NodeMaskClearBit

```
void ZW_NodeMaskClearBit (uint8_t * pMask, node\_id\_t bNodeID)
```

Clear the node bit in a node bitmask.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	N/A	pMask	Pointer to Nodemask list that should be set
node_id_t	N/A	bNodeID	Node to set in nodemask

ZW_NodeMaskClear

```
void ZW_NodeMaskClear (uint8_t * pMask, uint8_t bLength)
```

Clear all bits in a nodemask.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	N/A	pMask	Pointer to Nodemask list that should be cleared
uint8_t	N/A	bLength	Length of nodemask to be cleared

ZW_NodeMaskBitsIn

```
uint8_t ZW_NodeMaskBitsIn (const uint8_t * pMask, uint8_t bLength)
```

Check and count number of bit that is set in a nodemask.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	N/A	pMask	Pointer to nodemask that should be counted
uint8_t	N/A	bLength	Length of nodemask to count

Returns

- Number of bits set in nodemask

ZW_NodeMaskNodeIn

```
uint8_t ZW_NodeMaskNodeIn (const uint8_t * pMask, node_id_t bNode)
```

Check if a node is in a nodemask.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	N/A	pMask	Pointer to nodemask to check for bNode
node_id_t	N/A	bNode	Bit number that should be checked

Returns

- 0 if not in nodemask, non-zero if found in nodemask

ZW_NodeMaskGetNextNodeIndex

```
node_id_t ZW_NodeMaskGetNextNodeIndex (node_id_t currentNodeIndex, uint8_t * pMask)
```

Find the next NodeIndex that is set in a nodemask WARNING: this function is the only one working with indexes and not node IDs.

Parameters

Type	Direction	Argument Name	Description
node_id_t	N/A	currentNodeIndex	Last NodeId found (0 for first call)
uint8_t *	N/A	pMask	Pointer to Nodemask list that should be searched

Returns

- Next NodeId from the nodemask if found, or 0 if not found.

Queue

Queue

Modules

[QueueNotifying](#)

QueueNotifying

QueueNotifying

Modules

[SQueueNotifying](#)

Enumerations

```
enum EQueueNotifyingStatus {
    EQUEUENOTIFYING_STATUS_SUCCESS = 0
    EQUEUENOTIFYING_STATUS_TIMEOUT = 1
    EQUEUENOTIFYING_STATUS_WRONG_PARAMETER = 2
}
```

Defines return values from QueueBlocking methods.

Typedefs

```
typedef enum EQueueNotifyingStatus
EQueueNotifyingStatus
    Defines return values from QueueBlocking methods.

typedef struct SQueueNotifying
SQueueNotifying
    Notifying Queue object.
```

Functions

```
void QueueNotifyingInit(SQueueNotifying *pThis, void *Queue, void *ReceiverTask, uint8_t
iTaskNotificationBitNumber)
    Initialize QueueNotifying.

EQueueNotifyingStatus QueueNotifyingSendToBack(SQueueNotifying *pThis, const uint8_t *pltem, uint32_t iTimeToWait)
    Adds item to back of queue and notifies receiver task.

EQueueNotifyingStatus QueueNotifyingSendToBackFromISR(SQueueNotifying *pThis, const uint8_t *pltem)
    Adds item to back of queue and notifies receiver task.

EQueueNotifyingStatus QueueNotifyingSendToFront(SQueueNotifying *pThis, const uint8_t *pltem, uint32_t iTimeToWait)
    Adds item to front of queue and notifies receiver task.
```

Enumeration Documentation

EQueueNotifyingStatus

EQueueNotifyingStatus	
Defines return values from QueueBlocking methods.	
Enumerator	
EQUEUENOTIFYING_STATUS_SUCCESS	Item successfully put on queue and receiver task notified.
EQUEUENOTIFYING_STATUS_TIMEOUT	Failed to put item on queue as it was full.

EQUEUENOTIFYING_STATUS_WRONG_PARAMETER	Failed to put item on queue because of a wrong input parameter.

Typedef Documentation

EQueueNotifyingStatus

```
typedef enum EQueueNotifyingStatus EQueueNotifyingStatus
```

Defines return values from QueueBlocking methods.

SQueueNotifying

```
typedef struct SQueueNotifying SQueueNotifying
```

Notifying Queue object.

All content is public.

Function Documentation

QueueNotifyingInit

```
void QueueNotifyingInit (SQueueNotifying * pThis, void * Queue, void * ReceiverTask, uint8_t iTaskNotificationBitNumber)
```

Initialize QueueNotifying.

Parameters

Type	Direction	Argument Name	Description
SQueueNotifying *	[in]	pThis	Pointer to the QueueNotifying object
void *	[in]	Queue	FreeRTOS queue handle. The Queue wrapped by QueueNotifying.
void *	[in]	ReceiverTask	FreeRTOS task handle for the task to be notified on adding items to queue.
uint8_t	[in]	iTaskNotificationBitNumber	Number defines which bit to use when notifying receiver task of item added to queue (range 0 - 31)

Must be called prior to calling any other QueueNotifying methods on a QueueNotifying object.

QueueNotifyingSendToBack

```
EQueueNotifyingStatus QueueNotifyingSendToBack (SQueueNotifying * pThis, const uint8_t * pltem, uint32_t iTimeToWait)
```

Adds item to back of queue and notifies receiver task.

Parameters

Type	Direction	Argument Name	Description
SQueueNotifying *	[in]	pThis	Pointer to the QueueNotifying object.
const uint8_t *	[in]	pltem	Pointer to item to put on Queue. Expected size is the size configured in the Queue passed on QueueNotifyingInit.
uint32_t	[in]	iTimeToWait	How many milliseconds to wait if queue is full. 0 means dont wait.

Mainly a wrapper method for FreeRTOS xQueueSendToBack(). A notification is also sent to the receiver task.

QueueNotifyingSendToBackFromISR

```
EQueueNotifyingStatus QueueNotifyingSendToBackFromISR (SQueueNotifying * pThis, const uint8_t * pltem)
```

Adds item to back of queue and notifies receiver task.

Parameters

Type	Direction	Argument Name	Description
SQueueNotifying *	[in]	pThis	Pointer to the QueueNotifying object.
const uint8_t *	[in]	pltem	Pointer to item to put on Queue. Expected size if the size configured in the Queue passed on QueueNotifyingInit.

Mainly a wrapper method for FreeRTOS xQueueSendToBackFromISR(). A notification is also sent to the receiver task.

QueueNotifyingSendToFront

```
EQueueNotifyingStatus QueueNotifyingSendToFront (SQueueNotifying * pThis, const uint8_t * pltem, uint32_t iTimeToWait)
```

Adds item to front of queue and notifies receiver task.

Parameters

Type	Direction	Argument Name	Description
SQueueNotifying *	[in]	pThis	Pointer to the QueueNotifying object.
const uint8_t *	[in]	pltem	Pointer to item to put on Queue. Expected size if the size configured in the Queue passed on QueueNotifyingInit.
uint32_t	[in]	iTimeToWait	How many milliseconds to wait if queue is full. 0 means dont wait.

Mainly a wrapper method for FreeRTOS xQueueSendToFront(). A notification is also sent to the receiver task. It is recommended to use [QueueNotifyingSendToBack\(\)](#) instead, unless its a specific wish to put item in front of other items already on the queue.

SQueueNotifying

Notifying Queue object.

All content is public.

Public Attributes

void *	Queue	FreeRTOS Queue handle.
uint8_t	iTaskNotificationBitNumber	which bit to use when notifying receiver task of pending queue item (range 0 - 31)
void *	ReceiverTask	Handle to queue receiver task where notifications are sent to.

Public Attribute Documentation

Queue

```
void* SQueueNotifying::Queue
```

FreeRTOS Queue handle.

iTaskNotificationBitNumber

```
uint8_t SQueueNotifying::iTaskNotificationBitNumber
```

which bit to use when notifying receiver task of pending queue item (range 0 - 31)

ReceiverTask

```
void* SQueueNotifying::ReceiverTask
```

Handle to queue receiver task where notifications are sent to.

Timer

Timer

Modules

[SwTimer](#)

[SwTimerLiasion](#)

SwTimer

SwTimer

Modules

- SSwTimer
- SSwTimerPrivate_t

Enumerations

```
enum ESwTimerStatus {  
    ESWTIMER_STATUS_SUCCESS = 0  
    ESWTIMER_STATUS_FAILED = 1  
}  
Defines return values from SwTimer methods.
```

Typedefs

- typedef enum
ESwTimerStatus

ESwTimerStatus
Defines return values from SwTimer methods.
- typedef struct
SSwTimerLiaison
*

SSwTimerLiaisonPtr
Typedef needed to include pointer for struct SSwTimerLiaison in struct SSwTimer, without requiring include of definition SSwTimerLiaison.
- typedef struct
SSwTimer

SSwTimer
- typedef struct
SSwTimerPrivate_
t

SSwTimerPrivate_t
Timer Liaison object.

Functions

- void

TimerSetCallback(SSwTimer *pTimer, void(*pCallback)(SSwTimer *pTimer))
Configures the callback method called on timer timeout.
- ESwTimerStatus

TimerStart(SSwTimer *pTimer, uint32_t iTimeout)
Configures Timer timeout and starts the timer.
- ESwTimerStatus

TimerStartFromISR(SSwTimer *pTimer, uint32_t iTimeout)
Configures Timer timeout and starts the timer from an Interrupt Service Routine.
- ESwTimerStatus

TimerRestart(SSwTimer *pTimer)
Restarts the timer.
- ESwTimerStatus

TimerRestartFromISR(SSwTimer *pTimer)
Restarts the timer.
- ESwTimerStatus

TimerStop(SSwTimer *pTimer)
Stops an active timer.
- ESwTimerStatus

TimerStopFromISR(SSwTimer *pTimer)

Stops an active timer
from an Interrupt
Service Routine.

- bool

TimerIsActive

(SSwTimer *pTimer)

Returns if the timer is active.
- bool

TimerHasPendingCallback

(SSwTimer *pTimer)

Returns if timer has a timeout callback pending.
- bool

TimerHasPendingCallbackFromISR

(SSwTimer *pTimer)

Returns if timer has a timeout callback pending from an Interrupt Service Routine.

- ESwTimerStatus

TimerGetMsUntilTimeout

(SSwTimer *pTimer, uint32_t refTaskTickCount, uint32_t *pMsUntilTimeout)

Queries a timer for number of milliseconds remaining before time out Method may be called on inactive timers, but in that case ESWTIMER_STATUS_FAILED will be returned and pMsUntilTimeout will not be modified.
- uint32_t

TimerGetPeriod

(SSwTimer *pTimer)

Returns the period of a freeRTOS timer.

Macros

- #define

PRIVATE_TIMER_SIZE

11

Space for holding FreeRTOS StaticEventGroup_t and EventGroupHandle_t.

Enumeration Documentation

ESwTimerStatus

ESwTimerStatus

Defines return values from SwTimer methods.

Enumerator	
ESWTIMER_STATUS_SUCCESS	ESWTIMER_STATUS_SUCCESS.
ESWTIMER_STATUS_FAILED	ESWTIMER_STATUS_FAILED.

Typedef Documentation

ESwTimerStatus

typedef enum ESwTimerStatus ESwTimerStatus

Defines return values from SwTimer methods.

SSwTimerLiaisonPtr

typedef struct SSwTimerLiaison* SSwTimerLiaisonPtr

Typedef needed to include pointer for struct [SSwTimerLiaison](#) in struct [SSwTimer](#), without requiring include of definition [SSwTimerLiaison](#).

SSwTimer

```
typedef struct SSwTimer SSwTimer
```

SSwTimerPrivate_t

```
typedef struct SSwTimerPrivate_t SSwTimerPrivate_t
```

Timer Liaison object.

All content is private.

Function Documentation

TimerSetCallback

```
void TimerSetCallback (SSwTimer * pTimer, void(*) (SSwTimer *pTimer) pCallback)
```

Configures the callback method called on timer timeout.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object
void(*) (SSwTimer *pTimer)	[in]	pCallback	Callback method of type void Callback(SSwTimer* pTimer). Argument may be NULL, in which case no callback is performed.

If called on a timer that has a pending callback, the callback will be executed on the new callback method.

TimerStart

```
ESwTimerStatus TimerStart (SSwTimer * pTimer, uint32_t iTimeout)
```

Configures Timer timeout and starts the timer.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object
uint32_t	[in]	iTimeout	Timeout value in ms. Zero is illegal.

If method is called on a timer that is already started, the timer will restart from zero and count towards the new timeout.

TimerStartFromISR

```
ESwTimerStatus TimerStartFromISR (SSwTimer * pTimer, uint32_t iTimeout)
```

Configures Timer timeout and starts the timer from an Interrupt Service Routine.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object
uint32_t	[in]	iTimeout	Timeout value in ms. Zero is illegal.

If method is called on a timer that is already started, the timer will restart from zero and count towards the new timeout.

TimerRestart

ESwTimerStatus TimerRestart (SSwTimer * pTimer)

Restarts the timer.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

Starts an inactive timer. If called on a timer that is already started, the timer will restart from zero.

TimerRestartFromISR

ESwTimerStatus TimerRestartFromISR (SSwTimer * pTimer)

Restarts the timer.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

Starts an inactive timer. If called on a timer that is already started, the timer will restart from zero.

TimerStop

ESwTimerStatus TimerStop (SSwTimer * pTimer)

Stops an active timer.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

Method may be called on inactive timers. Any pending callbacks that has not been performed yet will be cleared.

TimerStopFromISR

ESwTimerStatus TimerStopFromISR ([SSwTimer](#) * pTimer)

Stops an active timer from an Interrupt Service Routine.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

Method may be called on inactive timers. Any pending callbacks that has not been performed yet will be cleared.

TimerIsActive

bool TimerIsActive ([SSwTimer](#) * pTimer)

Returns if the timer is active.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

Active is defined as the timer counting, or having a pending callback.

TimerHasPendingCallback

bool TimerHasPendingCallback ([SSwTimer](#) * pTimer)

Returns if timer has a timeout callback pending.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

TimerHasPendingCallbackFromISR

bool TimerHasPendingCallbackFromISR ([SSwTimer](#) * pTimer)

Returns if timer has a timeout callback pending from an Interrupt Service Routine.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

TimerGetMsUntilTimeout

ESwTimerStatus TimerGetMsUntilTimeout ([SSwTimer](#) * pTimer, uint32_t refTaskTickCount, uint32_t * pMsUntilTimeout)

Queries a timer for number of milliseconds remaining before time out Method may be called on inactive timers, but in that case ESWTIMER_STATUS_FAILED will be returned and pMsUntilTimeout will not be modified.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object
uint32_t	[in]	refTaskTickCount	The xTaskGetTickCount() value to use as a reference for the timeout.
uint32_t *	[out]	pMsUntilTimeout	Pointer to variable to receive remaining time until timeout

TimerGetPeriod

uint32_t TimerGetPeriod ([SSwTimer](#) * pTimer)

Returns the period of a freertos timer.

Parameters

Type	Direction	Argument Name	Description
SSwTimer *	[in]	pTimer	Pointer to the timer object

SSwTimer

Public Attributes

uint32_t	dummy	
void *	TimerHandle	FreeRTOS timer handle.
uint8_t	Id	Identifies timer to SwTimerLiaison.
void *	ptr	User pointer, which is free to be used.
SSwTimerLiaisonPtr	pLiaison	Pointer to TimerLiaison object which timer is registered to.
void(*)	pCallback	Callback function pointer.

Public Attribute Documentation

dummy

```
uint32_t SSwTimer::dummy[PRIVATE_TIMER_SIZE]
```

TimerHandle

```
void* SSwTimer::TimerHandle
```

FreeRTOS timer handle.

Id

```
uint8_t SSwTimer::Id
```

Identifies timer to SwTimerLiaison.

ptr

```
void* SSwTimer::ptr
```

User pointer, which is free to be used.

pLiaison

```
SSwTimerLiaisonPtr SSwTimer::pLiaison
```

Pointer to TimerLiaison object which timer is registered to.

pCallback

```
void(* SSwTimer::pCallback) (struct SSwTimer *pTimer)
```

Callback function pointer.

SSwTimerPrivate_t

Timer Liaison object.
All content is private.

Public Attributes

StaticTimer_t [xTimer](#)
Static buffer to store FreeRTOS timer.

Public Attribute Documentation

xTimer

StaticTimer_t SSwTimerPrivate_t::xTimer

Static buffer to store FreeRTOS timer.

SwTimerLiasion

SwTimerLiasion

Modules

SSwTimerLiasion

Enumerations

```
enum ESwtimerLiasionStatus {
    ESWTIMERLIAISON_STATUS_SUCCESS = 0
    ESWTIMERLIAISON_STATUS_LIST_FULL
    ESWTIMERLIAISON_STATUS_ALREADY_REGISTERED
}
```

Defines return values from SwTimerLiasion methods.

Typedefs

```
typedef enum ESwtimerLiasionStatus
ESwtimerLiasionStatus;
// Defines return values from SwTimerLiasion methods.

typedef struct SSwTimerLiasion
SSwtimerLiasion;
// TimerLiasion object.
```

Functions

```
void TimerLiasionInit(SSwtimerLiasion *pThis, uint32_t iTimerPointerArraySize, SSwTimer **pTimerPointerArray,
uint8_t iTaskNotificationBitNumber, void *ReceiverTask)
// Initialize TimerLiasion.

void TimerLiasionSetReceiverTask(SSwtimerLiasion *pThis, void *ReceiverTask)
// Configures the Receiver task of callbacks.

ESwtimerLiasionStatus TimerLiasionRegister(SSwtimerLiasion *pThis, SSwTimer *pTimer, bool bAutoReload, void(*pCallback)
(SSwtimer *pTimer))
// Register a SwTimer to a TimerLiasion.

void TimerLiasionNotificationHandler(SSwtimerLiasion *pThis)
// Must be called from requested callback task when the task notification bit assigned to TimerLiasion is set.

void TimerLiasionClearPendingTimerEvent(SSwtimerLiasion *pThis, uint32_t TimerId)
// Method provided for the SwTimers to call.

void TimerLiasionClearPendingTimerEventFromISR(SSwtimerLiasion *pThis, uint32_t TimerId)
// Method provided for the SwTimers to call from ISR.

bool TimerLiasionHasPendingTimerEvent(SSwtimerLiasion *pThis, uint32_t TimerId)
// Method provided for the SwTimers to call.

bool TimerLiasionHasPendingTimerEventFromISR(SSwtimerLiasion *pThis, uint32_t TimerId)
// Method provided for the SwTimers to call from ISR.
```


Macros

```
#define PRIVATE_STATIC_TIMER_SIZE 9
    Space for holding FreeRTOS StaticTimer_t.
```

Enumeration Documentation

ESwTimerLiaisonStatus

ESwTimerLiaisonStatus

Defines return values from SwTimerLiaison methods.

Enumerator	
ESWTIMERLIAISON_STATUS_SUCCESS	ESWTIMERLIAISON_STATUS_SUCCESS.
ESWTIMERLIAISON_STATUS_LIST_FULL	ESWTIMERLIAISON_STATUS_LIST_FULL.
ESWTIMERLIAISON_STATUS_ALREADY_REGISTRERED	ESWTIMERLIAISON_STATUS_ALREADY_REGISTRERED.

Typedef Documentation

ESwTimerLiaisonStatus

```
typedef enum ESwTimerLiaisonStatus ESwTimerLiaisonStatus
```

Defines return values from SwTimerLiaison methods.

SSwTimerLiaison

```
typedef struct SSwTimerLiaison SSwTimerLiaison
```

TimerLiaison object.

All content is private.

Function Documentation

TimerLiaisonInit

```
void TimerLiaisonInit (SSwTimerLiaison * pThis, uint32_t iTimerPointerArraySize, SSwTimer ** pTimerPointerArray, uint8_t iTaskNotificationBitNumber, void * ReceiverTask)
```

Initialize TimerLiaison.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object

Type	Direction	Argument Name	Description
uint32_t	[in]	iTimerPointerArraySize	Number of entries in provided SwTimer pointer array (range 0 - 24)
SSwTimer**	[in]	pTimerPointerArray	Pointer to provided array of pointers to SwTimer objects
uint8_t	[in]	iTaskNotificationBitNumber	Number defines which bit to use when notifying receiver task of pending timer event (range 0 - 31)
void *	[in]	ReceiverTask	Handle to FreeRTOS task that Timer callbacks Will be performed in. Argument may be NULL if task handle is not available at Liaison init time, but ReceiverTask must then be set with TimerLiaisonSetReceiverTask before any registered Timer times out.

TimerLiaisonSetReceiverTask

```
void TimerLiaisonSetReceiverTask (SSwTimerLiaison * pThis, void * ReceiverTask)
```

Configures the Receiver task of callbacks.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object
void *	[in]	ReceiverTask	Handle to FreeRTOS task that Timer callbacks will be performed in.

Provided as the TimerLiaison may be needed for registrering SwTimers before the receiver task is created, meaning TimerLiaison init is called prior to the Receiver task handle being available. Though not the intention, the receiver task can be changed run time.

TimerLiaisonRegister

```
ESwTimerLiaisonStatus TimerLiaisonRegister (SSwTimerLiaison * pThis, SSwTimer * pTimer, bool bAutoReload, void(*) (SSwTimer *pTimer) pCallback)
```

Register a SwTimer to a TimerLiaison.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object
SSwTimer *	[in]	pTimer	Pointer to the SwTimer object to register
bool	[in]	bAutoReload	Enable timer auto reload on timeout. Configuration of AutoReload cannot be changed after registration.
void(*) (SSwTimer *pTimer)	[in]	pCallback	Callback method of type void Callback(SSwTimer* pTimer). Argument may be NULL, in which case no callback is performed.

Initial SwTimer configuration is also performed.

Method creates a static FreeRTOS timer from the SwTimer object passed as argument. An SwTimer can NOT be registered to several TimerLiaisons. If calling method TimerLiaisonRegister on the same SwTimer object multiple times, all but the first call will be ignored. Unregistered timers are expected to be zero initialized SSwTimer structs.

TimerLiaisonNotificationHandler

```
void TimerLiaisonNotificationHandler (SSwTimerLiaison * pThis)
```

Must be called from requested callback task when the task notification bit assigned to TimerLiaison is set.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object

Method will perform pending callbacks. There is no side effects from calling method when assigned task notification bit was not set.

TimerLiaisonClearPendingTimerEvent

```
void TimerLiaisonClearPendingTimerEvent (SSwTimerLiaison * pThis, uint32_t TimerId)
```

Method provided for the SwTimers to call.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object
uint32_t	[in]	TimerId	Which timer to clear pending events of.

Method clears pending timeout callback (if one is pending).

TimerLiaisonClearPendingTimerEventFromISR

```
void TimerLiaisonClearPendingTimerEventFromISR (SSwTimerLiaison * pThis, uint32_t TimerId)
```

Method provided for the SwTimers to call from ISR.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object
uint32_t	[in]	TimerId	Which timer to clear pending events of.

Method clears pending timeout callback (if one is pending).

TimerLiaisonHasPendingTimerEvent

```
bool TimerLiaisonHasPendingTimerEvent (SSwTimerLiaison * pThis, uint32_t TimerId)
```

Method provided for the SwTimers to call.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object
uint32_t	[in]	TimerId	Which timer to check for pending callback.

Method returns if a timer has a pending timeout callback.

TimerLiaisonHasPendingTimerEventFromISR

```
bool TimerLiaisonHasPendingTimerEventFromISR (SSwTimerLiaison * pThis, uint32_t TimerId)
```

Method provided for the SwTimers to call from ISR.

Parameters

Type	Direction	Argument Name	Description
SSwTimerLiaison *	[in]	pThis	Pointer to the TimerLiaison object
uint32_t	[in]	TimerId	Which timer to check for pending callback.

Method returns if a timer has a pending timeout callback.

SSwTimerLiaison

TimerLiaison object.

All content is private.

Public Attributes

uint32_t	dummy	
uint8_t	iTimerCount	Keeps count of added timers.
uint8_t	iTimerListSize	Size of pTimerList.
struct SSwTimer **	pTimerList	Pointer to array of pointers to SwTimer objects.
uint8_t	iTaskNotificationBitNumber	which bit to use when notifying receiver task of pending timer event(range 0 - 31)
void *	ReceiverTask	Handle to task in which callbacks should be performed.

Public Attribute Documentation

dummy

```
uint32_t SSwTimerLiaison::dummy[PRIVATE_STATIC_TIMER_SIZE]
```

iTimerCount

```
uint8_t SSwTimerLiaison::iTimerCount
```

Keeps count of added timers.

iTimerListSize

```
uint8_t SSwTimerLiaison::iTimerListSize
```

Size of pTimerList.

pTimerList

```
struct SSwTimer** SSwTimerLiaison::pTimerList
```

Pointer to array of pointers to SwTimer objects.

iTaskNotificationBitNumber

```
uint8_t SSwTimerLiaison::iTaskNotificationBitNumber
```

which bit to use when notifying receiver task of pending timer event(range 0 - 31)

ReceiverTask

```
void* SSwTimerLiaison::ReceiverTask
```

Handle to task in which callbacks should be performed.

SyncEvent

SyncEvent

Modules

- [USyncEventFunctor](#)
- [SSyncEvent](#)
- [USyncEventRetFunctor](#)
- [SSyncEventRet](#)
- [USyncEventArg1Functor](#)
- [SSyncEventArg1](#)
- [USyncEventArg2Functor](#)
- [SSyncEventArg2](#)

Typedefs

<code>typedef union USyncEventFunctor or</code>	USyncEventFunctor
<code>typedef struct SSyncEvent</code>	SSyncEvent
<code>typedef union USyncEventRetFunctor</code>	USyncEventRetFunctor
<code>typedef struct SSyncEventRet</code>	SSyncEventRet
<code>typedef union USyncEventArg1Functor</code>	USyncEventArg1Functor
<code>typedef struct SSyncEventArg1</code>	SSyncEventArg1
<code>typedef union USyncEventArg2Functor</code>	USyncEventArg2Functor
<code>typedef struct SSyncEventArg2</code>	SSyncEventArg2

Functions

<code>bool</code>	SyncEventsIsBound (const SSyncEvent *pThis) Returns if SyncEvent is bound.
<code>void</code>	SyncEventUnbind (SSyncEvent *pThis) Unbind the SyncEvent.

void	SyncEventBind (SSyncEvent *pThis, void(*pFunction)(void)) Bind the SyncEvent to a function.
void	SyncEventBindObject (SSyncEvent *pThis, void(*pFunction)(void *), void *pObject) Bind the SyncEvent to a "class" function.
void	SyncEventInvoke (const SSyncEvent *pThis) Invoke the SyncEvent.
bool	SyncEventRetIsBound (const SSyncEventRet *pThis) Returns if SyncEvent is bound.
void	SyncEventRetUnbind (SSyncEventRet *pThis) Unbind the SyncEvent.
void	SyncEventRetBind (SSyncEventRet *pThis, uint32_t(*pFunction)(void)) Bind the SyncEvent to a function.
void	SyncEventRetBindObject (SSyncEventRet *pThis, uint32_t(*pFunction)(void *), void *pObject) Bind the SyncEvent to a "class" function.
uint32_t	SyncEventRetInvoke (const SSyncEventRet *pThis) Invoke the SyncEvent.
void	SyncEventRetSetDefault (SSyncEventRet *pThis, uint32_t DefaultReturnValue) Configure default return.
bool	SyncEventArg1IsBound (const SSyncEventArg1 *pThis) Returns if SyncEvent is bound.
void	SyncEventArg1Unbind (SSyncEventArg1 *pThis) Unbind the SyncEvent.
void	SyncEventArg1Bind (SSyncEventArg1 *pThis, void(*pFunction)(uint32_t)) Bind the SyncEvent to a function.
void	SyncEventArg1BindObject (SSyncEventArg1 *pThis, void(*pFunction)(void *, uint32_t), void *pObject) Bind the SyncEvent to a "class" function.
void	SyncEventArg1Invoke (const SSyncEventArg1 *pThis, uint32_t Arg1) Invoke the SyncEvent.
bool	SyncEventArg2IsBound (const SSyncEventArg2 *pThis) Returns if SyncEvent is bound.
void	SyncEventArg2Unbind (SSyncEventArg2 *pThis) Unbind the SyncEvent.
void	SyncEventArg2Bind (SSyncEventArg2 *pThis, void(*pFunction)(uint32_t, uint32_t)) Bind the SyncEvent to a function.
void	SyncEventArg2BindObject (SSyncEventArg2 *pThis, void(*pFunction)(void *, uint32_t, uint32_t), void *pObject) Bind the SyncEvent to a "class" function.
void	SyncEventArg2Invoke (const SSyncEventArg2 *pThis, uint32_t Arg1, uint32_t Arg2) Invoke the SyncEvent.

Typedef Documentation

USyncEventFunctor


```
typedef union USyncEventFunctor USyncEventFunctor
```

SSyncEvent

```
typedef struct SSyncEvent SSyncEvent
```

USyncEventRetFunctor

```
typedef union USyncEventRetFunctor USyncEventRetFunctor
```

SSyncEventRet

```
typedef struct SSyncEventRet SSyncEventRet
```

USyncEventArg1Functor

```
typedef union USyncEventArg1Functor USyncEventArg1Functor
```

SSyncEventArg1

```
typedef struct SSyncEventArg1 SSyncEventArg1
```

USyncEventArg2Functor

```
typedef union USyncEventArg2Functor USyncEventArg2Functor
```

SSyncEventArg2

```
typedef struct SSyncEventArg2 SSyncEventArg2
```

Function Documentation

SyncEventsBound

```
bool SyncEventsBound (const SSyncEvent * pThis)
```

Returns if SyncEvent is bound.

Parameters

Type	Direction	Argument Name	Description
const SSyncEvent *	N/A	pThis	SyncEvent object structure.

SyncEventUnbind

```
void SyncEventUnbind (SSyncEvent * pThis)
```

Unbind the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
SSyncEvent *	N/A	pThis	SyncEvent object structure.

Unbinds the SyncEvent. Once unbound there will be no action when invoked. It is crucial that Bind, BindObject or Unbind is called before first invoke (my kingdom for a constructor).

SyncEventBind

```
void SyncEventBind (SSyncEvent * pThis, void(*) (void) pFunction)
```

Bind the SyncEvent to a function.

Parameters

Type	Direction	Argument Name	Description
SSyncEvent *	N/A	pThis	SyncEvent object structure.
void(*) (void)	N/A	pFunction	Function pointer to function of type: void Foo(void)

Binds the SyncEvent to a function. Once bound invocation will call function.

SyncEventBindObject

```
void SyncEventBindObject (SSyncEvent * pThis, void(*) (void *) pFunction, void * pObject)
```

Bind the SyncEvent to a "class" function.

Parameters

Type	Direction	Argument Name	Description
SSyncEvent *	N/A	pThis	SyncEvent object structure.
void(*) (void *)	N/A	pFunction	Function pointer to function of type: void Foo(FooObject*)
void *	N/A	pObject	Object pointer to pass as argument to bound function

Binds the SyncEvent to a function wich takes an object pointer or similar. Once bound invocation will call function with pointer as argument.

SyncEventInvoke

```
void SyncEventInvoke (const SSyncEvent * pThis)
```

Invoke the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
const SSyncEvent *	N/A	pThis	SyncEvent object structure.

Invokes the SyncEvent. If the SyncEvent is not bound, nothing happens. If bound invocation will call function or Function with pointer as argument.

SyncEventRetIsBound

```
bool SyncEventRetIsBound (const SSyncEventRet * pThis)
```

Returns if SyncEvent is bound.

Parameters

Type	Direction	Argument Name	Description
const SSyncEventRet *	N/A	pThis	SyncEvent object structure.

SyncEventRetUnbind

```
void SyncEventRetUnbind (SSyncEventRet * pThis)
```

Unbind the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
SSyncEventRet *	N/A	pThis	SyncEvent object structure.

Unbinds the SyncEvent. Once unbound there will be no action when invoked. It is crucial that Bind, BindObject or Unbind is called before first invoke (my kingdom for a constructor).

SyncEventRetBind

```
void SyncEventRetBind (SSyncEventRet * pThis, uint32_t(*) (void) pFunction)
```

Bind the SyncEvent to a function.

Parameters

Type	Direction	Argument Name	Description
SSyncEventRet *	N/A	pThis	SyncEvent object structure.
uint32_t(*) (void)	N/A	pFunction	Function pointer to function of type: uint32_t Foo(void)

Binds the SyncEvent to a function. Once bound invocation will call function.

SyncEventRetBindObject

```
void SyncEventRetBindObject (SSyncEventRet * pThis, uint32_t(*) (void *) pFunction, void * pObject)
```

Bind the SyncEvent to a "class" function.

Parameters

Type	Direction	Argument Name	Description
SSyncEventRet *	N/A	pThis	SyncEvent object structure.
uint32_t(*) (void *)	N/A	pFunction	Function pointer to function of type: uint32_t Foo(FooObject*)
void *	N/A	pObject	Object pointer to pass as argument to bound function

Binds the SyncEvent to a function wich takes an object pointer or similar. Once bound invocation will call function with pointer as argument.

SyncEventRetInvoke

```
uint32_t SyncEventRetInvoke (const SSyncEventRet * pThis)
```

Invoke the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
const SSyncEventRet *	N/A	pThis	SyncEvent object structure.

Invokes the SyncEvent. If theSyncEvent is not bound, nothing happens. If bound invocation will call function or Function with pointer as argument.

SyncEventRetSetDefault

```
void SyncEventRetSetDefault (SSyncEventRet * pThis, uint32_t DefaultReturnValue)
```

Configure default return.

Parameters

Type	Direction	Argument Name	Description
SSyncEventRet *	N/A	pThis	SyncEvent object structure.
uint32_t	N/A	DefaultReturnValue	Value returned when SyncEvent is invoked while unbound.

Configure the default return value that is returned when the SyncEvent is invoked while unbound.

SyncEventArg1IsBound

```
bool SyncEventArg1IsBound (const SSyncEventArg1 * pThis)
```

Returns if SyncEvent is bound.

Parameters

Type	Direction	Argument Name	Description
const SSyncEventArg1 *	N/A	pThis	SyncEvent object structure.

SyncEventArg1Unbind

```
void SyncEventArg1Unbind (SSyncEventArg1 * pThis)
```

Unbind the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
SSyncEventArg1 *	N/A	pThis	SyncEvent object structure.

Unbinds the SyncEvent. Once unbound there will be no action when invoked. It is crucial that Bind, BindObject or Unbind is called before first invoke (my kingdom for a constructor).

SyncEventArg1Bind

```
void SyncEventArg1Bind (SSyncEventArg1 * pThis, void(*) (uint32_t) pFunction)
```

Bind the SyncEvent to a function.

Parameters

Type	Direction	Argument Name	Description
SSyncEventArg1 *	N/A	pThis	SyncEvent object structure.
void(*) (uint32_t)	N/A	pFunction	Function pointer to function of type: void Foo(uint32_t)

Binds the SyncEvent to a function. Once bound invocation will call function.

SyncEventArg1BindObject

```
void SyncEventArg1BindObject (SSyncEventArg1 * pThis, void(*) (void *, uint32_t) pFunction, void * pObject)
```

Bind the SyncEvent to a "class" function.

Parameters

Type	Direction	Argument Name	Description
SSyncEventArg1 *	N/A	pThis	SyncEvent object structure.
void(*) (void *, uint32_t)	N/A	pFunction	Function pointer to function of type: void Foo(FooObject*, uint32_t)
void *	N/A	pObject	Object pointer to pass as argument to bound function

Binds the SyncEvent to a function wich takes an object pointer or similar. Once bound invocation will call function with pointer as argument.

SyncEventArg1Invoke

```
void SyncEventArg1Invoke (const SSyncEventArg1 * pThis, uint32_t Arg1)
```

Invoke the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
const SSyncEventArg1 *	N/A	pThis	SyncEvent object structure.
uint32_t	N/A	Arg1	

Invokes the SyncEvent. If theSyncEvent is not bound, nothing happens. If bound invocation will call function or Function with pointer as argument.

SyncEventArg2IsBound

```
bool SyncEventArg2IsBound (const SSyncEventArg2 * pThis)
```

Returns if SyncEvent is bound.

Parameters

Type	Direction	Argument Name	Description
const SSyncEventArg2 *	N/A	pThis	SyncEvent object structure.

SyncEventArg2Unbind

```
void SyncEventArg2Unbind (SSyncEventArg2 * pThis)
```

Unbind the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
SSyncEventArg2 *	N/A	pThis	SyncEvent object structure.

Unbinds the SyncEvent. Once unbound there will be no action when invoked. It is crucial that Bind, BindObject or Unbind is called before first invoke (my kingdom for a constructor).

SyncEventArg2Bind

```
void SyncEventArg2Bind (SSyncEventArg2 * pThis, void(*) (uint32_t, uint32_t) pFunction)
```

Bind the SyncEvent to a function.

Parameters

Type	Direction	Argument Name	Description
SSyncEventArg2 *	N/A	pThis	SyncEvent object structure.
void(*) (uint32_t, uint32_t)	N/A	pFunction	Function pointer to function of type: void Foo(uint32_t)

Binds the SyncEvent to a function. Once bound invocation will call function.

SyncEventArg2BindObject

```
void SyncEventArg2BindObject (SSyncEventArg2 * pThis, void(*) (void *, uint32_t, uint32_t) pFunction, void * pObject)
```

Bind the SyncEvent to a "class" function.

Parameters

Type	Direction	Argument Name	Description
SSyncEventArg2 *	N/A	pThis	SyncEvent object structure.
void(*) (void *, uint32_t, uint32_t)	N/A	pFunction	Function pointer to function of type: void Foo(FooObject*, uint32_t, uint32_t)
void *	N/A	pObject	Object pointer to pass as argument to bound function

Binds the SyncEvent to a function wich takes an object pointer or similar. Once bound invocation will call function with pointer as argument.

SyncEventArg2Invoke

```
void SyncEventArg2Invoke (const SSyncEventArg2 * pThis, uint32_t Arg1, uint32_t Arg2)
```

Invoke the SyncEvent.

Parameters

Type	Direction	Argument Name	Description
const SSyncEventArg2 *	N/A	pThis	SyncEvent object structure.
uint32_t	N/A	Arg1	
uint32_t	N/A	Arg2	

Invokes the SyncEvent. If theSyncEvent is not bound, nothing happens. If bound invocation will call function or Function with pointer as argument.

USyncEventFunctor

Public Attributes

```
void(* pFunction)
void(* pClassFunction)
```

Public Attribute Documentation

pFunction

```
void(* USyncEventFunctor::pFunction) (void)
```

pClassFunction

```
void(* USyncEventFunctor::pClassFunction) (void *)
```


SSyncEvent

Public Attributes

USyncEventFunc
or
void * pObject

Public Attribute Documentation

uFunctor

USyncEventFuncor SSyncEvent::uFunctor

pObject

void* SSyncEvent::pObject

USyncEventRetFunctor

Public Attributes

- uint32_t(* pFunction
- uint32_t(* pClassFunction

Public Attribute Documentation

pFunction

```
uint32_t(* USyncEventRetFunctor::pFunction) (void)
```

pClassFunction

```
uint32_t(* USyncEventRetFunctor::pClassFunction) (void *)
```

SSyncEventRet

Public Attributes

USyncEventRetF	uFunctor
unctor	
void *	pObject
uint32_t	Default

Public Attribute Documentation

uFunctor

USyncEventRetFunctor SSyncEventRet::uFunctor

pObject

void* SSyncEventRet::pObject

Default

uint32_t SSyncEventRet::Default

USyncEventArg1Functor

Public Attributes

```
void(* pFunction)
void(* pClassFunction)
```

Public Attribute Documentation

pFunction

```
void(* USyncEventArg1Functor::pFunction) (uint32_t)
```

pClassFunction

```
void(* USyncEventArg1Functor::pClassFunction) (void *, uint32_t)
```

SSyncEventArg1

Public Attributes

USyncEventArg1F

uFunctor

unctor

void *

pObject

Public Attribute Documentation

uFunctor

USyncEventArg1Functor SSyncEventArg1::uFunctor

pObject

void* SSyncEventArg1::pObject

USyncEventArg2Functor

Public Attributes

```
void(* pFunction)
void(* pClassFunction)
```

Public Attribute Documentation

pFunction

```
void(* USyncEventArg2Functor::pFunction) (uint32_t, uint32_t)
```

pClassFunction

```
void(* USyncEventArg2Functor::pClassFunction) (void *, uint32_t, uint32_t)
```

SSyncEventArg2

Public Attributes

USyncEventArg2
Functor

uFunctor

void *

pObject

Public Attribute Documentation

uFunctor

USyncEventArg2Functor SSyncEventArg2::uFunctor

pObject

void* SSyncEventArg2::pObject

Zpal

Zpal

Z-Wave Platform Abstraction Layer.

Modules

[Zpal-bootloader](#)

[Zpal-entropy](#)

[Zpal-init](#)

[Zpal-misc](#)

[Zpal-nvm](#)

[Zpal-power-manager](#)

[Zpal-radio](#)

[Zpal-radio-utils](#)

[Zpal-retention-register](#)

[Zpal-status](#)

[Zpal-uart](#)

[Zpal-watchdog](#)

Zpal-bootloader

Zpal-bootloader

Defines a platform abstraction layer for the Z-Wave bootloader.

How to use the bootloader API

The ZPAL bootloader API is required if an application would like to store and use a new firmware image.

[zpal_bootloader_init\(\)](#) and [zpal_bootloader_get_info\(\)](#) must be implemented as Z-Wave always invokes these. The remaining API functions are optional in case a bootloader is not present in the ZPAL implementation.

The following outlines an example of use:

1. To enable the bootloader functionality, invoke [zpal_bootloader_init\(\)](#).
2. [zpal_bootloader_get_info\(\)](#) must be invoked to check whether a bootloader is present.
3. For every received fragment, [zpal_bootloader_write_data\(\)](#) must be invoked to store the new firmware image.
4. When all fragments are received, [zpal_bootloader_verify_image\(\)](#) must be invoked to verify the integrity of the received firmware image.
5. If the received firmware image is valid, invoke [zpal_bootloader_reboot_and_install\(\)](#) install and boot from the new firmware image.

Requirements:

- The ZPAL bootloader requires that a storage is available for storing a received firmware image. Z-Wave has no requirements on how and where the image is stored.

Modules

[zpal_bootloader_info_t](#)

[Zpal-bootloader-info-version](#)

Enumerations

```
enum    zpal_bootloader_type_t {  
        ZPAL_BOOTLOADER_NOT_PRESENT = 0  
        ZPAL_BOOTLOADER_PRESENT = 1  
    }  
    Type of bootloader.
```

Functions

```
void    zpal_bootloader_get_info(zpal_bootloader_info_t *info)  
        //zpal-bootloader-info-version  
  
zpal_status_t  zpal_bootloader_init(void)  
        Initialize components of the bootloader so the app can use the interface.  
  
void    zpal_bootloader_reboot_and_install(void)  
        Reboot into the bootloader to install something.
```

- zpal_status_t

zpal_bootloader_verify_image(void)
Verify that the image is valid.
- zpal_status_t

zpal_bootloader_write_data(uint32_t offset, uint8_t *data, uint16_t length)
Writes a fragment of a firmware image to the bootloader storage.
- bool

zpal_bootloader_is_first_boot(bool *updated_successfully)
Returns whether the software is running on the first boot after a new firmware image was written.
- void

zpal_bootloader_reset_page_counters(void)
Resets the bootloader page counters.

Macros

#define

ZPAL_BOOTLOADER_CAPABILITY_STORAGE (1 << 0)
Bootloader has the capability of storing data in an internal or external storage medium.

Enumeration Documentation

zpal_bootloader_type_t

zpal_bootloader_type_t		
Type of bootloader.		
Enumerator		
ZPAL_BOOTLOADER_NOT_PRESENT	No bootloader present.	
ZPAL_BOOTLOADER_PRESENT	Bootloader present.	

Function Documentation

zpal_bootloader_get_info

void zpal_bootloader_get_info (zpal_bootloader_info_t * info)			
//zpal-bootloader-info-version			
Parameters			
Type	Direction	Argument Name	Description
zpal_bootloader_info_t *	[out]	info	Pointer to the bootloader information struct.
Get information about the bootloader on this device. The information returned is fetched from the main bootloader information table.			

zpal_bootloader_init

zpal_status_t zpal_bootloader_init (void)			
Initialize components of the bootloader so the app can use the interface.			
Parameters			

Type	Direction	Argument Name	Description
void	N/A		

This typically includes initializing serial peripherals for communication with external SPI flashes, and so on.

Returns

- Error code: [ZPAL_STATUS_OK](#) on success, else [ZPAL_STATUS_FAIL](#)

zpal_bootloader_reboot_and_install

```
void zpal_bootloader_reboot_and_install (void )
```

Reboot into the bootloader to install something.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

If there is a storage component and a slot is marked for bootload, install the image in that slot after verifying it. If a communication component is present, open the communication channel and receive an image to be installed.

zpal_bootloader_verify_image

```
zpal_status_t zpal_bootloader_verify_image (void )
```

Verify that the image is valid.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- [ZPAL_STATUS_OK](#) if the image is valid, else [ZPAL_STATUS_FAIL](#).

zpal_bootloader_write_data

```
zpal_status_t zpal_bootloader_write_data (uint32_t offset, uint8_t * data, uint16_t length)
```

Writes a fragment of a firmware image to the bootloader storage.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	offset	Offset from the storage start.
uint8_t *	[in]	data	Pointer to fragment of firmware image.
uint16_t	[in]	length	Length of firmware image fragment.

Returns

[ZPAL_STATUS_OK](#) on success, and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_bootloader_is_first_boot

```
bool zpal_bootloader_is_first_boot (bool * updated_successfully)
```

Returns whether the software is running on the first boot after a new firmware image was written.

Parameters

Type	Direction	Argument Name	Description
bool *	[in]	updated_successfully	Set to true if update was successful, false otherwise. Valid only when it's the first boot.

Returns

- True if it's the first boot, false otherwise.

zpal_bootloader_reset_page_counters

```
void zpal_bootloader_reset_page_counters (void )
```

Resets the bootloader page counters.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Used when initiating an upgrade to jump back to the start of the storage slot.

zpal_bootloader_info_t

Information about the current bootloader.

Public Attributes

<code>zpal_bootloader_t</code>	<code>type</code>
<code>type_t</code>	The type of bootloader.
<code>uint32_t</code>	<code>version</code>
	Version number of the bootloader. See Zpal-bootloader-info-version for masking.
<code>uint32_t</code>	<code>capabilities</code>
	Capability mask for the bootloader.

Public Attribute Documentation

type

`zpal_bootloader_type_t zpal_bootloader_info_t::type`

The type of bootloader.

version

`uint32_t zpal_bootloader_info_t::version`

Version number of the bootloader. See [Zpal-bootloader-info-version](#) for masking.

capabilities

`uint32_t zpal_bootloader_info_t::capabilities`

Capability mask for the bootloader.

Zpal-bootloader-info-version

Zpal-bootloader-info-version

Defines a platform abstraction layer for the Z-Wave bootloader.

Macros

```
#define ZPAL_BOOTLOADER_VERSION_MAJOR_SHIFT (24U)
Bootloader version major version shift value.

#define ZPAL_BOOTLOADER_VERSION_MINOR_SHIFT (16U)
Bootloader version minor version shift value.

#define ZPAL_BOOTLOADER_VERSION_BUGFIX_SHIFT (0U)
Bootloader version bugfix version shift value.

#define ZPAL_BOOTLOADER_VERSION_MAJOR_MASK (0xFF000000U)
Bootloader version major version mask.

#define ZPAL_BOOTLOADER_VERSION_MINOR_MASK (0x00FF0000U)
Bootloader version minor version mask.

#define ZPAL_BOOTLOADER_VERSION_BUGFIX_MASK (0x0000FFFFU)
Bootloader version bugfix version mask.
```

Zpal-entropy

Zpal-entropy

Defines a platform abstraction layer for the Z-Wave random number generation.

How to use the entropy API:

All the ZPAL entropy APIs are required to be implemented.

The following outlines an example of use:

- 1. Initialize module with `zpal_entropy_init()`.
- 2. Invoke `zpal_get_random_data()` or `zpal_get_pseudo_random()` to get generated data.

Requirements:

- `zpal_get_random_data()` generates random data from entropy sources available on platform. Since collecting entropy from hardware may take some time, Z-Wave SDK shall invoke it only once during initialization of PRNG.
- `zpal_get_pseudo_random()` is used for non-security related operations such as generating session IDs or calculating delays. Therefore, the execution speed is highly preferred over true randomness of the output.

Functions

void	<code>zpal_entropy_init(void)</code> Initialize entropy module.
<code>zpal_status_t</code>	<code>zpal_get_random_data(uint8_t *data, size_t len)</code> Generate true random array of bytes based on the given length and fill a data buffer.
uint8_t	<code>zpal_get_pseudo_random(void)</code> Returns a pseudo random byte.

Function Documentation

zpal_entropy_init

```
void zpal_entropy_init (void )
```

Initialize entropy module.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_get_random_data

```
zpal_status_t zpal_get_random_data (uint8_t * data, size_t len)
```

Generate true random array of bytes based on the given length and fill a data buffer.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	data	A pointer to where the random array of bytes are stored in data buffer. It MUST NOT be NULL.
size_t	[in]	len	The requested size of the random array of bytes that shall be stored in the data buffer.

Note that this function may shut down the radio while getting the data.

Returns

- [ZPAL_STATUS_OK](#) if the random array of bytes are generated and stored in the data buffer. [ZPAL_STATUS_OK](#) if the random array of bytes are not generated.

zpal_get_pseudo_random

```
uint8_t zpal_get_pseudo_random (void )
```

Returns a pseudo random byte.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Pseudo random byte.

Zpal-init

Zpal-init

Defines a platform abstraction layer for the Z-Wave initialization.

How to use the initialization API

The ZPAL initialization API is required to validate if application can be run on specific chip.

`zpal_init_is_valid()` must be implemented as Z-Wave always invokes it. `zpal_init_invalidate()` will be invoked only if `zpal_init_is_valid()` return false. `zpal_system_startup()` is an externally defined and implemented by Z-Wave SDK.

The following outlines an example of use:

1. Initialize hardware.
2. Invoke `zpal_system_startup()` at the end of `main()`.
3. During Z-Wave SDK initialization `zpal_init_is_valid()` will be invoked.
4. If `zpal_init_is_valid()` returns false, `zpal_init_invalidate()` will be invoked.

Requirements:

- The ZPAL initialization requires that `zpal_system_startup()` is invoked at the end of `main()` after hardware initialization is done. (check `main()` from PAL/templates/main.c)
- Function `zpal_init_invalidate()` never exits. It must setup the hardware so that the chip stops working.

Enumerations

```
enum    zpal_reset_reason_t {
        ZPAL_RESET_REASON_PIN = 0
        ZPAL_RESET_REASON_DEEP_SLEEP_WUT = 1
        ZPAL_RESET_REASON_WATCHDOG = 3
        ZPAL_RESET_REASON_DEEP_SLEEP_EXT_INT = 4
        ZPAL_RESET_REASON_POWER_ON = 5
        ZPAL_RESET_REASON_SOFTWARE = 7
        ZPAL_RESET_REASON_BROWNOUT = 9
        ZPAL_RESET_REASON_TAMPER = 10
        ZPAL_RESET_REASON_OTHER = 0xFF
    }
Reset reason.
```

Typedefs

```
typedef enum    zpal_reset_reason_t
zpal_reset_reason_t
Reset reason.
```

Functions

```
bool    zpal_init_is_valid(uint8_t generic_type, uint8_t specific_type)
Returns whether generic and specific is a valid combination for current chip.

void    zpal_init_invalidate(void)
Setup the hardware so that the chip stops working.
```

```
zpal_reset_reason_t zpal_get_reset_reason(void)
    Get reset reason.

void zpal_system_startup(zpal_reset_reason_t reset_reason)
    System startup, implemented by Z-Wave SDK.
```

Enumeration Documentation

zpal_reset_reason_t

```
zpal_reset_reason_t
```

Reset reason.

Enumerator	
ZPAL_RESET_REASON_PIN	Reset triggered by reset pin.
ZPAL_RESET_REASON_DEEP_SLEEP_WUT	Reset triggered by wake up by timer from deep sleep state.
ZPAL_RESET_REASON_WATCHDOG	Reset triggered by watchdog.
ZPAL_RESET_REASON_DEEP_SLEEP_EXT_INT	Reset triggered by external interrupt event in deep sleep state.
ZPAL_RESET_REASON_POWER_ON	Reset triggered by power on.
ZPAL_RESET_REASON_SOFTWARE	Reset triggered by software.
ZPAL_RESET_REASON_BROWNOUT	Reset triggered by brownout circuit.
ZPAL_RESET_REASON_TAMPER	Reset triggered by a tamper attempt.
ZPAL_RESET_REASON_OTHER	Reset triggered by none of the above.

Typedef Documentation

zpal_reset_reason_t

```
typedef enum zpal_reset_reason_t zpal_reset_reason_t
```

Reset reason.

Function Documentation

zpal_init_is_valid

```
bool zpal_init_is_valid (uint8_t generic_type, uint8_t specific_type)
```

Returns whether generic and specific is a valid combination for current chip.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	generic_type	Generic type of the running application.
uint8_t	[in]	specific_type	Specific type of the running application.

Returns

- True if combination is valid, false otherwise.

zpal_init_invalidate

```
void zpal_init_invalidate (void )
```

Setup the hardware so that the chip stops working.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- This function never exits.

zpal_get_reset_reason

```
zpal_reset_reason_t zpal_get_reset_reason (void )
```

Get reset reason.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Reset reason.

zpal_system_startup

```
void zpal_system_startup (zpal_reset_reason_t reset_reason)
```

System startup, implemented by Z-Wave SDK.

Parameters

Type	Direction	Argument Name	Description
zpal_reset_reason_t	[in]	reset_reason	Reset reason.

Note

- This function never exits.

Zpal-misc

Zpal-misc

Defines a platform abstraction layer for the Z-Wave miscellaneous functions, not covered by other modules.

Modules

[zpal_product_id_t](#)

Enumerations

```
enum zpal_chip_se_type_t {
    ZPAL_CHIP_SE_UNKNOWN
    ZPAL_CHIP_SE_MID
    ZPAL_CHIP_SE_HIGH
}
```

Defines for identifying the secure element type supported by the chip.

Typedefs

```
typedef uint16_t zpal_soft_reset_info_t
Manufacturer's reset information.

typedef uint16_t zpal_soft_reset_mfid_t
Manufacturer ID used by zpal_reboot_with_info.

typedef uint32_t zpal_time_t
type to store a time in zpal format.
```

Variables

```
const zpal_soft_reset_info_t ZPAL_RESET_REQUESTED_BY_SAPI

const zpal_soft_reset_info_t ZPAL_RESET_UNHANDLED_RADIO_EVENT

const zpal_soft_reset_info_t ZPAL_RESET_RADIO_ASSERT

const zpal_soft_reset_info_t ZPAL_RESET_ASSERT_PTR

const zpal_soft_reset_info_t ZPAL_RESET_EVENT_FLUSH_MEMORY

const zpal_soft_reset_info_t ZPAL_RESET_INFO_DEFAULT
```

Functions

void	zpal_reboot_with_info (const zpal_soft_reset_mfid_t manufacturer_id, const zpal_soft_reset_info_t reset_info) Perform a system reboot and provide information about the context.
void	zpal_initiate_shutdown_handler (void) Prepare for shutdown handler.
void	zpal_shutdown_handler (void) Shutdown handler.
size_t	zpal_get_serial_number_length (void) Get serial number length.
void	zpal_get_serial_number (uint8_t *serial_number) Get serial number.
bool	zpal_in_isr (void) Check if in ISR context.
uint8_t	zpal_get_chip_type (void) Get chip type.
uint8_t	zpal_get_chip_revision (void) Get chip revision.
uint32_t	zpal_get_app_version (void) Get application version.
uint8_t	zpal_get_app_version_major (void) Get major part of application version.
uint8_t	zpal_get_app_version_minor (void) Get minor part of application version.
uint8_t	zpal_get_app_version_patch (void) Get patch part of application version.
void	zpal_get_product_id (zpal_product_id_t *product_id) Get product id.
void	zpal_debug_init (void) Initialize debug output.
void	zpal_debug_output (const uint8_t *data, uint32_t length) Output debug logs.
void	zpal_disable_interrupts (void) Disable interrupts.
zpal_chip_se_type_t	zpal_get_secure_element_type (void) Get secure element type supported in the chip.
void	zpal_psa_set_location_persistent_key (const void *attributes) Set vendor specific location for storing keys persistently in wrapped or plain form based on the secure element type supported by the chip.
void	zpal_psa_set_location_volatile_key (const void *attributes) Set vendor specific location for storing keys in volatile memory, in wrapped or plain form based on the secure element type supported by the chip.

Enumeration Documentation

zpal_chip_se_type_t

```
zpal_chip_se_type_t
```

Defines for identifying the secure element type supported by the chip.

Enumerator	
ZPAL_CHIP_SE_UNKNOWN	
ZPAL_CHIP_SE_MID	
ZPAL_CHIP_SE_HIGH	

Typedef Documentation

zpal_soft_reset_info_t

```
typedef uint16_t zpal_soft_reset_info_t
```

Manufacturer's reset information.

zpal_soft_reset_mfid_t

```
typedef uint16_t zpal_soft_reset_mfid_t
```

Manufacturer ID used by zpal_reboot_with_info.

zpal_time_t

```
typedef uint32_t zpal_time_t
```

type to store a time in zpal format.

This format of this data depend on the zpal, so the stack should never try to manipulate this data. TODO: how to make this definition configurable by the zpal?

Variable Documentation

ZPAL_RESET_REQUESTED_BY_SAPI

```
const zpal_soft_reset_info_t ZPAL_RESET_REQUESTED_BY_SAPI
```

ZPAL_RESET_UNHANDLED_RADIO_EVENT

```
const zpal_soft_reset_info_t ZPAL_RESET_UNHANDLED_RADIO_EVENT
```

ZPAL_RESET_RADIO_ASSERT

```
const zpal_soft_reset_info_t ZPAL_RESET_RADIO_ASSERT
```

ZPAL_RESET_ASSERT_PTR

```
const zpal_soft_reset_info_t ZPAL_RESET_ASSERT_PTR
```

ZPAL_RESET_EVENT_FLUSH_MEMORY

```
const zpal_soft_reset_info_t ZPAL_RESET_EVENT_FLUSH_MEMORY
```

ZPAL_RESET_INFO_DEFAULT

```
const zpal_soft_reset_info_t ZPAL_RESET_INFO_DEFAULT
```

Function Documentation

zpal_reboot_with_info

```
void zpal_reboot_with_info (const zpal\_soft\_reset\_mfid\_t manufacturer_id, const zpal\_soft\_reset\_info\_t reset_info)
```

Perform a system reboot and provide information about the context.

Parameters

Type	Direction	Argument Name	Description
const zpal_soft_reset_mfid_t	[in]	manufacturer_id	manufacturer ID identifier.
const zpal_soft_reset_info_t	[in]	reset_info	the information to pass to boot.

zpal_initiate_shutdown_handler

```
void zpal_initiate_shutdown_handler (void )
```

Prepare for shutdown handler.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_shutdown_handler

```
void zpal_shutdown_handler (void )
```

Shutdown handler.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_get_serial_number_length

```
size_t zpal_get_serial_number_length (void )
```

Get serial number length.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Serial number length.

zpal_get_serial_number

```
void zpal_get_serial_number (uint8_t * serial_number)
```

Get serial number.

Parameters

Type	Direction	Argument Name	Description
uint8_t *	[out]	serial_number	Serial number.

zpal_in_isr

```
bool zpal_in_isr (void )
```

Check if in ISR context.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if the CPU is in handler mode (currently executing an interrupt handler). False if the CPU is in thread mode.

zpal_get_chip_type

```
uint8_t zpal_get_chip_type (void )
```

Get chip type.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Chip type.

zpal_get_chip_revision

```
uint8_t zpal_get_chip_revision (void )
```

Get chip revision.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Chip revision.

zpal_get_app_version

```
uint32_t zpal_get_app_version (void )
```

Get application version.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Application version.

Note

- This function exists in PAL to allow use app version by external module (e.g. bootloader).

zpal_get_app_version_major

```
uint8_t zpal_get_app_version_major (void )
```

Get major part of application version.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Major part of application version.

Note

- This function exists in PAL to allow use app version by external module (e.g. bootloader).

zpal_get_app_version_minor

```
uint8_t zpal_get_app_version_minor (void )
```

Get minor part of application version.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Minor part of application version.

Note

- This function exists in PAL to allow use app version by external module (e.g. bootloader).

zpal_get_app_version_patch

```
uint8_t zpal_get_app_version_patch (void )
```

Get patch part of application version.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Patch part of application version.

Note

- This function exists in PAL to allow use app version by external module (e.g. bootloader).

zpal_get_product_id

```
void zpal_get_product_id (zpal_product_id_t * product_id)
```

Get product id.

Parameters

Type	Direction	Argument Name	Description
zpal_product_id_t *	[out]	product_id	Product id.

Note

- This function exists in PAL to allow use product id by external module (e.g. bootloader).

zpal_debug_init

```
void zpal_debug_init (void )
```

Initialize debug output.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_debug_output

```
void zpal_debug_output (const uint8_t * data, uint32_t length)
```

Output debug logs.

Parameters

Type	Direction	Argument Name	Description
const uint8_t *	[out]	data	Pointer to debug data.
uint32_t	[in]	length	Length of debug data.

zpal_disable_interrupts

```
void zpal_disable_interrupts (void )
```

Disable interrupts.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_get_secure_element_type

```
zpal_chip_se_type_t zpal_get_secure_element_type (void )
```

Get secure element type supported in the chip.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Secure element type supported.

zpal_psa_set_location_persistent_key

```
void zpal_psa_set_location_persistent_key (const void * attributes)
```

Set vendor specific location for storing keys persistently in wrapped or plain form based on the secure element type supported by the chip.

Parameters

Type	Direction	Argument Name	Description
const void *	[in]	attributes	of the key

zpal_psa_set_location_volatile_key

```
void zpal_psa_set_location_volatile_key (const void * attributes)
```

Set vendor specific location for storing keys in volatile memory, in wrapped or plain form based on the secure element type supported by the chip.

Parameters

Type	Direction	Argument Name	Description
const void *	[in]	attributes	of the key

zpal_product_id_t

Product id struct.

Public Attributes

uint16_t	app_manufacturer_id	Manufacturer ID identifiers. (MFG_ID_XXX_XXX)
uint16_t	app_product_id	Product ID. (PRODUCT_ID_XXX_XXX)
uint16_t	app_product_type	Product type Id. (PRODUCT_TYPE_ID_ZWAVE_XXX_XXX)
uint16_t	app_icon_type	Z-Wave+ Icon Type identifiers. (ICON_TYPE_XXX_XXX)
uint8_t	generic_type	Generic Device Class identifier. (GENERIC_TYPE_XXX_XXX)
uint8_t	specyfifc_type	Specific Device Class identifier. (SPECYFIC_TYPE_XXX_XXX)
uint8_t	requested_security_key	Bitmask for security keys. (SECURITY_KEY_SX_XXX)

Public Attribute Documentation

app_manufacturer_id

uint16_t zpal_product_id_t::app_manufacturer_id

Manufacturer ID identifiers. (MFG_ID_XXX_XXX)

app_product_id

uint16_t zpal_product_id_t::app_product_id

Product ID. (PRODUCT_ID_XXX_XXX)

app_product_type

uint16_t zpal_product_id_t::app_product_type

Product type Id. (PRODUCT_TYPE_ID_ZWAVE_XXX_XXX)

app_icon_type

uint16_t zpal_product_id_t::app_icon_type

Z-Wave+ Icon Type identifiers. (ICON_TYPE_XXX_XXX)

generic_type

uint8_t zpal_product_id_t::generic_type

Generic Device Class identifier. (GENERIC_TYPE_XXX_XXX)

specyfic_type

uint8_t zpal_product_id_t::specyfic_type

Specific Device Class identifier. (SPECYFIC_TYPE_XXX_XXX)

requested_security_key

uint8_t zpal_product_id_t::requested_security_key

Bitmask for security keys. (SECURITY_KEY_SX_XXX)

Zpal-nvm

Zpal-nvm

Defines a platform abstraction layer for the Z-Wave NVM.

How to use the entropy API

The ZPAL NVM provides a way for an application to safely store and retrieve variable size data in a page-based non-volatile memory (NVM). Data are identified with 20-bit identifiers denoted as `data_id`.

The following outlines an example of use:

1. Initialize module with `zpal_nvm_init()`.
2. Invoke `zpal_nvm_write()` to store data.
3. Invoke `zpal_nvm_read()` to get data.

Enumerations

```
enum    zpal_nvm_area_t {
        ZPAL_NVM_AREA_APPLICATION
        ZPAL_NVM_AREA_ZAF
        ZPAL_NVM_AREA_STACK
        ZPAL_NVM_AREA_MANUFACTURER_TOKENS
    }
    Defines the different areas of NVM storage.
```

Typedefs

```
typedef uint32_t    zpal_nvm_object_key_t
    Data id type.

typedef void *      zpal_nvm_handle_t
    Defines the NVM handle type.
```

Functions

<code>zpal_nvm_handle_t</code>	<code>zpal_nvm_init(zpal_nvm_area_t area)</code> Initializes the NVM for a given area.
<code>zpal_status_t</code>	<code>zpal_nvm_close(zpal_nvm_handle_t handle)</code> Closes the NVM of a given handle.
<code>zpal_status_t</code>	<code>zpal_nvm_read(zpal_nvm_handle_t handle, zpal_nvm_object_key_t key, void *object, size_t object_size)</code> Reads object from a given area handle and given object key.
<code>zpal_status_t</code>	<code>zpal_nvm_read_object_part(zpal_nvm_handle_t handle, zpal_nvm_object_key_t key, void *object, size_t offset, size_t object_size)</code> Reads part of object from a given area handle and given object key.
<code>zpal_status_t</code>	<code>zpal_nvm_write(zpal_nvm_handle_t handle, zpal_nvm_object_key_t key, const void *object, size_t object_size)</code> Writes object to a given area handle and given object key.

zpal_status_t	zpal_nvm_erase_all(zpal_nvm_handle_t handle) Erases everything in a given area.
zpal_status_t	zpal_nvm_erase_object(zpal_nvm_handle_t handle, zpal_nvm_object_key_t key) Delete an object from NVM.
zpal_status_t	zpal_nvm_get_object_size(zpal_nvm_handle_t handle, zpal_nvm_object_key_t key, size_t *len) Get the object size identified with a given key from NVM.
size_t	zpal_nvm_enum_objects(zpal_nvm_handle_t handle, zpal_nvm_object_key_t *key_list, size_t key_list_size, zpal_nvm_object_key_t key_min, zpal_nvm_object_key_t key_max) Create a list of key for valid objects in NVM.
zpal_status_t	zpal_nvm_backup_open(void) Opens the NVM for a backup/restore operation and temporarily disables access to other processes.
zpal_status_t	zpal_nvm_backup_close(void) Closes the NVM after backup/restore operation.
zpal_status_t	zpal_nvm_backup_read(uint32_t offset, void *data, size_t data_length) Reads raw data from the NVM.
zpal_status_t	zpal_nvm_backup_write(uint32_t offset, const void *data, size_t data_length) Writes raw data to the NVM.
size_t	zpal_nvm_backup_get_size(void) Get size of raw NVM area.
zpal_status_t	zpal_nvm_lock(zpal_nvm_handle_t handle) Set write protection for storage.
zpal_status_t	zpal_nvm_migrate_legacy_app_file_system(void) Migrates 800s legacy application file system to shared app/protocol file system.

Enumeration Documentation

zpal_nvm_area_t

```
zpal_nvm_area_t
```

Defines the different areas of NVM storage.

Enumerator	
ZPAL_NVM_AREA_APPLICATION	Reserved for application data.
ZPAL_NVM_AREA_ZAF	Reserved for ZAF data.
ZPAL_NVM_AREA_STACK	Reserved for stack data.
ZPAL_NVM_AREA_MANUFACTURER_TOKENS	Reserved for manufacturer tokens.

Typedef Documentation

zpal_nvm_object_key_t

```
typedef uint32_t zpal_nvm_object_key_t
```

Data id type.

Only the 20 least significant bits are used.

zpal_nvm_handle_t

```
typedef void* zpal_nvm_handle_t
```

Defines the NVM handle type.

Function Documentation

zpal_nvm_init

```
zpal_nvm_handle_t zpal_nvm_init (zpal_nvm_area_t area)
```

Initializes the NVM for a given area.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_area_t	[in]	area	NVM area to initialize.

Returns

- A valid NVM handle if successfully initialized. Otherwise it returns NULL.

zpal_nvm_close

```
zpal_status_t zpal_nvm_close (zpal_nvm_handle_t handle)
```

Closes the NVM of a given handle.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	NVM area handle.

Returns

- ZPAL_STATUS_OK if the file system was successfully closed and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_read

```
zpal_status_t zpal_nvm_read (zpal_nvm_handle_t handle, zpal_nvm_object_key_t key, void * object, size_t object_size)
```

Reads object from a given area handle and given object key.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	NVM area handle.
zpal_nvm_object_key_t	[in]	key	Object key.
void *	[out]	object	Address of array where object can be written to.

Type	Direction	Argument Name	Description
size_t	[in]	object_size	Size of the stored object.

Returns

- [ZPAL_STATUS_OK](#) if the object was successfully read and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_read_object_part

```
zpal_status_t zpal_nvm_read_object_part (zpal_nvm_handle_t handle, zpal_nvm_object_key_t key, void * object, size_t offset, size_t object_size)
```

Reads part of object from a given area handle and given object key.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	NVM area handle.
zpal_nvm_object_key_t	[in]	key	Object key.
void *	[out]	object	Address of array where object can be written to.
size_t	[in]	offset	The offset where object shall be read from.
size_t	[in]	object_size	Size of the stored object.

Returns

- [ZPAL_STATUS_OK](#) if the object was successfully read and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_write

```
zpal_status_t zpal_nvm_write (zpal_nvm_handle_t handle, zpal_nvm_object_key_t key, const void * object, size_t object_size)
```

Writes object to a given area handle and given object key.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	NVM area handle.
zpal_nvm_object_key_t	[in]	key	Object key.
const void *	[out]	object	Address of array of object that must be written.
size_t	[in]	object_size	Size of the object to be stored.

Returns

- [ZPAL_STATUS_OK](#) if the object was successfully written and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_erase_all

```
zpal_status_t zpal_nvm_erase_all (zpal_nvm_handle_t handle)
```

Erases everything in a given area.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	NVM area handle.

Returns

- [ZPAL_STATUS_OK](#) if the all objects was successfully erased and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_erase_object

```
zpal_status_t zpal_nvm_erase_object (zpal\_nvm\_handle\_t handle, zpal\_nvm\_object\_key\_t key)
```

Delete an object from NVM.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	Nvm storage handle.
zpal_nvm_object_key_t	[in]	key	Object key.

Returns

- [ZPAL_STATUS_OK](#) on success and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_get_object_size

```
zpal_status_t zpal_nvm_get_object_size (zpal\_nvm\_handle\_t handle, zpal\_nvm\_object\_key\_t key, size_t * len)
```

Get the object size identified with a given key from NVM.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	Nvm storage handle.
zpal_nvm_object_key_t	[in]	key	Object key.
size_t *	[out]	len	Object size.

Returns

- [ZPAL_STATUS_OK](#) on success and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_enum_objects

```
size_t zpal_nvm_enum_objects (zpal\_nvm\_handle\_t handle, zpal\_nvm\_object\_key\_t * key_list, size_t key_list_size, zpal\_nvm\_object\_key\_t key_min, zpal\_nvm\_object\_key\_t key_max)
```

Create a list of key for valid objects in NVM.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	Nvm storage handle.
zpal_nvm_object_key_t *	[out]	key_list	A pointer to a buffer for the key list.
size_t	[in]	key_list_size	The size of the key list buffer.
zpal_nvm_object_key_t	[in]	key_min	The lower search key.
zpal_nvm_object_key_t	[in]	key_max	The upper search key.

Returns

- The number of keys written to the key list. This value is less than or equal to `key_list_size` .

zpal_nvm_backup_open

```
zpal_status_t zpal_nvm_backup_open (void )
```

Opens the NVM for a backup/restore operation and temporarily disables access to other processes.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- [ZPAL_STATUS_OK](#) on success and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_backup_close

```
zpal_status_t zpal_nvm_backup_close (void )
```

Closes the NVM after backup/restore operation.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_nvm_backup_read

```
zpal_status_t zpal_nvm_backup_read (uint32_t offset, void * data, size_t data_length)
```

Reads raw data from the NVM.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	offset	The offset where data shall be read from.
void *	[out]	data	Address of array where data can be written to.
size_t	[in]	data_length	Length of the stored data.

Returns

- [ZPAL_STATUS_OK](#) if the data was successfully read and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_backup_write

```
zpal_status_t zpal_nvm_backup_write (uint32_t offset, const void * data, size_t data_length)
```

Writes raw data to the NVM.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	offset	The offset where data shall be written to.
const void *	[out]	data	Address of array of data that must be written.
size_t	[in]	data_length	Length of the data to be stored.

Returns

- [ZPAL_STATUS_OK](#) if the data was successfully written and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_backup_get_size

```
size_t zpal_nvm_backup_get_size (void )
```

Get size of raw NVM area.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Size of raw NVM area.

zpal_nvm_lock

```
zpal_status_t zpal_nvm_lock (zpal\_nvm\_handle\_t handle)
```

Set write protection for storage.

Parameters

Type	Direction	Argument Name	Description
zpal_nvm_handle_t	[in]	handle	Nvm storage handle.

Returns

- [ZPAL_STATUS_OK](#) if the lock was set successfully, [ZPAL_STATUS_FAIL](#) otherwise.

zpal_nvm_migrate_legacy_app_file_system

```
zpal_status_t zpal_nvm_migrate_legacy_app_file_system (void )
```

Migrates 800s legacy application file system to shared app/protocol file system.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- ZPAL_STATUS_OK if the file system was successfully migrated and [ZPAL_STATUS_FAIL](#) otherwise.

Zpal-power-manager

Zpal-power-manager

Defines a platform abstraction layer for the Z-Wave power manager.

The ZPAL power manager offers an API for registering power locks and for receiving events on power mode transitions. Power lock forces the chip to stay awake in a given power mode for a given time. An event handler is invoked every time the transition between power modes occurs.

Infinite lock example:

```
zpal_pm_lock(ZPAL_PM_TYPE_USE_RADIO, ZPAL_PM_DOMAIN_APP, 0, id);          Keep the radio active forever, id is the lock ID find it in
zpal_power_manager_ids.h
                                Other stuff
zpal_pm_lock_cancel(type, id);      The chip may go to sleep now
```

Lock with timeout example:

```
zpal_pm_lock(ZPAL_PM_TYPE_USE_RADIO, ZPAL_PM_DOMAIN_APP, 2000, id);      Keep the radio active for 2000 ms from now
```

Requirements:

- The storage for power locks should be able to hold at least 20 items. Note that this number may change in the future.

Modules

[pm_domain_status_t](#)

Enumerations

```
enum    zpal_pm_type_t {
        ZPAL_PM_TYPE_USE_RADIO
        ZPAL_PM_TYPE_USE_RADIO_TX = ZPAL_PM_TYPE_USE_RADIO
        ZPAL_PM_TYPE_USE_RADIO_RX = ZPAL_PM_TYPE_USE_RADIO
        ZPAL_PM_TYPE_DEEP_SLEEP
        ZPAL_PM_TYPE_MAX
    }
    Power manager type in which a lock can request an operation to prevent the system from entering a sleep mode.
```

```
enum    zpal_pm_powerlevel_t {
        ZPAL_PM_ACTIVE
        ZPAL_PM_LIGHT_SLEEP
        ZPAL_PM_DEEP_SLEEP
        ZPAL_PM_HALTED
    }
    Power manager power levels, each level corresponds to physical power mode implementation.
```

```
enum zpal_pm_state_t {
    ZPAL_PM_STATE_ACTIVE_PERMANENT
    ZPAL_PM_STATE_ACTIVE_TIMER
    ZPAL_PM_STATE_INACTIVE
    ZPAL_PM_STATE_UNDEFINED
}
```

For each lock, three states are defined:

```
enum zpal_pm_domain_t {
    ZPAL_PM_DOMAIN_APP
    ZPAL_PM_DOMAIN_PAL
    ZPAL_PM_DOMAIN_STACK
    ZPAL_PM_DOMAIN_UNDEFINED
}
```

A domain is a group of locks.

Functions

zpal_status_t	zpal_pm_lock (zpal_pm_type_t type, zpal_pm_domain_t domain, uint32_t timeout_ms, uint8_t idx)	Locks the power mode for a given time period.
zpal_status_t	zpal_pm_relock (zpal_pm_type_t type, zpal_pm_domain_t domain, uint32_t timeout_ms, uint8_t idx)	Locks the power mode for a given time period.
uint32_t	zpal_get_max_timeout (void)	Gets the maximum timeout value that can be set for a lock.
bool	zpal_pm_lock_is_active (zpal_pm_type_t type, zpal_pm_domain_t domain, uint8_t idx, zpal_pm_state_t *state)	Return the state of a given lock.
bool	zpal_pm_lock_type_is_active (zpal_pm_type_t pm_type, zpal_pm_state_t *state)	Return the state of the given type of lock.
zpal_status_t	zpal_pm_register_domain (zpal_pm_domain_t domain, pm_domain_status_t *radio_domain_status, pm_domain_status_t *deep_sleep_domain_status)	Registers a domain for power locks.
zpal_status_t	zpal_pm_lock_cancel (zpal_pm_type_t type, zpal_pm_domain_t domain, uint8_t idx)	This function cancels an active power lock despite of the ongoing mode (permanent or timer).
void	zpal_pm_lock_cancel_all (void)	Cancels all active power locks on both radio and deep sleep power locks.
void	zpal_zw_pm_event_handler (zpal_pm_powerlevel_t from, zpal_pm_powerlevel_t to)	Function to update stack logic (radio operations and configuration) before power mode transitions defined in zpal_pm_powerlevel_t .
void	zpal_pm_halt (void)	

Enumeration Documentation

zpal_pm_type_t

zpal_pm_type_t

Power manager type in which a lock can request an operation to prevent the system from entering a sleep mode.

Enumerator

ZPAL_PM_TYPE_USE_RADIO	
ZPAL_PM_TYPE_USE_RADIO_TX	
ZPAL_PM_TYPE_USE_RADIO_RX	
ZPAL_PM_TYPE_DEEP_SLEEP	
ZPAL_PM_TYPE_MAX	

zpal_pm_powerlevel_t

zpal_pm_powerlevel_t

Power manager power levels, each level corresponds to physical power mode implementation.

Enumerator	
ZPAL_PM_ACTIVE	Processor is active.
ZPAL_PM_LIGHT_SLEEP	Processor is sleeping, but all peripherals are enabled.
ZPAL_PM_DEEP_SLEEP	Most peripherals are disabled, but the radio is in FLiRS mode.
ZPAL_PM_HALTED	Chip is off and will boot from reset. Hence, no transition from this power mode will ever happen.

zpal_pm_state_t

zpal_pm_state_t

For each lock, three states are defined:

- ZPAL_PM_STATE_ACTIVE_PERMANENT: The lock is active and will remain active until it is cancelled.
- ZPAL_PM_STATE_ACTIVE_TIMER: The lock is active and will remain active until the timer expires.
- ZPAL_PM_STATE_INACTIVE: The lock is inactive.

Enumerator	
ZPAL_PM_STATE_ACTIVE_PERMANENT	
ZPAL_PM_STATE_ACTIVE_TIMER	
ZPAL_PM_STATE_INACTIVE	
ZPAL_PM_STATE_UNDEFINED	

zpal_pm_domain_t

zpal_pm_domain_t

A domain is a group of locks.

They are stored in the same domain structure.

Enumerator	
ZPAL_PM_DOMAIN_APP	
ZPAL_PM_DOMAIN_PAL	
ZPAL_PM_DOMAIN_STACK	
ZPAL_PM_DOMAIN_UNDEFINED	

Function Documentation

zpal_pm_lock

```
zpal_status_t zpal_pm_lock (zpal_pm_type_t type, zpal_pm_domain_t domain, uint32_t timeout_ms, uint8_t idx)
```

Locks the power mode for a given time period.

Parameters

Type	Direction	Argument Name	Description
zpal_pm_type_t	N/A	type	The power manager type for the lock. If the lock is already active, this function does not overwrite the previous lock state. zpal_pm_relock can be used to overwrite the previous lock.
zpal_pm_domain_t	N/A	domain	The domain in which the lock is registered.
uint32_t	N/A	timeout_ms	The timeout duration in ms. If set to 0, the lock will be permanent. The maximum timeout value is defined by zpal_get_max_timeout .
uint8_t	N/A	idx	The lock index

Returns

- true when the lock is successfully acquired
- ZPAL_STATUS_BUSY if the lock is already in active state.
- ZPAL_STATUS_INVALID_ARGUMENT when the type or idx are invalid or timeout is greater than the maximum timeout.
- ZPAL_STATUS_OK if the lock is successfully acquired.

Note

- to acquire by force, use [zpal_pm_relock](#)

zpal_pm_relock

```
zpal_status_t zpal_pm_relock (zpal_pm_type_t type, zpal_pm_domain_t domain, uint32_t timeout_ms, uint8_t idx)
```

Locks the power mode for a given time period.

Parameters

Type	Direction	Argument Name	Description
zpal_pm_type_t	N/A	type	The power manager type for the lock.
zpal_pm_domain_t	N/A	domain	The domain in which the lock is registered.
uint32_t	N/A	timeout_ms	The timeout duration in ms. If set to 0, the lock will be permanent. The maximum timeout is return by zpal_get_max_timeout .
uint8_t	N/A	idx	The lock index

If the lock is already active this function overwrites the previous lock. Use with caution. You can get the lock active state for the given id by calling [zpal_pm_lock_is_active](#)

- Returns
- true when the lock is successfully acquired, false otherwise (bad parameters).
 - ZPAL_STATUS_INVALID_ARGUMENT when the type or idx are invalid or timeout is greater than the maximum timeout.
 - ZPAL_STATUS_OK if the lock is successfully acquired.

zpal_get_max_timeout

```
uint32_t zpal_get_max_timeout (void )
```

Gets the maximum timeout value that can be set for a lock.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The maximum timeout value in ms.

zpal_pm_lock_is_active

```
bool zpal_pm_lock_is_active (zpal_pm_type_t type, zpal_pm_domain_t domain, uint8_t idx, zpal_pm_state_t * state)
```

Return the state of a given lock.

Parameters

Type	Direction	Argument Name	Description
zpal_pm_type_t	[in]	type	The power manager type for the lock.
zpal_pm_domain_t	[in]	domain	The domain of the lock. A domain is a group of locks that can be managed together.
uint8_t	[in]	idx	The lock index
zpal_pm_state_t *	[out]	state	(optional) More detailed state of the lock: INACTIVE, ACTIVE_PERMANENT, ACTIVE_TIMER.

Returns

- true is in active state, false if inactive.

Note

- Similar to [zpal_pm_lock_type_is_active](#) but for an individual lock.

zpal_pm_lock_type_is_active

```
bool zpal_pm_lock_type_is_active (zpal_pm_type_t pm_type, zpal_pm_state_t * state)
```

Return the state of the given type of lock.

Parameters

Type	Direction	Argument Name	Description
zpal_pm_type_t	[in]	pm_type	The power manager type to be checked.
zpal_pm_state_t *	[out]	state	(optional) More detailed state: INACTIVE, ACTIVE_PERMANENT, ACTIVE_TIMER.

Resulted from all zwave locks operations on that type. **Returns**

- true if the lock is active, false if inactive.

Note

- : Similar to [zpal_pm_lock_is_active](#) but done at the type level which contains any locks operating on that type.

zpal_pm_register_domain

```
zpal_status_t zpal_pm_register_domain (zpal_pm_domain_t domain, pm_domain_status_t * radio_domain_status, pm_domain_status_t * deep_sleep_domain_status)
```

Registers a domain for power locks.

Parameters

Type	Direction	Argument Name	Description
zpal_pm_domain_t	N/A	domain	The domain to be registered.
pm_domain_status_t *	[in]	radio_domain_status	Pointer to the radio domain status. If NULL, the domain will not be registered.
pm_domain_status_t *	[in]	deep_sleep_domain_status	Pointer to the deep sleep domain status. If NULL, the domain will not be registered.

Returns

- ZPAL_STATUS_OK if one or two of the domains status are successfully registered, ZPAL_STATUS_FAIL otherwise.

zpal_pm_lock_cancel

```
zpal_status_t zpal_pm_lock_cancel (zpal_pm_type_t type, zpal_pm_domain_t domain, uint8_t idx)
```

This function cancels an active power lock despite of the ongoing mode (permanent or timer).

Parameters

Type	Direction	Argument Name	Description
zpal_pm_type_t	[in]	type	The lock type to cancel.
zpal_pm_domain_t	N/A	domain	The domain in which the lock is associated.
uint8_t	[in]	idx	The lock index to cancel.

Note

- If the specified type is not supported, this function will not perform any action.

Returns

- ZPAL_STATUS_OK if the lock is successfully cancelled, ZPAL_STATUS_FAIL otherwise.

zpal_pm_lock_cancel_all

```
void zpal_pm_lock_cancel_all (void )
```

Cancels all active power locks on both radio and deep sleep power locks.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Note

- It cancels all power locks created during runtime, hence it can be used ONLY as a part of power down routine.

zpal_zw_pm_event_handler

```
void zpal_zw_pm_event_handler (zpal_pm_powerlevel_t from, zpal_pm_powerlevel_t to)
```

Function to update stack logic (radio operations and configuration) before power mode transitions defined in [zpal_pm_powerlevel_t](#).

Parameters

Type	Direction	Argument Name	Description
zpal_pm_powerlevel_t	N/A	from	The power level from which the transition is happening.
zpal_pm_powerlevel_t	N/A	to	The power level to which the transition is happening.

Note

- This function is called through app_pm_transition_event and is mapped onto power manager platform specific events.

zpal_pm_halt

```
void zpal_pm_halt (void )
```

Parameters

Type	Direction	Argument Name	Description
void	N/A		

pm_domain_status_t

Power manager domain status structure.

A status is a memory block that holds the state of the locks in a domain. A domain consists of two statuses struct: one for the radio and one for the deep sleep.

Public Attributes

uint8_t	length
unsigned int	permanent_bitfield
uint32_t *	pm_timeouts

Public Attribute Documentation

length

uint8_t pm_domain_status_t::length

permanent_bitfield

unsigned int pm_domain_status_t::permanent_bitfield

pm_timeouts

uint32_t* pm_domain_status_t::pm_timeouts

Zpal-radio

Zpal-radio

Defines a platform abstraction layer for the Z-Wave radio.

The ZPAL Radio module contains several APIs which are required to be implemented.

A user of this radio module shall first define all parameters of the Z-Wave Radio Profile (i.e., [zpal_radio_profile_t](#)) and initialise the radio using `zpal_radio_init` API.

After the initialisation of the radio is executed, the user can use any of the radio APIs to execute radio related paradigm for instance

- `zpal_radio_transmit` shall be used to transmit a Z-Wave frame on radio
- `zpal_radio_start_receive` shall be used to enable the reception of Z-Wave frame

The radio API assumes that the radio will return to receive mode with channel hopping enabled after transmitting a frame.

Initialization of the radio

```

void
RXHandlerFromISR(zpal_radio_event_t rxStatus)
{
    // Rx handle in ISR context
}

void
TXHandlerFromISR(zpal_radio_event_t txEvent)
{
    // Tx complete handle in ISR context
}

void
RegionChangeHandler(zpal_radio_event_t regionChangeStatus)
{
    // Region changed, make sure region specific data and statistics are cleared
}

void
RadioAssertHandler(zpal_radio_event_t assertVal)
{
    // Radio driver or hardware asserted, handle it
}

initialize_radio()
{
    static zpal_radio_profile_t RfProfile;
    static zpal_radio_network_stats_t sNetworkStatistic = {0};
    static uint8_t network_homeid[4] = {0xDE, 0xAD, 0xBE, 0xEF};

    // Set radio for US always on mode
    static zpal_radio_profile_t RfProfile = {.region = REGION_US,
        .wakeup = ZPAL_RADIO_WAKEUP_ALWAYS_LISTEN,
        .listen_before_talk_threshold = ELISTENBEFORETALKTRESHOLD_DEFAULT,
        .tx_power_max = 0,
        .tx_power_adjust = 33,
        .tx_power_max_lr = 140,
        .home_id = &network_homeid,
        .rx_cb = RXHandlerFromISR,
        .tx_cb = TXHandlerFromISR,
        .region_change_cb = RegionChangeHandler,
        .assert_cb = RadioAssertHandler,
        .network_stats = &sNetworkStatistic,
        .radio_debug_enable = false,
        .primary_lr_channel = ZPAL_RADIO_LR_CHANNEL_A};

    zpal_radio_init(RfProfile);
}

```

Transmitting a frame


```

#define TX_FAILED      0
#define TX_FRAME_SUCCESS  1
#define TX_BEAM_SUCCESS  2

static const zpal_radio_transmit_parameter_t TxParameter100kCh1 = {.speed = ZPAL_RADIO_SPEED_100K,
                                                                    .channel_id = 0,
                                                                    .crc = ZPAL_RADIO_CRC_16_BIT_CCITT,
                                                                    .preamble = 0x55,
                                                                    .preamble_length = 40,
                                                                    .start_of_frame = 0xF0,
                                                                    .repeats = 0};

zpal_status_t transmit_frame()
{
    // Singlecast MAC header from node 1 to node 2
    uint8_t header_buffer[9] = {0xDE, 0xAD, 0xBE, 0xEF, 0x01, 0x41, 0x01, 14, 0x02};
    // Basic set On frame
    uint8_t payload_buffer[3] = {0x20, 0x01, 0xFF};

    return zpal_radio_transmit(&TxParameter100kCh1, // use 100kbps FSK profile
                               9,
                               (uint8_t *)&header_buffer,
                               3,
                               (uint8_t *)&payload_buffer,
                               true, // Do LBT before transmit
                               0); // Transmit power 0dBm
}

void
TXHandlerFromISR(zpal_radio_event_t txEvent)
{
    uint8_t status;

    if ((ZPAL_RADIO_EVENT_TX_FAIL == txEvent) || (ZPAL_RADIO_EVENT_TX_FAIL_LBT == txEvent))
    {
        status = TX_FAILED;
    }
    else if (txEvent & ZPAL_RADIO_EVENT_FLAG_BEAM)
    {
        status = TX_BEAM_SUCCESS;
    }
    else
    {
        status = TX_FRAME_SUCCESS;
    }
    // Get out of ISR context and handle transmit complete
    HandleTransmitComplete(status);
}

```

Receiving a frame

```
#define RX_ABORT          0
#define RX_BEAM          1
#define RX_FRAME         2
#define RX_CALIBRATION_NEEDED  3

void
RXHandlerFromISR(zpal_radio_event_t rxStatus)
{
    uint8_t status;

    switch (rxStatus)
    {
        case ZPAL_RADIO_EVENT_RX_BEAM:
            // Beam received
            status = RX_BEAM;
            break;
        case ZPAL_RADIO_EVENT_RX_ABORT:
            // Receive was aborted due to an error in the frame
            status = RX_ABORT;
            break;
        case ZPAL_RADIO_EVENT_RX:
            // Valid frame received
            status = RX_FRAME;
            break;
        case ZPAL_RADIO_EVENT_RXTX_CALIBRATE:
            // Radio reports that calibration is needed
            status = RX_CALIBRATION_NEEDED;
            break;
    }

    // Get out of ISR context and handle frame
    HandleFrame(status);
}
```

Modules

[zpal_radio_transmit_parameter_t](#)

[zpal_radio_rx_parameters_t](#)

[zpal_radio_receive_frame_t](#)

[zpal_radio_network_stats_t](#)

[zpal_radio_rf_channel_statistic_t](#)

[zpal_radio_profile_t](#)

[Number of Channels](#)

Enumerations

```
enum    zpal_radio_wakeup_t {
    ZPAL_RADIO_WAKEUP_NEVER_LISTEN
    ZPAL_RADIO_WAKEUP_ALWAYS_LISTEN
    ZPAL_RADIO_WAKEUP_EVERY_250ms
    ZPAL_RADIO_WAKEUP_EVERY_1000ms
}
Wakeup interval for the radio.
```

```
enum    zpal_radio_crc_t {  
        ZPAL_RADIO_CRC_NONE  
        ZPAL_RADIO_CRC_8_BIT_XOR  
        ZPAL_RADIO_CRC_16_BIT_CCITT  
    }  
    Enumeration containing supported checksum types in Z-Wave.  
  
enum    zpal_radio_speed_t {  
        ZPAL_RADIO_SPEED_UNDEFINED  
        ZPAL_RADIO_SPEED_9600  
        ZPAL_RADIO_SPEED_40K  
        ZPAL_RADIO_SPEED_100K  
        ZPAL_RADIO_SPEED_100KLR  
    }  
    Enumeration containing supported baud rates.  
  
enum    zpal_radio_lr_channel_t {  
        ZPAL_RADIO_LR_CHANNEL_UNINITIALIZED  
        ZPAL_RADIO_LR_CHANNEL_A  
        ZPAL_RADIO_LR_CHANNEL_B  
        ZPAL_RADIO_LR_CHANNEL_UNKNOWN  
        ZPAL_RADIO_LR_CHANNEL_AUTO = 255  
    }  
    Enumeration containing Long Range Channels.  
  
enum    zpal_radio_zwave_channel_t {  
        ZPAL_RADIO_ZWAVE_CHANNEL_0 = 0  
        ZPAL_RADIO_ZWAVE_CHANNEL_1 = 1  
        ZPAL_RADIO_ZWAVE_CHANNEL_2 = 2  
        ZPAL_RADIO_ZWAVE_CHANNEL_3 = 3  
        ZPAL_RADIO_ZWAVE_CHANNEL_4 = 4  
        ZPAL_RADIO_ZWAVE_CHANNEL_NUM  
        ZPAL_RADIO_ZWAVE_CHANNEL_UNKNOWN = 0xFF  
    }  
    Enumeration containing Z-Wave channels.  
  
enum    zpal_radio_protocol_mode_t {  
        ZPAL_RADIO_PROTOCOL_MODE_1  
        ZPAL_RADIO_PROTOCOL_MODE_2  
        ZPAL_RADIO_PROTOCOL_MODE_3  
        ZPAL_RADIO_PROTOCOL_MODE_4  
        ZPAL_RADIO_PROTOCOL_MODE_UNDEFINED = 0xFF  
    }  
    Enumeration containing Z-Wave channels known as protocol modes.  
  
enum    zpal_radio_lr_channel_config_t {  
        ZPAL_RADIO_LR_CH_CFG_NO_LR  
        ZPAL_RADIO_LR_CH_CFG1  
        ZPAL_RADIO_LR_CH_CFG2  
        ZPAL_RADIO_LR_CH_CFG3  
        ZPAL_RADIO_LR_CH_CFG_COUNT  
    }  
    List of LR channel configuration.
```

```
enum zpal\_radio\_region\_t {  
    REGION_2CH_FIRST = 0  
    REGION_EU = REGION_2CH_FIRST  
    REGION_US  
    REGION_ANZ  
    REGION_HK  
    REGION_DEPRECATED_4  
    REGION_IN = 5  
    REGION_IL  
    REGION_RU  
    REGION_CN  
    REGION_US_LR  
    REGION_DEPRECATED_10  
    REGION_EU_LR  
    REGION_2CH_END  
    REGION_3CH_FIRST = 32  
    REGION_JP = REGION_3CH_FIRST  
    REGION_KR  
    REGION_3CH_END  
    REGION_DEPRECATED_48 = 48  
    REGION_UNDEFINED = 0xFE  
    REGION_DEFAULT = 0xFF  
  
}  
Enumeration official Z-Wave regions.
```

```
enum zpal\_radio\_tx\_power\_t {  
    ZPAL_RADIO_TX_POWER_DEFAULT  
    ZPAL_RADIO_TX_POWER_MINUS1_DBM  
    ZPAL_RADIO_TX_POWER_MINUS2_DBM  
    ZPAL_RADIO_TX_POWER_MINUS3_DBM  
    ZPAL_RADIO_TX_POWER_MINUS4_DBM  
    ZPAL_RADIO_TX_POWER_MINUS5_DBM  
    ZPAL_RADIO_TX_POWER_MINUS6_DBM  
    ZPAL_RADIO_TX_POWER_MINUS7_DBM  
    ZPAL_RADIO_TX_POWER_MINUS8_DBM  
    ZPAL_RADIO_TX_POWER_MINUS9_DBM  
    ZPAL_RADIO_TX_POWER_UNINITIALIZED = 126  
    ZPAL_RADIO_TX_POWER_REDUCED = 127  
  
}  
Enumeration containing Tx power settings.
```

```
enum zpal\_radio\_header\_type\_t {  
    ZPAL_RADIO_HEADER_TYPE_2CH  
    ZPAL_RADIO_HEADER_TYPE_3CH  
    ZPAL_RADIO_HEADER_TYPE_LR  
    ZPAL_RADIO_HEADER_TYPE_UNDEFINED  
  
}  
Z-Wave Frame header format types.
```

```
enum zpal\_radio\_event\_t {
    ZPAL_RADIO_EVENT_NONE
    ZPAL_RADIO_EVENT_RX_COMPLETE
    ZPAL_RADIO_EVENT_TX_COMPLETE
    ZPAL_RADIO_EVENT_RX_BEAM_COMPLETE
    ZPAL_RADIO_EVENT_TX_BEAM_COMPLETE
    ZPAL_RADIO_EVENT_RX_ABORT
    ZPAL_RADIO_EVENT_TX_FAIL
    ZPAL_RADIO_EVENT_TX_FAIL_LBT
    ZPAL_RADIO_EVENT_RXTX_CALIBRATE
    ZPAL_RADIO_EVENT_MASK = 0x1F
    ZPAL_RADIO_EVENT_FLAG_SUCCESS = 0x80
    ZPAL_RADIO_EVENT_TX_TIMEOUT = 254
}
Enumeratio radio events.

enum zpal\_radio\_status\_t {
    ZPAL_RADIO_STATUS_IDLE
    ZPAL_RADIO_STATUS_RX
    ZPAL_RADIO_STATUS_TX
    ZPAL_RADIO_STATUS_TX_BEAM
    ZPAL_RADIO_STATUS_RX_BEAM
}

enum zpal\_radio\_application\_t {
    ZPAL_RADIO_APPLICATION_DEFAULT = 0
    ZPAL_RADIO_APPLICATION_ZNIFFER
}
```

Typedefs

```
typedef uint16_t node\_id\_t
Node ID type.

typedef int16_t zpal\_tx\_power\_t
Parameter type to store deci dBm values.

typedef void(*) zpal\_radio\_callback\_t(const zpal_radio_event_t event)

typedef void(*) zpal\_radio\_receive\_handler\_t(zpal_radio_rx_parameters_t *rx_parameters, zpal_radio_receive_frame_t *frame)
Function pointer declaration for handling of received frames.
```

Functions

```
void zpal\_radio\_set\_network\_ids(uint32_t home_id, node_id_t node_id, uint8_t home_id_hash)
Sets the home ID & the node ID.

void zpal\_radio\_init(zpal_radio_profile_t *const profile)
Initializes the radio.

zpal\_status\_t zpal\_radio\_change\_region(zpal_radio_region_t eRegion, zpal_radio_lr_channel_config_t eLrChCfg)
Function used to change region and Long Range mode at the same time.

zpal\_radio\_region\_t zpal\_radio\_get\_region(void)
Function for getting REGION runtime.
```

zpal_status_t	zpal_radio_transmit (zpal_radio_transmit_parameter_t const *const tx_parameters, uint8_t frame_header_length, uint8_t const *const frame_header_buffer, uint8_t frame_payload_length, uint8_t const *const frame_payload_buffer) Function for transmitting a Z-Wave frame though the radio.
zpal_status_t	zpal_radio_transmit_beam (zpal_radio_transmit_parameter_t const *const tx_parameters, uint8_t beam_data_len, uint8_t const *const beam_data) Function for transmitting a Z-Wave Beam frame though the radio.
void	zpal_radio_start_receive (void) Starts the receiver and enables reception of frames.
void	zpal_radio_get_last_received_frame (void) Function to get last received frame.
void	zpal_radio_power_down (void) Powers down the radio transceiver.
zpal_radio_protocol_mode_t	zpal_radio_get_protocol_mode (void) Function to get the protocol mode used in the configured region.
zpal_radio_zwave_channel_t	zpal_radio_get_last_beam_channel (void) Get last beam channel.
int8_t	zpal_radio_get_last_beam_rssi (void) Get last beam RSSI.
void	zpal_radio_set_lbt_level (uint8_t channel, int8_t level) Function for setting the LBT RSSI level.
void	zpal_radio_enable_rx_broadcast_beam (bool enable) Enable or disables reception of broadcast beam.
void	zpal_radio_clear_tx_timers (void) Function for clearing current Channel Transmit timers.
void	zpal_radio_clear_network_stats (void) Function for clearing current Network statistics.
void	zpal_radio_rf_channel_statistic_clear (zpal_radio_zwave_channel_t zwavechannel) Function for clearing specified rf channel statistics.
bool	zpal_radio_rf_channel_statistic_get (zpal_radio_zwave_channel_t zwavechannel, zpal_radio_rf_channel_statistic_t *p_radio_channel_statistic)
void	zpal_radio_rf_channel_statistic_tx_channel_set (zpal_radio_zwave_channel_t zwavechannel) Function for setting the zwavechannel used when calling zpal_radio_rf_channel_statistic_tx_frames , zpal_radio_rf_channel_statistic_tx_retries and zpal_radio_rf_channel_statistic_tx_lbt_failures .
void	zpal_radio_rf_channel_statistic_tx_frames (void) Function for incrementing the rf channel tx frame statistic.
void	zpal_radio_rf_channel_statistic_tx_retries (void) Function for incrementing the rf channel tx retries statistic.
void	zpal_radio_rf_channel_statistic_tx_lbt_failures (void) Function for incrementing the rf channel tx lbt failures.
void	zpal_radio_rf_channel_statistic_background_rssi_average_update (zpal_radio_zwave_channel_t zwavechannel, int8_t rssi) Function for updating the rf channel background noise rssi average statistic.

void	zpal_radio_rf_channel_statistic_end_device_rssi_average_update (zpal_radio_zwave_channel_t zwavechannel, int8_t rssi) Function for updating the rf channel end device noise rssi average statistic.
zpal_status_t	zpal_radio_get_background_rssi (uint8_t channel, int8_t *rssi) Returns the background RSSI.
zpal_tx_power_t	zpal_radio_get_default_tx_power (void) Function for getting the default tx power in deci dBm.
zpal_radio_tx_power_t	zpal_radio_get_reduce_tx_power (void) Function for getting the current reduce RF tx power compared to the default normal power in dBm.
void	zpal_radio_enable_flirs (void) Allows the radio to go into FLIRS receive mode.
bool	zpal_radio_is_flirs_enabled (void) Returns whether FLIRS mode is enabled in the radio.
void	zpal_radio_start_receive_after_power_down (bool wait_for_beam) Starts the receiver after power down.
void	zpal_radio_abort (void) Turn radio off without changing configuration.
void	zpal_radio_reset_after_beam_receive (bool start_receiver) Resets the radio configuration to receive mode after having received a beam.
bool	zpal_radio_is_fragmented_beam_enabled (void) Returns whether use of fragmented beams is enabled or not for the active region.
void	zpal_radio_calibrate (void) Calibrates the radio.
bool	zpal_radio_is_lbt_enabled (void) Returns whether listen before talk (LBT) is enabled.
uint16_t	zpal_radio_get_beam_startup_time (void) Returns the time it takes to start transmission of wake up beams.
node_id_t	zpal_radio_get_beam_node_id (void) Returns the node ID associated with most recently received beam frame.
zpal_tx_power_t	zpal_radio_get_minimum_lr_tx_power (void) Returns the minimum transmit power for Z-Wave Long Range.
zpal_tx_power_t	zpal_radio_get_maximum_lr_tx_power (void) Returns the maximum transmit power for Z-Wave Long Range.
bool	zpal_radio_is_debug_enabled (void) Returns whether debug is enabled or disabled.
void	zpal_radio_debug_configure (bool enable) Configures the radio debug mode.
const zpal_radio_profile_t *	zpal_radio_get_rf_profile (void) a getter on the current rf profile.
zpal_radio_lr_channel_config_t	zpal_radio_get_lr_channel_config (void) Function to read current Long Range Channel Configuration.

zpal_radio_lr_channel_t	zpal_radio_get_primary_long_range_channel (void) Function to read current Primary Long Range Channel.
void	zpal_radio_set_primary_long_range_channel (zpal_radio_lr_channel_t channel) Function to set the Primary Long Range Channel.
bool	zpal_radio_get_long_range_channel_auto_mode (void) Function to read current Long Range Channel selection mode.
void	zpal_radio_set_long_range_channel_auto_mode (bool enable) Function to set the Long Range Channel selection mode.
void	zpal_radio_set_long_range_lock (bool lock) Function to set the Long Range channel Locked status.
bool	zpal_radio_is_long_range_locked (void) Function to read the Long Range channel Locked status.
bool	zpal_radio_is_region_supported (zpal_radio_region_t region) Function to check if the stack implementation supports a given region.
int8_t	zpal_radio_get_flirs_beam_tx_power (void) Read the saved tx power of the last received long-range beam.
bool	zpal_radio_is_transmit_allowed (uint8_t channel, uint8_t frame_length, uint8_t frame_priority) Check if transmission is allowed for specified channel.
bool	zpal_radio_attenuate (zpal_radio_tx_power_t adjust_tx_power) Function to reduce Tx power of classic non-listening devices.
zpal_tx_power_t	zpal_radio_get_maximum_tx_power (void) Return the maximum board supported tx power for classic z-wave.
void	zpal_radio_request_calibration (bool forced) Function to radio calibration.

Macros

#define	ZPAL_RADIO_INVALID_RSSI_DBM (-128) RSSI value is invalid or not measured.
#define	ZPAL_RADIO_RSSI_NOT_AVAILABLE (127)
#define	ZW_TX_POWER_10DBM 100
#define	ZW_TX_POWER_14DBM 140
#define	ZW_TX_POWER_20DBM 200
#define	REGION_2CH_NUM = (REGION_2CH_END - REGION_2CH_FIRST) Number of 2 channel regions.
#define	REGION_3CH_NUM = (REGION_3CH_END - REGION_3CH_FIRST) Number of 3 channel regions.

Enumeration Documentation

zpal_radio_wakeup_t

zpal_radio_wakeup_t

Wakeup interval for the radio.

A FLiRS node will use 250 or 1000 ms interval, all other nodes should be configured as always listening.

Enumerator	
ZPAL_RADIO_WAKEUP_NEVER_LISTEN	Node is not listening (Only listen when application requests it).
ZPAL_RADIO_WAKEUP_ALWAYS_LISTEN	Node is always listening.
ZPAL_RADIO_WAKEUP_EVERY_250ms	Node wakes up every 250 ms interval to listen for a wakeup beam.
ZPAL_RADIO_WAKEUP_EVERY_1000ms	Node wakes up every 1000 ms interval to listen for a wakeup beam.

zpal_radio_crc_t

zpal_radio_crc_t

Enumeration containing supported checksum types in Z-Wave.

Enumerator	
ZPAL_RADIO_CRC_NONE	No checksum.
ZPAL_RADIO_CRC_8_BIT_XOR	8 bit XOR checksum.
ZPAL_RADIO_CRC_16_BIT_CCITT	16 bit CRC-CCITT checksum.

zpal_radio_speed_t

zpal_radio_speed_t

Enumeration containing supported baud rates.

Enumerator	
ZPAL_RADIO_SPEED_UNDEFINED	
ZPAL_RADIO_SPEED_9600	
ZPAL_RADIO_SPEED_40K	
ZPAL_RADIO_SPEED_100K	
ZPAL_RADIO_SPEED_100KLR	

zpal_radio_lr_channel_t

zpal_radio_lr_channel_t

Enumeration containing Long Range Channels.

Enumerator	
ZPAL_RADIO_LR_CHANNEL_UNINITIALIZED	Long Range Channel setting not initialized.
ZPAL_RADIO_LR_CHANNEL_A	Long Range Channel A.
ZPAL_RADIO_LR_CHANNEL_B	Long Range Channel B.

ZPAL_RADIO_LR_CHANNEL_UNKNOWN	Long Range Channel Unknown.
ZPAL_RADIO_LR_CHANNEL_AUTO	Long Range automatically selected Channel.

zpal_radio_zwave_channel_t

zpal_radio_zwave_channel_t

Enumeration containing Z-Wave channels.

Enumerator	
ZPAL_RADIO_ZWAVE_CHANNEL_0	Z-Wave channel 0.
ZPAL_RADIO_ZWAVE_CHANNEL_1	Z-Wave channel 1.
ZPAL_RADIO_ZWAVE_CHANNEL_2	Z-Wave channel 2.
ZPAL_RADIO_ZWAVE_CHANNEL_3	Z-Wave channel 3.
ZPAL_RADIO_ZWAVE_CHANNEL_4	Z-Wave channel 4.
ZPAL_RADIO_ZWAVE_CHANNEL_NUM	Number of Z-Wave channels. must be last.
ZPAL_RADIO_ZWAVE_CHANNEL_UNKNOWN	Z-Wave channel Unknown.

zpal_radio_protocol_mode_t

zpal_radio_protocol_mode_t

Enumeration containing Z-Wave channels known as protocol modes.

Enumerator	
ZPAL_RADIO_PROTOCOL_MODE_1	2 Channel Protocol Mode.
ZPAL_RADIO_PROTOCOL_MODE_2	3 Channel Protocol Mode.
ZPAL_RADIO_PROTOCOL_MODE_3	4 Channel Protocol Mode - Combination of 2 Channel (9.6kb/40kb, 100kb) and Long Range Channel.
ZPAL_RADIO_PROTOCOL_MODE_4	2 Long Range Channel Protocol Mode - End Device.
ZPAL_RADIO_PROTOCOL_MODE_UNDEFINED	Protocol mode undefined means invalid region code.

zpal_radio_lr_channel_config_t

zpal_radio_lr_channel_config_t

List of LR channel configuration.

Used to select the correct phy configuration.

Enumerator	
ZPAL_RADIO_LR_CH_CFG_NO_LR	used if region does not support Long range or end device included in classic network.
ZPAL_RADIO_LR_CH_CFG1	used by controller with primary channel A and not included end device Phy is configured to listen on classic Z-Wave channels and Long Range channel A.
ZPAL_RADIO_LR_CH_CFG2	used only by controller with primary channel B Phy is configured to listen on classic Z-Wave channels and Long Range channel B.

ZPAL_RADIO_LR_CH_CFG3	used only by end device when included in a LR network.
ZPAL_RADIO_LR_CH_CFG_COUNT	enumeration items count

zpal_radio_region_t

zpal_radio_region_t

Enumeration official Z-Wave regions.

Enumerator	
REGION_2CH_FIRST	First 2 channels region (with 3 data rates)
REGION_EU	Radio is located in Region EU. 2 Channel region.
REGION_US	Radio is located in Region US. 2 Channel region.
REGION_ANZ	Radio is located in Region Australia/New Zealand. 2 Channel region.
REGION_HK	Radio is located in Region Hong Kong. 2 Channel region.
REGION_DEPRECATED_4	Deprecated value, should never be used.
REGION_IN	Radio is located in Region India. 2 Channel region.
REGION_IL	Radio is located in Region Israel. 2 Channel region.
REGION_RU	Radio is located in Region Russia. 2 Channel region.
REGION_CN	Radio is located in Region China. 2 Channel region.
REGION_US_LR	Radio is located in Region US. 2 Channel LR region.
REGION_DEPRECATED_10	Deprecated value, should never be used.
REGION_EU_LR	Radio is located in Region EU. 2 Channel LR region.
REGION_2CH_END	end of 2 channels regions (with 3 data rates)
REGION_3CH_FIRST	First 3 channels region (with 1 data rate)
REGION_JP	Radio is located in Region Japan. 3 Channel region.
REGION_KR	Radio is located in Region Korea. 3 Channel region.
REGION_3CH_END	end of 3 channels regions (with 1 data rate)
REGION_DEPRECATED_48	Deprecated value, should never be used.
REGION_UNDEFINED	
REGION_DEFAULT	Radio is located in Library Default Region EU. 2 Channel region.

zpal_radio_tx_power_t

zpal_radio_tx_power_t

Enumeration containing Tx power settings.

Enumerator	
ZPAL_RADIO_TX_POWER_DEFAULT	Max Tx power.
ZPAL_RADIO_TX_POWER_MINUS1_DBM	
ZPAL_RADIO_TX_POWER_MINUS2_DBM	
ZPAL_RADIO_TX_POWER_MINUS3_DBM	
ZPAL_RADIO_TX_POWER_MINUS4_DBM	

ZPAL_RADIO_TX_POWER_MINUS5_DBM	
ZPAL_RADIO_TX_POWER_MINUS6_DBM	
ZPAL_RADIO_TX_POWER_MINUS7_DBM	
ZPAL_RADIO_TX_POWER_MINUS8_DBM	
ZPAL_RADIO_TX_POWER_MINUS9_DBM	
ZPAL_RADIO_TX_POWER_UNINITIALIZED	
ZPAL_RADIO_TX_POWER_REDUCED	Default reduced Tx power.

zpal_radio_header_type_t

zpal_radio_header_type_t

Z-Wave Frame header format types.

Enumerator	
ZPAL_RADIO_HEADER_TYPE_2CH	2 Channel header format.
ZPAL_RADIO_HEADER_TYPE_3CH	3 Channel header format.
ZPAL_RADIO_HEADER_TYPE_LR	LR Channel header format.
ZPAL_RADIO_HEADER_TYPE_UNDEFINED	Undefined Channel header format.

zpal_radio_event_t

zpal_radio_event_t

Enumeratio radio events.

Enumerator	
ZPAL_RADIO_EVENT_NONE	
ZPAL_RADIO_EVENT_RX_COMPLETE	Frame received.
ZPAL_RADIO_EVENT_TX_COMPLETE	Transmit complete.
ZPAL_RADIO_EVENT_RX_BEAM_COMPLETE	Beam received.
ZPAL_RADIO_EVENT_TX_BEAM_COMPLETE	Beam sent.
ZPAL_RADIO_EVENT_RX_ABORT	Receive was aborted.
ZPAL_RADIO_EVENT_TX_FAIL	Transmit failed.
ZPAL_RADIO_EVENT_TX_FAIL_LBT	Transmit failed because of an LBT failure.
ZPAL_RADIO_EVENT_RXTX_CALIBRATE	Radio needs calibration.
ZPAL_RADIO_EVENT_MASK	
ZPAL_RADIO_EVENT_FLAG_SUCCESS	Indicates a successful event.
ZPAL_RADIO_EVENT_TX_TIMEOUT	

zpal_radio_status_t

zpal_radio_status_t

Enumerator

ZPAL_RADIO_STATUS_IDLE	
ZPAL_RADIO_STATUS_RX	
ZPAL_RADIO_STATUS_TX	RX in progress.
ZPAL_RADIO_STATUS_TX_BEAM	TX in progress.
ZPAL_RADIO_STATUS_RX_BEAM	Beam TX in progress.

zpal_radio_application_t

zpal_radio_application_t

Enumerator

ZPAL_RADIO_APPLICATION_DEFAULT	Normal mode.
ZPAL_RADIO_APPLICATION_ZNIFFER	Zniffer mode.

Typedef Documentation

node_id_t

typedef uint16_t node_id_t

Node ID type.

zpal_tx_power_t

typedef int16_t zpal_tx_power_t

Parameter type to store deci dBm values.

zpal_radio_callback_t

typedef void(* zpal_radio_callback_t) (const zpal_radio_event_t event) (const zpal_radio_event_t event)

zpal_radio_receive_handler_t

typedef void(* zpal_radio_receive_handler_t) (zpal_radio_rx_parameters_t *rx_parameters, zpal_radio_receive_frame_t *frame) (zpal_radio_rx_parameters_t *rx_parameters, zpal_radio_receive_frame_t *frame)

Function pointer declaration for handling of received frames.

Parameters

Type	Direction	Argument Name	Description
	[in]	rx_parameters	Pointer to the structure with channel and rssi values.

Type	Direction	Argument Name	Description
	[in]	frame	Pointer to the received frame. The frame is expected to be located in Z-Wave stack reserved memory and allocated throughout lifetime of stack processing. Application should copy payload data if required for unsynchronized data processing.

When a frame is received this function will be invoked in order to process the frame.

Function Documentation

zpal_radio_set_network_ids

```
void zpal_radio_set_network_ids (uint32_t home_id, node_id_t node_id, uint8_t home_id_hash)
```

Sets the home ID & the node ID.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	home_id	Network home ID.
node_id_t	[in]	node_id	Network node ID.
uint8_t	[in]	home_id_hash	HomeID Hash. Used only for Flirs.

zpal_radio_init

```
void zpal_radio_init (zpal_radio_profile_t *const profile)
```

Initializes the radio.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_profile_t *const	[in]	profile	Pointer to the profile with information to configure the radio.

zpal_radio_change_region

```
zpal_status_t zpal_radio_change_region (zpal_radio_region_t eRegion, zpal_radio_lr_channel_config_t eLrChCfg)
```

Function used to change region and Long Range mode at the same time.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_region_t	[in]	eRegion	Region to change to.
zpal_radio_lr_channel_config_t	[in]	eLrChCfg	long range mode to change to (in case of no LR region, should be set to ZPAL_RADIO_LR_CH_CFG_NO_LR).

Returns

- ZPAL_STATUS_OK if the region was successfully changed and ZPAL_STATUS_FAIL otherwise.

zpal_radio_get_region

```
zpal_radio_region_t zpal_radio_get_region (void )
```

Function for getting REGION runtime.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Current region.

zpal_radio_transmit

```
zpal_status_t zpal_radio_transmit (zpal_radio_transmit_parameter_t const *const tx_parameters, uint8_t frame_header_length, uint8_t const *const frame_header_buffer, uint8_t frame_payload_length, uint8_t const *const frame_payload_buffer)
```

Function for transmitting a Z-Wave frame though the radio.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_transmit_parameter_t const *const	[in]	tx_parameters	Parameter setting specifying speed, channel, wakeup, crc for transmission.
uint8_t	[in]	frame_header_length	Length of frame header data to transmit.
uint8_t const *const	[in]	frame_header_buffer	Pointer to data array containing the frame header.
uint8_t	[in]	frame_payload_length	Length of frame payload data to transmit.
uint8_t const *const	[in]	frame_payload_buffer	Pointer to data array containing the frame payload.

Returns

- [ZPAL_STATUS_OK](#) if the data was successfully transmit, [ZPAL_STATUS_BUFFER_FULL](#) when queue is full.

zpal_radio_transmit_beam

```
zpal_status_t zpal_radio_transmit_beam (zpal_radio_transmit_parameter_t const *const tx_parameters, uint8_t beam_data_len, uint8_t const *const beam_data)
```

Function for transmitting a Z-Wave Beam frame though the radio.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_transmit_parameter_t const *const	[in]	tx_parameters	Parameter setting specifying speed, channel, wakeup.
uint8_t	[in]	beam_data_len	Length of the Beam data to transmit.
uint8_t const *const	[in]	beam_data	Pointer to data array containing the BEAM data.

Returns

- [ZPAL_STATUS_OK](#) if the data was successfully transmit, [ZPAL_STATUS_BUFFER_FULL](#) when queue is full.

zpal_radio_start_receive

```
void zpal_radio_start_receive (void )
```

Starts the receiver and enables reception of frames.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

If the receiver is already started, nothing will happen.

zpal_radio_get_last_received_frame

```
void zpal_radio_get_last_received_frame (void )
```

Function to get last received frame.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

If a frame is received, [zpal_radio_receive_handler_t](#) will be invoked.

zpal_radio_power_down

```
void zpal_radio_power_down (void )
```

Powers down the radio transceiver.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_get_protocol_mode

```
zpal_radio_protocol_mode_t zpal_radio_get_protocol_mode (void )
```

Function to get the protocol mode used in the configured region.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Protocol mode used in the configured region.

zpal_radio_get_last_beam_channel

```
zpal_radio_zwave_channel_t zpal_radio_get_last_beam_channel (void )
```

Get last beam channel.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Retrieve the channel on which we have last received a beam on.

Returns

- Last beam channel.

zpal_radio_get_last_beam_rssi

```
int8_t zpal_radio_get_last_beam_rssi (void )
```

Get last beam RSSI.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Retrieve the RSSI of the last beam received.

Returns

- Last beam rssi.

zpal_radio_set_lbt_level

```
void zpal_radio_set_lbt_level (uint8_t channel, int8_t level)
```

Function for setting the LBT RSSI level.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	channel	uint8_t channel to set LBT threshold for.
int8_t	[in]	level	int8_t LBT RSSI level in dBm.

zpal_radio_enable_rx_broadcast_beam

```
void zpal_radio_enable_rx_broadcast_beam (bool enable)
```

Enable or disables reception of broadcast beam.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	true to enable FLiRS broadcast address, false to disable.

Enable or disable FLiRS broadcast address.

zpal_radio_clear_tx_timers

```
void zpal_radio_clear_tx_timers (void )
```

Function for clearing current Channel Transmit timers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_clear_network_stats

```
void zpal_radio_clear_network_stats (void )
```

Function for clearing current Network statistics.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_rf_channel_statistic_clear

```
void zpal_radio_rf_channel_statistic_clear (zpal_radio_zwave_channel_t zwavechannel)
```

Function for clearing specified rf channel statistics.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_zwave_channel_t	N/A	zwavechannel	rf channel for which statistics will be cleared.

zpal_radio_rf_channel_statistic_get

```
bool zpal_radio_rf_channel_statistic_get (zpal_radio_zwave_channel_t zwavechannel, zpal_radio_rf_channel_statistic_t * p_radio_channel_statistic)
```

Parameters

Type	Direction	Argument Name	Description
zpal_radio_zwave_channel_t	N/A	zwavechannel	zwave channel for which statistics will be returned.
zpal_radio_rf_channel_statistic_t *	N/A	p_radio_channel_statistic	pointer to structure where statistics for specified zwave channel should be copied.

Returns

- true If delivered structure has been filled with current statistic for specified zwavechannel.
- false If delivered structure has NOT been filled with any channel statistics.

zpal_radio_rf_channel_statistic_tx_channel_set

```
void zpal_radio_rf_channel_statistic_tx_channel_set (zpal_radio_zwave_channel_t zwavechannel)
```

Function for setting the zwavechannel used when calling `zpal_radio_rf_channel_statistic_tx_frames`, `zpal_radio_rf_channel_statistic_tx_retries` and `zpal_radio_rf_channel_statistic_tx_lbt_failures`.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_zwave_channel_t	N/A	zwavechannel	Z-Wave channel current rf channel statistic 'tx channel' should be set to.

zpal_radio_rf_channel_statistic_tx_frames

```
void zpal_radio_rf_channel_statistic_tx_frames (void )
```

Function for incrementing the rf channel tx frame statistic.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_rf_channel_statistic_tx_retries

```
void zpal_radio_rf_channel_statistic_tx_retries (void )
```

Function for incrementing the rf channel tx retries statistic.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_rf_channel_statistic_tx_lbt_failures

```
void zpal_radio_rf_channel_statistic_tx_lbt_failures (void )
```

Function for incrementing the rf channel tx lbt failures.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_rf_channel_statistic_background_rssi_average_update

```
void zpal_radio_rf_channel_statistic_background_rssi_average_update (zpal_radio_zwave_channel_t zwavechannel, int8_t rssi)
```

Function for updating the rf channel background noise rssi average statistic.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_zwave_channel_t	N/A	zwavechannel	Z-Wave channel on which background noise rssi average should be updated.
int8_t	N/A	rssi	rssi to add to sampleset used for calculating average background noise rssi on specified Z-Wave channel.

zpal_radio_rf_channel_statistic_end_device_rssi_average_update

```
void zpal_radio_rf_channel_statistic_end_device_rssi_average_update (zpal_radio_zwave_channel_t zwavechannel, int8_t rssi)
```

Function for updating the rf channel end device noise rssi average statistic.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_zwave_channel_t	N/A	zwavechannel	Z-Wave channel on which end device noise rssi average should be updated.
int8_t	N/A	rssi	rssi to add to sampleset used for calculating average end device noise rssi on specified Z-Wave channel.

zpal_radio_get_background_rssi

```
zpal_status_t zpal_radio_get_background_rssi (uint8_t channel, int8_t * rssi)
```

Returns the background RSSI.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	channel	uint8_t channel Id for measurement.
int8_t *	[out]	rssi	pointer to background RSSI.

Returns

- [ZPAL_STATUS_OK](#) if a valid RSSI value is available and read. [ZPAL_STATUS_FAIL](#) if an RSSI value cannot be read. The value of *rssi is invalid. [ZPAL_STATUS_INVALID_ARGUMENT](#) if rssi is NULL.

zpal_radio_get_default_tx_power

zpal_tx_power_t zpal_radio_get_default_tx_power (void)

Function for getting the default tx power in deci dBm.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The default RF TX power in deci dBm

zpal_radio_get_reduce_tx_power

zpal_radio_tx_power_t zpal_radio_get_reduce_tx_power (void)

Function for getting the current reduce RF tx power compared to the default normal power in dBm.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The current reduce RF TX power in dBm.

zpal_radio_enable_flirs

void zpal_radio_enable_flirs (void)

Allows the radio to go into FLiRS receive mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_is_flirs_enabled

```
bool zpal_radio_is_flirs_enabled (void )
```

Returns whether FLIRS mode is enabled in the radio.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True when FLIRS mode is enabled.

zpal_radio_start_receive_after_power_down

```
void zpal_radio_start_receive_after_power_down (bool wait_for_beam)
```

Starts the receiver after power down.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	wait_for_beam	If set to true, the radio will listen for a beam. Otherwise, it will listen normally.

zpal_radio_abort

```
void zpal_radio_abort (void )
```

Turn radio off without changing configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_radio_reset_after_beam_receive

```
void zpal_radio_reset_after_beam_receive (bool start_receiver)
```

Resets the radio configuration to receive mode after having received a beam.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	start_receiver	If set to true, the receiver will start listening. Otherwise, it will stay inactive.

zpal_radio_is_fragmented_beam_enabled

```
bool zpal_radio_is_fragmented_beam_enabled (void )
```

Returns whether use of fragmented beams is enabled or not for the active region.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if use of fragmented beams is enabled, false otherwise.

zpal_radio_calibrate

```
void zpal_radio_calibrate (void )
```

Calibrates the radio.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Z-Wave expects the radio ZPAL implementation to generate a [ZPAL_RADIO_EVENT_RXTX_CALIBRATE](#) event when calibration is required. The event will invoke this function in non-interrupt context.

zpal_radio_is_lbt_enabled

```
bool zpal_radio_is_lbt_enabled (void )
```

Returns whether listen before talk (LBT) is enabled.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if LBT is enabled, false otherwise.

zpal_radio_get_beam_startup_time

```
uint16_t zpal_radio_get_beam_startup_time (void )
```

Returns the time it takes to start transmission of wake up beams.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Includes time spent on LBT.

Returns

- startup time in milli seconds.

zpal_radio_get_beam_node_id

```
node_id_t zpal_radio_get_beam_node_id (void )
```

Returns the node ID associated with most recently received beam frame.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Node ID associated with the most recently received beam frame.

zpal_radio_get_minimum_lr_tx_power

```
zpal_tx_power_t zpal_radio_get_minimum_lr_tx_power (void )
```

Returns the minimum transmit power for Z-Wave Long Range.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Minimum TX power in dBm.

zpal_radio_get_maximum_lr_tx_power

```
zpal_tx_power_t zpal_radio_get_maximum_lr_tx_power (void )
```

Returns the maximum transmit power for Z-Wave Long Range.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- Maximum TX power in dBm.

zpal_radio_is_debug_enabled

```
bool zpal_radio_is_debug_enabled (void )
```


Returns whether debug is enabled or disabled.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if debug is enabled, false otherwise.

zpal_radio_debug_configure

```
void zpal_radio_debug_configure (bool enable)
```

Configures the radio debug mode.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	A boolean value to enable or disable the radio debug mode. • true: Enable debug mode. • false: Disable debug mode.

This function enables or disables the radio debug mode based on the provided parameter. When enabled, additional debug information may be available for troubleshooting and development purposes.

zpal_radio_get_rf_profile

```
const zpal_radio_profile_t * zpal_radio_get_rf_profile (void )
```

a getter on the current rf profile.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Function return a pointer (instead of a struct) to reduce RAM memory usage and execution time. The pointer target a const structure because it should not be used to modify the content of the structure.

Returns

- pointer on the current rf profile.

zpal_radio_get_lr_channel_config

```
zpal_radio_lr_channel_config_t zpal_radio_get_lr_channel_config (void )
```

Function to read current Long Range Channel Configuration.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- current long rang channel configuration (ZPAL_RADIO_LR_Ch_CFG if region without lr)

zpal_radio_get_primary_long_range_channel

```
zpal_radio_lr_channel_t zpal_radio_get_primary_long_range_channel (void )
```

Function to read current Primary Long Range Channel.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- [ZPAL_RADIO_LR_CHANNEL_A](#) or [ZPAL_RADIO_LR_CHANNEL_B](#).

zpal_radio_set_primary_long_range_channel

```
void zpal_radio_set_primary_long_range_channel (zpal\_radio\_lr\_channel\_t channel)
```

Function to set the Primary Long Range Channel.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_lr_channel_t	[in]	channel	zpal_radio_lr_channel_t Long Range Channel to set as Primary Long Range Channel.

zpal_radio_get_long_range_channel_auto_mode

```
bool zpal_radio_get_long_range_channel_auto_mode (void )
```

Function to read current Long Range Channel selection mode.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- true for automatically slected channel, false for manually selected channel

zpal_radio_set_long_range_channel_auto_mode

```
void zpal_radio_set_long_range_channel_auto_mode (bool enable)
```

Function to set the Long Range Channel selction mode.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	true to enable the automatically channel selection mode, false to enable the manual channel selection mode

zpal_radio_set_long_range_lock

```
void zpal_radio_set_long_range_lock (bool lock)
```

Function to the set Long Range channel Locked status.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	lock	Long Range channel Locked status.

zpal_radio_is_long_range_locked

```
bool zpal_radio_is_long_range_locked (void )
```

Function to read the Long Range channel Locked status.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if node shall use Long Range channel only.

zpal_radio_is_region_supported

```
bool zpal_radio_is_region_supported (zpal_radio_region_t region)
```

Function to check if the stack implementation supports a given region.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_region_t	[in]	region	Region to check

Returns

- True if the region is supported, False if it is not

zpal_radio_get_flirs_beam_tx_power

```
int8_t zpal_radio_get_flirs_beam_tx_power (void )
```

Read the saved tx power of the last received long-range beam.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The tx power of the last received long-range beam.

zpal_radio_is_transmit_allowed

```
bool zpal_radio_is_transmit_allowed (uint8_t channel, uint8_t frame_length, uint8_t frame_priority)
```

Check if transmission is allowed for specified channel.

Parameters

Type	Direction	Argument Name	Description
uint8_t	[in]	channel	The channel to check.
uint8_t	[in]	frame_length	The length of the frame to send.
uint8_t	[in]	frame_priority	The Tx priority of the frame.

Returns

- True if node shall use Long Range channel only.

Note

- In Japan all communication must comply to a max 100ms transmit followed by min 100ms silence period.
- Cause a channel shift.

zpal_radio_attenuate

```
bool zpal_radio_attenuate (zpal_radio_tx_power_t adjust_tx_power)
```

Function to reduce Tx power of classic non-listening devices.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_tx_power_t	[in]	adjust_tx_power	Reduces the devices default Tx power in dB. Valid range: 0-9 dB.

Returns

- True when reduction is allowed, false for listening devices and out of range input.

zpal_radio_get_maximum_tx_power

```
zpal_tx_power_t zpal_radio_get_maximum_tx_power (void )
```

Return the maximum board supported tx power for classic z-wave.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- The maximum board supported tx power in deci dBm.

zpal_radio_request_calibration

```
void zpal_radio_request_calibration (bool forced)
```

Function to radio calibration.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	forced	If true, radio calibration is performed regardless if it is required. If false, radio calibration is performed only if it is required.

zpal_radio_transmit_parameter_t

Parameters required when transmitting a frame.

Public Attributes

<code>zpal_radio_speed_t</code>	<code>speed</code> Channel Speed to use when transmitting a frame.
<code>zpal_radio_zwave_channel_t</code>	<code>channel_id</code> Channel id to use when transmitting a frame.
<code>zpal_radio_crc_t</code>	<code>crc</code> CRC Type to use. XOR is used in 2 channel at 9600 and 40k baud rate. CRC CCITT is used in 100k frames.
<code>uint8_t</code>	<code>preamble</code> The byte value used for the preamble sequence.
<code>uint8_t</code>	<code>preamble_length</code> Length of the preamble. Minimum preamble length is specified in ITU G.9959-2015.
<code>uint8_t</code>	<code>start_of_frame</code> The start of frame byte used to indicate the end of preamble and the start of frame.
<code>uint8_t</code>	<code>repeats</code> Number of repetitions for the frame. This is used for wakeup beams where the beam is to be repeated for 250 or 1000 ms.
<code>uint8_t</code>	<code>use_lbt</code> 1 to transmit frame with LBT, 0 without LBT
<code>zpal_radio_tx_power_t</code>	<code>tx_power</code> The RF tx power to use for transmitting in dBm.
<code>zpal_time_t</code>	<code>turnaround_ref_tick</code> Timestamp (in zpal base time) that must be used as reference for the Rx-To-Tx turnaround time.

Public Attribute Documentation

speed

```
zpal_radio_speed_t zpal_radio_transmit_parameter_t::speed
```

Channel Speed to use when transmitting a frame.

channel_id

```
zpal_radio_zwave_channel_t zpal_radio_transmit_parameter_t::channel_id
```

Channel id to use when transmitting a frame.

crc

```
zpal_radio_crc_t zpal_radio_transmit_parameter_t::crc
```

CRC Type to use. XOR is used in 2 channel at 9600 and 40k baud rate. CRC CCITT is used in 100k frames.

preamble

```
uint8_t zpal_radio_transmit_parameter_t::preamble
```

The byte value used for the preamble sequence.

preamble_length

```
uint8_t zpal_radio_transmit_parameter_t::preamble_length
```

Length of the preamble. Minimum preamble length is specified in ITU G.9959-2015.

start_of_frame

```
uint8_t zpal_radio_transmit_parameter_t::start_of_frame
```

The start of frame byte used to indicate the end of preamble and the start of frame.

repeats

```
uint8_t zpal_radio_transmit_parameter_t::repeats
```

Number of repetitions for the frame. This is used for wakeup beams where the beam is to be repeated for 250 or 1000 ms.

use_lbt

```
uint8_t zpal_radio_transmit_parameter_t::use_lbt
```

1 to transmit frame with LBT, 0 without LBT

tx_power

```
zpal_radio_tx_power_t zpal_radio_transmit_parameter_t::tx_power
```

The RF tx power to use for transmitting in dBm.

turnaround_ref_tick

```
zpal_time_t zpal_radio_transmit_parameter_t::turnaround_ref_tick
```

Timestamp (in zpal base time) that must be used as reference for the Rx-To-Tx turnaround time.

Mostly used for Ack. If not 0, the zpal must guarantee the minimum Rx-To-Tx turnaround time before the Tx. Not done with the TxQueue delay because:

- TxQueue delay is 1ms base time which is not enough (Rx-to-Tx turnaround time is 1ms)
- Ack is high priority and should "lock" the TxQueue without any delay (TxQueue delay is handled before adding the frame to the TxQueue).

zpal_radio_rx_parameters_t

Structure with radio parameters for a received frame.

Public Attributes

<code>zpal_radio_speed_t</code>	<code>speed</code> Speed for the frame received.
<code>zpal_radio_zwave_channel_t</code>	<code>channel_id</code> Channel id on which the frame was received.
<code>zpal_radio_header_type_t</code>	<code>channel_header_format</code> Z-Wave Header format used in channel frame was received on.
<code>int8_t</code>	<code>rss_i</code> Rssi value.
<code>zpal_time_t</code>	<code>rx_zpal_tick</code> Timestamp when the frame was received.

Public Attribute Documentation

speed

`zpal_radio_speed_t zpal_radio_rx_parameters_t::speed`

Speed for the frame received.

channel_id

`zpal_radio_zwave_channel_t zpal_radio_rx_parameters_t::channel_id`

Channel id on which the frame was received.

channel_header_format

`zpal_radio_header_type_t zpal_radio_rx_parameters_t::channel_header_format`

Z-Wave Header format used in channel frame was received on.

rss_i

`int8_t zpal_radio_rx_parameters_t::rss_i`

Rssi value.

rx_zpal_tick

zpal_time_t zpal_radio_rx_parameters_t:rx_zpal_tick

Timestamp when the frame was received.

zpal_radio_receive_frame_t

Z-Wave receive frame.

Public Attributes

uint8_t	frame_content_length	Length of payload following this frame.
uint8_t	frame_content	Array with complete frame data received.

Public Attribute Documentation

frame_content_length

```
uint8_t zpal_radio_receive_frame_t::frame_content_length
```

Length of payload following this frame.

frame_content

```
uint8_t zpal_radio_receive_frame_t::frame_content[]
```

Array with complete frame data received.

zpal_radio_network_stats_t

Network statistics structure.

Public Attributes

uint32_t	tx_frames	Transmitted Frames.
uint32_t	tx_lbt_back_offs	LBT backoffs.
uint32_t	rx_frames	Received Frames (No errors).
uint32_t	rx_lrc_errors	Checksum Errors.
uint32_t	rx_crc_errors	CRC16 Errors.
uint32_t	rx_foreign_home_id	Foreign Home ID.
uint32_t	tx_time_channel	Accumulated transmission time in ms for channel 0.

Public Attribute Documentation

tx_frames

```
uint32_t zpal_radio_network_stats_t:tx_frames
```

Transmitted Frames.

tx_lbt_back_offs

```
uint32_t zpal_radio_network_stats_t:tx_lbt_back_offs
```

LBT backoffs.

rx_frames

```
uint32_t zpal_radio_network_stats_t:rx_frames
```

Received Frames (No errors).

rx_lrc_errors

uint32_t zpal_radio_network_stats_t::rx_lrc_errors

Checksum Errors.

rx_crc_errors

uint32_t zpal_radio_network_stats_t::rx_crc_errors

CRC16 Errors.

rx_foreign_home_id

uint32_t zpal_radio_network_stats_t::rx_foreign_home_id

Foreign Home ID.

tx_time_channel

uint32_t zpal_radio_network_stats_t::tx_time_channel[ZPAL_RADIO_ZWAVE_CHANNEL_NUM]

Accumulated transmission time in ms for channel 0.

zpal_radio_rf_channel_statistic_t

rf channel statistics structure

Public Attributes

uint32_t	rf_channel_tx_frames	Transmitted frames on rf channel.
uint32_t	rf_channel_tx_retries	Frame transmit retries on rf channel.
uint32_t	rf_channel_tx_lbt_failures	Frame transmit failed by lbt on rf channel.
uint32_t	rf_channel_rx_foreign_homeid	Frames received with foreign homeid on rf channel.
uint32_t	rf_channel_rx_crc_error	Frames received with CRC error on rf channel.
int8_t	rf_channel_background_rssi_average	Background RSSI average on rf channel.
int8_t	rf_channel_end_device_rssi_average	End Device RSSI average on rf channel.

Public Attribute Documentation

rf_channel_tx_frames

uint32_t zpal_radio_rf_channel_statistic_t::rf_channel_tx_frames

Transmitted frames on rf channel.

rf_channel_tx_retries

uint32_t zpal_radio_rf_channel_statistic_t::rf_channel_tx_retries

Frame transmit retries on rf channel.

rf_channel_tx_lbt_failures

uint32_t zpal_radio_rf_channel_statistic_t::rf_channel_tx_lbt_failures

Frame transmit failed by lbt on rf channel.

rf_channel_rx_foreign_homeid

```
uint32_t zpal_radio_rf_channel_statistic_t::rf_channel_rx_foreign_homeid
```

Frames received with foreign homeid on rf channel.

rf_channel_rx_crc_error

```
uint32_t zpal_radio_rf_channel_statistic_t::rf_channel_rx_crc_error
```

Frames received with CRC error on rf channel.

rf_channel_background_rssi_average

```
int8_t zpal_radio_rf_channel_statistic_t::rf_channel_background_rssi_average
```

Background RSSI average on rf channel.

rf_channel_end_device_rssi_average

```
int8_t zpal_radio_rf_channel_statistic_t::rf_channel_end_device_rssi_average
```

End Device RSSI average on rf channel.

zpal_radio_profile_t

Radio Profile containing region, baud rate, and wakeup interval for this device.

Public Attributes

<code>zpal_radio_region_t</code>	<code>region</code> Region in which this system operates.
<code>zpal_radio_wakeup_t</code>	<code>wakeup</code> Wakeup interval for the radio.
<code>zpal_radio_lr_channel_t</code>	<code>primary_lr_channel</code> Primary Long Range Channel.
<code>bool</code>	<code>lr_channel_auto_mode</code> Longe Range channel selection mode.
<code>zpal_radio_lr_channel_config_t</code>	<code>active_lr_channel_config</code> Long Range channel configuration.
<code>int8_t</code>	<code>listen_before_talk_threshold</code> LBT Threshold for Transmit backoff in dBm.
<code>zpal_tx_power_t</code>	<code>tx_power_max</code> Z-Wave Transmit Power in deci dBm.
<code>zpal_tx_power_t</code>	<code>tx_power_adjust</code> Adjustment for antenna gain in deci dBm.
<code>zpal_tx_power_t</code>	<code>tx_power_max_lr</code> Max transmit power for Z-Wave LR in deci dBm.
<code>zpal_radio_callback_k_t</code>	<code>rx_cb</code> Pointer to function called by RF on Rx Completion.
<code>zpal_radio_callback_k_t</code>	<code>tx_cb</code> Pointer to function called by RF on Tx Completion.
<code>zpal_radio_callback_k_t</code>	<code>region_change_cb</code> Pointer to function called by RF on Region change.
<code>zpal_radio_callback_k_t</code>	<code>assert_cb</code> Pointer to function called by RF on fatal Assert.
<code>zpal_radio_network_stats_t *</code>	<code>network_stats</code> Pointer to structure where to RF Statistics are placed.
<code>uint8_t</code>	<code>radio_debug_enable</code> Enable radio debugging which is vendor specific.
<code>zpal_radio_receive_handler_t</code>	<code>receive_handler_cb</code> Pointer to receive handler.
<code>zpal_radio_application_t</code>	<code>radio_application</code> Application type.

Public Attribute Documentation

region


```
zpal_radio_region_t zpal_radio_profile_t::region
```

Region in which this system operates.

wakeup

```
zpal_radio_wakeup_t zpal_radio_profile_t::wakeup
```

Wakeup interval for the radio.

primary_lr_channel

```
zpal_radio_lr_channel_t zpal_radio_profile_t::primary_lr_channel
```

Primary Long Range Channel.

lr_channel_auto_mode

```
bool zpal_radio_profile_t::lr_channel_auto_mode
```

Long Range channel selection mode.

active_lr_channel_config

```
zpal_radio_lr_channel_config_t zpal_radio_profile_t::active_lr_channel_config
```

Long Range channel configuration.

Set only when radio chip configuration is updated. Used to check that requested channel configuration is different than the active one.

listen_before_talk_threshold

```
int8_t zpal_radio_profile_t::listen_before_talk_threshold
```

LBT Threshold for Transmit backoff in dBm.

tx_power_max

```
zpal_tx_power_t zpal_radio_profile_t::tx_power_max
```

Z-Wave Transmit Power in deci dBm.

tx_power_adjust

```
zpal_tx_power_t zpal_radio_profile_t::tx_power_adjust
```

Adjustment for antenna gain in deci dBm.

tx_power_max_lr

```
zpal_tx_power_t zpal_radio_profile_t::tx_power_max_lr
```

Max transmit power for Z-Wave LR in deci dBm.

rx_cb

```
zpal_radio_callback_t zpal_radio_profile_t::rx_cb
```

Pointer to function called by RF on Rx Completion.

tx_cb

```
zpal_radio_callback_t zpal_radio_profile_t::tx_cb
```

Pointer to function called by RF on Tx Completion.

region_change_cb

```
zpal_radio_callback_t zpal_radio_profile_t::region_change_cb
```

Pointer to function called by RF on Region change.

assert_cb

```
zpal_radio_callback_t zpal_radio_profile_t::assert_cb
```

Pointer to function called by RF on fatal Assert.

network_stats

```
zpal_radio_network_stats_t* zpal_radio_profile_t::network_stats
```

Pointer to structure where to RF Statistics are placed.

radio_debug_enable

```
uint8_t zpal_radio_profile_t::radio_debug_enable
```

Enable radio debugging which is vendor specific.

receive_handler_cb

```
zpal_radio_receive_handler_t zpal_radio_profile_t::receive_handler_cb
```

Pointer to receive handler.

radio_application

```
zpal_radio_application_t zpal_radio_profile_t::radio_application
```

Application type.

Number of Channels

Number of Channels

Macros

#define	ZPAL_RADIO_NUM_CHANNELS 3	Number of channels of classic Z-Wave protocol.
#define	ZPAL_RADIO_NUM_CHANNELS_LR_CH_CFG1_2 4	Number of channels of Z-Wave Long Range protocol 100kbps, 40kbps, 9.6kbps and LRA/LRB (channel configuration 1 & 2)
#define	ZPAL_RADIO_NUM_CHANNELS_LR_CH_CFG3 2	LRA and LRB (channel configuration 3)

Zpal-radio-utils

Zpal-radio-utils

Defines a platform abstraction layer for the Z-Wave platform independent radio utils.

Functions

- zpal_radio_protocol_mode_t

zpal_radio_region_get_protocol_mode

(uint32_t region, zpal_radio_lr_channel_config_t eLrChCfg)

Function to get region Protocol mode if supported by Radio.
- bool

zpal_radio_protocol_mode_supports_long_range

(zpal_radio_protocol_mode_t mode)

Function to get if giver Protocol mode supports Z-Wave Long Range.
- bool

zpal_radio_region_is_long_range

(zpal_radio_region_t region)

Function to check if specified region has a long range channel.
- zpal_radio_region_t

zpal_radio_get_valid_region

(zpal_radio_region_t region)

Function to get valid region value.

Function Documentation

zpal_radio_region_get_protocol_mode

zpal_radio_protocol_mode_t zpal_radio_region_get_protocol_mode (uint32_t region, zpal_radio_lr_channel_config_t eLrChCfg)

Function to get region Protocol mode if supported by Radio.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	region	Describe where radio is located. zpal_radio_region_t or @ ref zpal_radio_region_tf_t .
zpal_radio_lr_channel_config_t	[in]	eLrChCfg	Active long range channel configuration.

Returns

- [ZPAL_RADIO_PROTOCOL_MODE_1](#), [ZPAL_RADIO_PROTOCOL_MODE_2](#), [ZPAL_RADIO_PROTOCOL_MODE_3](#) or [ZPAL_RADIO_PROTOCOL_MODE_4](#) if supported by Radio. [ZPAL_RADIO_PROTOCOL_MODE_UNDEFINED](#) if not supported by Radio.

zpal_radio_protocol_mode_supports_long_range

bool zpal_radio_protocol_mode_supports_long_range ([zpal_radio_protocol_mode_t](#) mode)

Function to get if giver Protocol mode supports Z-Wave Long Range.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_protocol_mode_t	[in]	mode	Protocol mode.

Returns

- True if given Protocol mode supports Z-Wave Long Range, false if not.

zpal_radio_region_is_long_range

```
bool zpal_radio_region_is_long_range (zpal\_radio\_region\_t region)
```

Function to check if specified region has a long range channel.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_region_t	[in]	region	Describe where radio is located.

Returns

- True if specified region has long range channel.

zpal_radio_get_valid_region

```
zpal\_radio\_region\_t zpal_radio_get_valid_region (zpal\_radio\_region\_t region)
```

Function to get valid region value.

Parameters

Type	Direction	Argument Name	Description
zpal_radio_region_t	[in]	region	Describe where radio is located.

If region is not specified (default value), it would return region set as default.

Returns

- The valid region value

Zpal-retention-register

Zpal-retention-register

Defines a platform abstraction layer for the Z-Wave retention register.

How to use the retention register API

The retention register should support at least 32 x 32 bit registers which can be retained in deep sleep.

The following outlines an example of use:

1. Store value in retention register using [zpal_retention_register_write\(\)](#).
2. Read value from retention register using [zpal_retention_register_read\(\)](#).

Note:

- Retention registers with index 0-15 are reserved for Z-Wave protocol code.
- Retention registers are frequently written, so it is not recommended to use flash memory as storage.

Functions

[zpal_status_t](#) [zpal_retention_register_read](#)(uint32_t index, uint32_t *data)
Reads a 32-bit value from the specified retention register.

[zpal_status_t](#) [zpal_retention_register_write](#)(uint32_t index, uint32_t value)
Write a 32-bit value to the specified retention register.

size_t [zpal_retention_register_count](#)(void)
Get number of available retention registers.

Macros

```
#define ZPAL_RETENTION_REGISTER_SMARTSTART 0
Reserved for Smart Start.

#define ZPAL_RETENTION_REGISTER_RESPONSEROUTE_1 1
Reserved for response route 1.

#define ZPAL_RETENTION_REGISTER_RESPONSEROUTE_2 2
Reserved for response route 2.

#define ZPAL_RETENTION_REGISTER_RESPONSEROUTE_3 3
Reserved for response route 3.

#define ZPAL_RETENTION_REGISTER_RESPONSEROUTE_4 4
Reserved for response route 4.

#define ZPAL_RETENTION_REGISTER_TXPOWER_RSSI_LR 5
Reserved for Long Range transmit power and RSSI.

#define ZPAL_RETENTION_REGISTER_RESET_INFO 6
Reserved for reset information.

#define ZPAL_RETENTION_REGISTER_PROTOCOL_RESERVED_COUNT 16
Number of registers reserved for the stack.
```

Function Documentation

zpal_retention_register_read

```
zpal_status_t zpal_retention_register_read (uint32_t index, uint32_t * data)
```

Reads a 32-bit value from the specified retention register.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	index	Retention register number (zero-based).
uint32_t *	[out]	data	Pointer to a 32-bit variable where the value can be stored.

Returns

- [ZPAL_STATUS_OK](#) on success, [ZPAL_STATUS_INVALID_ARGUMENT](#) on invalid `index` or `data` and [ZPAL_STATUS_FAIL](#) otherwise.

Note

- Retention Register should support at least 32 objects.

zpal_retention_register_write

```
zpal_status_t zpal_retention_register_write (uint32_t index, uint32_t value)
```

Write a 32-bit value to the specified retention register.

Parameters

Type	Direction	Argument Name	Description
uint32_t	[in]	index	Retention register number (zero-based).
uint32_t	[in]	value	32-bit value to save in retention register.

Returns

- [ZPAL_STATUS_OK](#) on success, [ZPAL_STATUS_INVALID_ARGUMENT](#) on invalid `index` and [ZPAL_STATUS_FAIL](#) otherwise.

Note

- Retention Register should support at least 32 objects.

zpal_retention_register_count

```
size_t zpal_retention_register_count (void )
```

Get number of available retention registers.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- number of retention registers.

Zpal-status

Zpal-status

Defines status codes returned by zpal functions.

Enumerations

```
enum zpal_status_t {
    ZPAL_STATUS_OK = 0
    ZPAL_STATUS_FAIL
    ZPAL_STATUS_BUFFER_FULL
    ZPAL_STATUS_BUSY
    ZPAL_STATUS_INVALID_ARGUMENT
}
```

Enumeration Documentation

zpal_status_t

zpal_status_t

Enumerator	
ZPAL_STATUS_OK	No error.
ZPAL_STATUS_FAIL	Generic error.
ZPAL_STATUS_BUFFER_FULL	There is no more room in the buffer.
ZPAL_STATUS_BUSY	Process is busy.
ZPAL_STATUS_INVALID_ARGUMENT	At least one argument is invalid.

Zpal-uart

Zpal-uart

Defines a platform abstraction layer for the Z-Wave UART.

How to use the UART API

The ZPAL UART API is used by only SerialAPI application, all functions are required.

The following outlines an example of use:

1. To initialize UART interface, invoke [zpal_uart_init\(\)](#).
2. Enable UART by invoking [zpal_uart_enable\(\)](#).
3. Invoke [zpal_uart_transmit\(\)](#) to send data.
4. Use [zpal_uart_transmit_in_progress\(\)](#) to check if transmission is in progress.
5. To get number of bytes available for reading, invoke [zpal_uart_get_available\(\)](#).
6. Read received data with [zpal_uart_receive\(\)](#).

Requirements:

- This interface must implement a non-blocking behavior.
- Transmit and receive buffers are provided by application.
- Transmit and receive buffers should be at least 200 bytes large.

Modules

[zpal_uart_config_t](#)

Enumerations

```
enum    zpal\_uart\_id\_t {  
        ZPAL_UART0  
        ZPAL_UART1  
    }  
    IDs for each of the UARTs.  
  
enum    zpal\_uart\_parity\_bit\_t {  
        ZPAL_UART_NO_PARITY  
        ZPAL_UART_EVEN_PARITY  
        ZPAL_UART_ODD_PARITY  
    }  
    IDs for each of the parity mode.  
  
enum    zpal\_uart\_stop\_bits\_t {  
        ZPAL_UART_STOP_BITS_0P5  
        ZPAL_UART_STOP_BITS_1  
        ZPAL_UART_STOP_BITS_1P5  
        ZPAL_UART_STOP_BITS_2  
    }  
    IDs for each of the stop bits configuration.
```

Typedefs

- typedef void *

[zpal_uart_handle_t](#)

UART handle type.
- typedef void(*

[zpal_uart_receive_callback_t](#)

(zpal_uart_handle_t handle, size_t length)
Defines a type for the UART receive callback that is invoked whenever data is ready.
- typedef void(*

[zpal_uart_transmit_done_t](#)

(zpal_uart_handle_t handle)
Defines a type for the UART transmit done callback that is invoked when data has been transmitted.

Functions

- [zpal_status_t](#)

[zpal_uart_init](#)

(const zpal_uart_config_t *config, zpal_uart_handle_t *handle)
Initializes a UART based on the given configuration.
- [zpal_status_t](#)

[zpal_uart_enable](#)

(zpal_uart_handle_t handle)
Enables a given UART.
- [zpal_status_t](#)

[zpal_uart_disable](#)

(zpal_uart_handle_t handle)
Disables a given UART.
- [zpal_status_t](#)

[zpal_uart_transmit](#)

(zpal_uart_handle_t handle, const uint8_t *data, size_t length, zpal_uart_transmit_done_t tx_cb)
Transmits data using a given UART.
- bool

[zpal_uart_transmit_in_progress](#)

(zpal_uart_handle_t handle)
Returns whether transmission is in progress.
- size_t

[zpal_uart_get_available](#)

(zpal_uart_handle_t handle)
Get the number of bytes ready for reading.
- size_t

[zpal_uart_receive](#)

(zpal_uart_handle_t handle, uint8_t *data, size_t length)
Receive available data into buffer using a given UART.

Macros

- #define

ZPAL_UART_CONFIG_FLAG_BLOCKING (1 << 1)

Set this flag in the configuration if the UART must be configured as a blocking UART.

Enumeration Documentation

zpal_uart_id_t

zpal_uart_id_t	
IDs for each of the UARTs.	
Enumerator	
ZPAL_UART0	
ZPAL_UART1	

zpal_uart_parity_bit_t

zpal_uart_parity_bit_t	
IDs for each of the parity mode.	

Enumerator

ZPAL_UART_NO_PARITY	No parity.
ZPAL_UART_EVEN_PARITY	Even parity.
ZPAL_UART_ODD_PARITY	Odd parity.

zpal_uart_stop_bits_t

zpal_uart_stop_bits_t

IDs for each of the stop bits configuration.

Enumerator

ZPAL_UART_STOP_BITS_0P5	0.5 stop bits.
ZPAL_UART_STOP_BITS_1	1 stop bits.
ZPAL_UART_STOP_BITS_1P5	1.5 stop bits.
ZPAL_UART_STOP_BITS_2	2 stop bits.

Typedef Documentation

zpal_uart_handle_t

typedef void* zpal_uart_handle_t

UART handle type.

zpal_uart_receive_callback_t

typedef void(* zpal_uart_receive_callback_t) (zpal_uart_handle_t handle, size_t length))(zpal_uart_handle_t handle, size_t length)

Defines a type for the UART receive callback that is invoked whenever data is ready.

Parameters

Type	Direction	Argument Name	Description
	[in]	handle	UART handle.
	[in]	length	Length of available data in bytes.

zpal_uart_transmit_done_t

typedef void(* zpal_uart_transmit_done_t) (zpal_uart_handle_t handle))(zpal_uart_handle_t handle)

Defines a type for the UART transmit done callback that is invoked when data has been transmitted.

Parameters

Type	Direction	Argument Name	Description
	[in]	handle	UART handle.

Function Documentation

zpal_uart_init

```
zpal_status_t zpal_uart_init (const zpal\_uart\_config\_t * config, zpal\_uart\_handle\_t * handle)
```

Initializes a UART based on the given configuration.

Parameters

Type	Direction	Argument Name	Description
const zpal_uart_config_t *	[in]	config	Pointer to UART configuration.
zpal_uart_handle_t *	[out]	handle	Outputs a handle to a UART.

Returns

- [ZPAL_STATUS_OK](#) on success and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_uart_enable

```
zpal_status_t zpal_uart_enable (zpal\_uart\_handle\_t handle)
```

Enables a given UART.

Parameters

Type	Direction	Argument Name	Description
zpal_uart_handle_t	[in]	handle	UART handle.

Returns

- [ZPAL_STATUS_OK](#) on success and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_uart_disable

```
zpal_status_t zpal_uart_disable (zpal\_uart\_handle\_t handle)
```

Disables a given UART.

Parameters

Type	Direction	Argument Name	Description
zpal_uart_handle_t	[in]	handle	UART handle.

Returns

- [ZPAL_STATUS_OK](#) on success and [ZPAL_STATUS_FAIL](#) otherwise.

zpal_uart_transmit

```
zpal_status_t zpal_uart_transmit (zpal_uart_handle_t handle, const uint8_t * data, size_t length, zpal_uart_transmit_done_t tx_cb)
```

Transmits data using a given UART.

Parameters

Type	Direction	Argument Name	Description
zpal_uart_handle_t	[in]	handle	UART handle.
const uint8_t *	[in]	data	Pointer to data.
size_t	[in]	length	Length of data.
zpal_uart_transmit_done_t	[in]	tx_cb	Transmission done callback.

Returns

- [ZPAL_STATUS_OK](#) if transmission has started, [ZPAL_STATUS_FAIL](#) otherwise.

Note

- Expect `tx_cb` callback to be invoked in interrupt context.
- This function will block if length of data exceeds internal transmit buffer.

zpal_uart_transmit_in_progress

```
bool zpal_uart_transmit_in_progress (zpal_uart_handle_t handle)
```

Returns whether transmission is in progress.

Parameters

Type	Direction	Argument Name	Description
zpal_uart_handle_t	[in]	handle	UART handle.

Returns

- True if transmission is in progress, false otherwise.

zpal_uart_get_available

```
size_t zpal_uart_get_available (zpal_uart_handle_t handle)
```

Get the number of bytes ready for reading.

Parameters

Type	Direction	Argument Name	Description
zpal_uart_handle_t	[in]	handle	UART handle.

Returns

- Number of bytes in the receive buffer available for reading with [zpal_uart_receive](#)

zpal_uart_receive

```
size_t zpal_uart_receive (zpal_uart_handle_t handle, uint8_t * data, size_t length)
```

Receive available data into buffer using a given UART.

Parameters

Type	Direction	Argument Name	Description
zpal_uart_handle_t	[in]	handle	UART handle.
uint8_t *	[out]	data	The data buffer to receive into
size_t	[in]	length	the maximum number of bytes we'd like to read

Returns

- Number of bytes received into buffer

zpal_uart_config_t

UART configuration.

Public Attributes

zpal_uart_id_t	id	UART ID.
uint8_t *	tx_buffer	Poniter to transmit buffer.
size_t	tx_buffer_len	Size of transmit buffer.
uint8_t *	rx_buffer	Poniter to receive buffer.
size_t	rx_buffer_len	Size of receive buffer.
uint32_t	baud_rate	Baud rate of UART.
uint8_t	data_bits	Defines amount of data bits in the UART frame.
zpal_uart_parity_bit_t	parity_bit	Defines parity bit in the UART frame.
zpal_uart_stop_bits_t	stop_bits	Define stop bit in the UART frame.
zpal_uart_receive_callback_t	receive_callback	Expect this callback to be invoked in interrupt context.
const void *	ptr	This can be used for anything, but might not work for all platform implementations.
uint8_t	flags	Configuration flags.

Public Attribute Documentation

id

```
zpal_uart_id_t zpal_uart_config_t::id
```

UART ID.

tx_buffer

```
uint8_t* zpal_uart_config_t::tx_buffer
```

Poniter to transmit buffer.

tx_buffer_len

```
size_t zpal_uart_config_t::tx_buffer_len
```

Size of transmit buffer.

rx_buffer

```
uint8_t* zpal_uart_config_t::rx_buffer
```

Pointer to receive buffer.

rx_buffer_len

```
size_t zpal_uart_config_t::rx_buffer_len
```

Size of receive buffer.

baud_rate

```
uint32_t zpal_uart_config_t::baud_rate
```

Baud rate of UART.

data_bits

```
uint8_t zpal_uart_config_t::data_bits
```

Defines amount of data bits in the UART frame.

parity_bit

```
zpal_uart_parity_bit_t zpal_uart_config_t::parity_bit
```

Defines parity bit in the UART frame.

stop_bits

```
zpal_uart_stop_bits_t zpal_uart_config_t::stop_bits
```

Define stop bit in the UART frame.

receive_callback

```
zpal_uart_receive_callback_t zpal_uart_config_t::receive_callback
```

Expect this callback to be invoked in interrupt context.

ptr

```
const void* zpal_uart_config_t::ptr
```

This can be used for anything, but might not work for all platform implementations.

flags

```
uint8_t zpal_uart_config_t::flags
```

Configuration flags.

Zpal-watchdog

Zpal-watchdog

Defines a platform abstraction layer for the Z-Wave watchdog.

How to use the watchdog API

The following outlines an example of use:

- 1. Enable the watchdog functionality with `zpal_enable_watchdog()`.
- 2. Periodically invoke `zpal_feed_watchdog()` to restart watchdog.
- 3. If watchdog needs to be disabled, check current state with `zpal_is_watchdog_enabled()`, disable it by invoking `zpal_enable_watchdog()`, then restore previous state.

Note:

- Recommended watchdog timeout period is 1 second.

Functions

- void

`zpal_watchdog_init(void)`
Initialize watchdog before use, typically the next call will be `zpal_enable_watchdog`.
- bool

`zpal_is_watchdog_enabled(void)`
Returns whether the watchdog is enabled.
- void

`zpal_enable_watchdog(bool enable)`
Enables or disables the watchdog based on the input.
- void

`zpal_feed_watchdog(void)`
Feeds the watchdog.

Function Documentation

zpal_watchdog_init

```
void zpal_watchdog_init (void )
```

Initialize watchdog before use, typically the next call will be `zpal_enable_watchdog`.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

zpal_is_watchdog_enabled

```
bool zpal_is_watchdog_enabled (void )
```

Returns whether the watchdog is enabled.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Returns

- True if watchdog is enabled, false if not.

zpal_enable_watchdog

```
void zpal_enable_watchdog (bool enable)
```

Enables or disables the watchdog based on the input.

Parameters

Type	Direction	Argument Name	Description
bool	[in]	enable	true enables the watchdog and false disables it.

zpal_feed_watchdog

```
void zpal_feed_watchdog (void )
```

Feeds the watchdog.

Parameters

Type	Direction	Argument Name	Description
void	N/A		

Migration Guide

Migration Guide

This guide shows how to migrate projects from previous SDKs to a newer one.

7.24.0

Power Manager

It is strongly recommended that applications rely on the `zw_power_manager` abstraction and do not use the `zpal_pm` APIs, see `zw_power_manager_ids.h` for available wrapper.

- **Note**
 - : an application must call `zw_power_manager_init` prior to using `zw_power_manager` APIs. If not, the lock operation will be ignored. This call is introduced in `ZAF_Init()`. For NCP Serial API (End Device/Controller) applications, it is done in `ApplicationInit()` as these apps do not depend on ZAF layer. If your application does not depend on ZAF, please add this function call prior to using `zw_power_manager` APIs.
- If the application still needs to use `zpal_pm` API, then: Any application code previously using `zpal_pm_register`, `zpal_pm_stay_awake`, `zpal_pm_cancel` are expected to align with the following rules:
- `zpal_pm_register` is deprecated. Instead, a power lock ID must be defined in either `ZPAL_PM_AppRadioID` or `ZPAL_PM_AppDeepSleepID`. This ID must be used in all operations of this power lock.
- Any call to `zpal_pm_stay_awake`, `zpal_pm_cancel` should now be aligned with `zw_power_manager_(lock/relock/lock_cancel)`

Watchdog

Apps are now responsible for the watchdog activation by calling "`zpal_watchdog_init`" and "`zpal_enable_watchdog`" functions consecutively. (see sample applications for reference).

Logging System

The Debug Print component was replaced by the ZPAL Log API, which allows filtering log messages by source component and log level.

To ensure compatibility with the new logging system, some files relying on Debug Print may have to be modified by including the header `zpal_log.h`, replacing `DPRINT` and `DPRINTF` with `ZPAL_LOG` and removing any references to the old component.

To enable debug output, either use the **Configure** button of the **Z-Wave Log** component in Simplicity Studio or edit `zw_log_config.h` directly in the project's `config` directory:

- configure the output channels for the 4 log levels (debug, info, warning, error) by setting `ZW_LOG_CHANNEL_<log level>` to an existing IO Stream instance (`vcom` , `rtt` , etc.)
- set the relevant defines `ZPAL_LOG_ENABLE_<component name>` to `1`.

To extend the ZPAL Log API with a new component, add a new entry to the enum `zpal_log_component_t` and update the helper macros in `zpal_log_internal.h`.

main.c

The file `main.c` is now generated by the **System Setup (sl_main)** component. If you previously extended this file with your own initialization functions, these should be relocated and registered using the Event Handler API. The `main.c` file should then be removed.

For more information, refer to the documentation for **System Initialization and Action Processing**.

User Credential Command Class

If your project uses or redefines the `CC_UserCredential_move_credential_and_report` function: remove the `destination_credential_slot` parameter and rename the `source_credential_slot` parameter to `credential_slot`.

The following structs were renamed:

Old name	New name
<code>u3c_user</code>	<code>u3c_user_t</code>
<code>u3c_credential</code>	<code>u3c_credential_t</code>
<code>u3c_event_data_validate</code>	<code>u3c_event_data_validate_t</code>
<code>u3c_event_data_learn_read_done</code>	<code>u3c_event_data_learn_read_done_t</code>
<code>u3c_credential_metadata</code>	<code>u3c_credential_metadata_t</code>
<code>u3c_credential_learn_event_data</code>	<code>u3c_event_data_learn_start_t</code>

Application Specific Hardware Initialization

In `app.c`, ensure that the function `app_hw_init()` is invoked by the `ApplicationInit` function (after `ZAF_Init`, if present). In an unmodified project, this can be done by removing the `#if` preprocessor directives:

```
- #if (!defined(SL_CATALOG_SILICON_LABS_ZWAVE_APPLICATION_PRESENT) && !defined(UNIT_TEST))
  #include "app_hw.h"
- #endif

- #if (!defined(SL_CATALOG_SILICON_LABS_ZWAVE_APPLICATION_PRESENT) && !defined(UNIT_TEST))
  app_hw_init();
- #endif
```

Waking Button Press Detection for Sleeping Applications

If your application has the **Sleeping Reporting** role type, extend the project's `.slcp` file with the following configuration and regenerate the project:

```
configuration:
(...)
- name: APP_BUTTON_PRESS_WAKEUP_DELAY
  value: 46 # Average time for Sleep Timer to start after a reset when built in ocelot
  condition: [device_sdid_210]
- name: APP_BUTTON_PRESS_WAKEUP_DELAY
  value: 285 # Average time for Sleep Timer to start after a reset when built in margay
  condition: [device_sdid_235]
```

This will ensure that the application detects the correct button press duration when waking up from a sleeping state via a button press.

RAIL Power Manager Initialization

The `SL_RAIL_UTIL_RAIL_POWER_MANAGER_INIT` configuration value should be set to `0` for the Door Lock Keypad application and to `1` for all other applications. During the SDK update, this configuration value is automatically set to `1` for all applications, including Door Lock Keypad. Therefore, for the Door Lock Keypad application, it must be set manually to `0`; other applications are handled automatically.

This is necessary for a proper initialization sequence of the power manager component for all applications.

How to upgrade sample applications

In the following sections, an example is shown about the upgrade steps of each sample application.

Steps for an SDK upgrade:

1. Upgrade SDK for the selected application.
 - See the corresponding guide: [Upgrading a Project to a New Software Version](#)
2. Make changes according to the steps described in the current document.
3. Regenerate the application (force regeneration).
4. Build the application

Note: This guide is not exhaustive. Please ensure that any changes you make follow the steps outlined above.

General steps for all applications

All .c files

- Remove `#include "DebugPrint.h"` and `#include "DebugPrintConfig.h"`
- Remove any references to `DEBUGPRINT` because it is deprecated.
- Replace all `DPRINT(...)` and `DPRINTF(...)` API calls to `ZPAL_LOG_DEBUG(ZPAL_LOG_APP, ...)`.
- Replace `DebugPrintf` with `ZPAL_LOG_INFO(ZPAL_LOG_APP, ...)`.
- Remove `DebugPrintConfig(...)` calls.
- Add `#include "zpal_log.h"` if `ZPAL_LOG_...` calls are used.

app.c of all applications

- Remove `m_aDebugPrintBuffer` variable and its references.
- Add `zpal_watchdog_init()` before `zpal_enable_watchdog(true)` in `ApplicationInit`.

<project_name>.slcp

- Remove `sl_system` and `sl_system_kernel` components if present.
- Add `sl_main` component.
- Remove `main.c` from source files.
 - This will be generated by the `sl_main` component after project regeneration.
 - If `main.c` was modified, it is advised to move custom code to a different file, e.g. `app.c`.
- Set the configuration value `SL_RAIL_UTIL_RAIL_POWER_MANAGER_INIT` to `0` for Door Lock Keypad, and to `1` for all the other applications.
- Add `clock_manager` component.
- Add the following configuration values:
 - name: `SL_CLOCK_MANAGER_WDOG0CLK_SOURCE`, value: `CMU_WDOG0CLKCTRL_CLKSEL_LFRCO`
- Add the following provides:

```
provides:
- name: zwave_sample_application
- name: <project_name> # e.g. zwave_soc_power_strip
```

NCP Zniffer and Zniffer PTI

NCP Zniffer and Zniffer PTI applications were merged into one application. Now, NCP Zniffer contains both applications' functionalities. It is not possible to upgrade these applications directly. Instead, please create a new project using the latest SDK and manually transfer any changes made in the previous project to the new one.

NCP Serial API Controller

SDK upgrades are not supported for this application. To migrate, please create a new project using the latest SDK and manually transfer any changes from the previous project to the new one.

Door Lock Keypad

app.c

- Add `#include "app_pm_transition_event.h"`
- Add these lines after `RadioConfig = zaf_get_radio_config();`

```
#ifdef SL_CATALOG_ZW_PM_TRANSITION_EVENT_PRESENT
// register callback from power manager transitions
ZW_PmTransitionEventInit();
#endif
```

- Remove `zw_cli_sleeping_util_prevent_sleeping_timeout(ZW_CLI_SLEEPING_WAKEUP_TIME_AFTER_RESET);`
- CLI baud rate was changed to 9600, which enables the device to receive CLI commands in EM2 sleep. There is no need to prevent sleeping for this application. To apply the new configuration on the baud rate, see the section `zwave_soc_door_lock_keypad.slc`.

app_cli_cc_user_credential.c

- Rename `u3c_user` struct type to `u3c_user_t`.
- Rename `u3c_credential` struct type to `u3c_credential_t`.
- Rename `u3c_event_data_validate` struct type to `u3c_event_data_validate_t`.
- Rename `u3c_event_data_learn_read_done` to `u3c_event_data_learn_read_done_t`.
- Rename `u3c_credential_metadata` to `u3c_credential_metadata_t`.
- Rename `u3c_credential_learn_event_data` to `u3c_event_data_learn_start_t`.
- When calling `CC_UserCredential_move_credential_and_report(...)`, remove `slot` argument.
- In function definition `u3c_move_credential_slot(...)`, use `CC_UserCredential_move_credential(...)` instead of `CC_UserCredential_move_credential_and_report(...)`.
 - According to the specification, this action is no longer supported. The lifeline group cannot be notified about this change.

app_credentials.c

- Rename `u3c_user` struct type to `u3c_user_t`
- Rename `u3c_credential` struct type to `u3c_credential_t`
- Rename `u3c_credential_learn_event_data` to `u3c_event_data_learn_start_t`

zwave_soc_door_lock_keypad.slc

- Do the steps described in General steps for all applications
- Additionally, add the following configuration to the `.slc` file:

```
- name: SL_IOSTREAM_EUSART_VCOM_BAUDRATE
  value: 9600 # In EM2 the eusart is working only with 9600 baud
- name: SL_IOSTREAM_EUSART_VCOM_ENABLE_HIGH_FREQUENCY # Disable high frequency mode for EUSART to work in EM2
  value: 0
- name: SL_CLOCK_MANAGER_WDOG0CLK_SOURCE
  value: CMU_WDOG0CLKCTRL_CLKSEL_LFRCO
- name: SL_CLOCK_MANAGER_EUSART0CLK_SOURCE
  value: CMU_EUSART0CLKCTRL_CLKSEL_LFRCO
```

Led Bulb

All steps are explained in section General steps for all applications.

Multilevel Sensor

app.c

- Add `#include "app_pm_transition_event.h"`.
- Add these lines after `RadioConfig = zaf_get_radio_config();` :

```
#ifdef SL_CATALOG_ZW_PM_TRANSITION_EVENT_PRESENT
// register callback from power manager transitions
ZW_PmTransitionEventInit();
#endif
```

app_btn_handler.c

- See General steps for all applications

app_sleep_callbacks.c

- Add the following include:

```
#include "sl_board_control_config.h"
#if SL_BOARD_ENABLE_SENSOR_RHT || SL_BOARD_ENABLE_SENSOR_IMU
#include "sl_board_control.h"
#endif
```

- Add at the end of void EMU_EM4PresleepHook(void) :

```
#if SL_BOARD_ENABLE_SENSOR_RHT
sl_board_disable_sensor(SL_BOARD_SENSOR_RHT);
#endif
#if SL_BOARD_ENABLE_SENSOR_IMU
sl_board_disable_sensor(SL_BOARD_SENSOR_IMU);
#endif
```

MultilevelSensor_interface_icm20689.c

See General steps for all applications

MultilevelSensor_interface_nosensor.c

See General steps for all applications

MultilevelSensor_interface_sensor.c

See General steps for all applications

Switch On Off

All steps are explained in section General steps for all applications.

Power Strip

All steps are explained in section General steps for all applications.

Sensor PIR

app.c

- Add #include "app_pm_transition_event.h".
- Add these lines after RadioConfig = zaf_get_radio_config(); :

```
#ifdef SL_CATALOG_ZW_PM_TRANSITION_EVENT_PRESENT
// register callback from power manager transitions
ZW_PmTransitionEventInit();
#endif
```

app_btn_handler.c

- See General steps for all applications

app_cli.c

- Add `#include <string.h>` to suppress compiler warning about the implicit declaration of the `strcmp` function.

SensorPIR_UserTask_DataAcquisition.c

- See General steps for all applications

Wall Controller

All steps are explained in section General steps for all applications.

7.23.0

How to migrate legacy HMLs to Silicon Labs based solution

General Information

In the 7.23 Z-Wave release, Silicon Labs changed the legacy button and LED handling to the Silicon Labs platform provided solution. This document describes how to migrate the legacy human-machine interfaces (from 7.19) to the Silicon Labs based solution. This document can be divided into two parts: one will describe the new components that have been introduced in the 7.23 release; and the other part will describe how to migrate a 7.22 project to the 7.23 release.

New components

SL Simple Button

The Simple Button Driver provides an implementation of the Generic Button API. This driver supports both active high and low buttons and configurable debouncing. This component is instantiable, meaning that several named instances can be created. For each instance a configuration is generated along with an initialization struct set according to this configuration. These instance defines and variables are available in a generated header file, `sl_simple_button_instances.h`. Selecting this component will also include the Simple Button Core component, which is the implementation of the Simple Button driver itself. [API Reference](#)

SL Simple LED

The Simple LED Driver provides an implementation of the Generic LED API. This driver supports controlling GPIO based on/off type LEDs. This component is instantiable, meaning that several named instances can be created. For each instance a configuration is generated along with an initialization struct set according to this configuration. These instance defines and variables are available in a generated header file, `sl_simple_led_instances.h`. Selecting this component will also include the Simple LED Core component, which is the implementation of the Simple LED driver itself. [API Reference \(5.0.2\)](#)

SL App Button Press

The Button Press component is used for handling the button state, which can be:

- Button is not pressed,
- Button is pressed for a short duration,
- Button is pressed for a medium duration,
- Button is pressed for a long duration,
- Button is pressed for a very long duration and
- Button has just been pressed down.

This component is built on SL Simple Button. Button press duration for various lengths is configurable and each end user can configure it according to their needs.

Differences in the project structure

In this document, the most complex project will be used as an example, which is the Z-Wave SoC Door Lock Keypad project. In the following table, the differences in the source files can be seen between the 7.22 and 7.23 projects.

7.22 Top Project Stucture	7.23 Top Project Structure
	app_btn_handler.c

7.22 Top Project Stucture	7.23 Top Project Stucture
	app_cc_battery_callbacks.c
	app_cc_door_lock_callbacks.c
app_cc_event_handlers	app_cc_event_handlers
app_cli.c	app_cli.c
app_cli.h	app_cli.h
app_credentials.h	app_credentials.h
app.c	app.c
events.h	events.h
main.c	main.c

The main differences are:

- app_btn_handler.c (API change)
- app_cc_battery_callbacks.c
- app_cc_door_lock_callbacks.c (API change)

Crucially, with this release, application related sources files like

platform\SiliconLabs\AppsHw\src\DoorLockKeyPad\DoorLockKeyPad_hw.c were removed. All application related logic was moved to the application project's directory. In this case, **DoorLockKeyPad_hw.c** was split into three separated sources: **app_btn_handler.c**, **app_cc_battery_callbacks.c** and **app_cc_door_lock_callbacks.c**.

Migrating the app_btn_handler.c file

In previous releases, button and LED configurations were spread out across multiple files. From the 7.23 release, Silicon labs switched to the standard SiSDK components to handle those peripherals, such as sl_simple_button and sl_simple_led. Because of those components, we can use Simplicity Studio's Pin Tool to remap a button or LED pin in the application. There is no need to do any additional configuration regarding these peripherals.

Buttons

Previously, button handling was implemented in platform\SiliconLabs\AppsHw\src\<application>\<application>_hw.c source files, in this case **DoorLockKeyPad_hw.c**. Now, this logic can be found in the application's directory, in **app_btn_handler.c**. A helper SiSDK component, called app_button_press, is also used to generate events on button pressing, such as short, medium or long press events. The duration of these events can be configured in **app_button_press_config.h** header file, in the config directory inside of the application project's directory. In case you had any custom logic regarding the button handling in platform\SiliconLabs\AppsHw\src\<application>\<application>_hw.c, in this example, **DoorLockKeyPad_hw.c**, please move them to the app_btn_handler.c file in the application project's directory.

In app_btn_handler.c, only the **app_button_press_btn_0_handler** or **app_button_press_btn_1_handler** functions need to be handled, which take the duration as an input parameter. Those callbacks are called automatically when a button event is fired.

LEDs

LED handling was simplified, which means LEDs can be manipulated via sl_simple_led API calls. They can also be configured using the Pin Tool. No additional configuration is required. In case you are using functions prefixed with **Board_** to manipulate LEDs or GPIOs, please use the sl_simple_led and em_gpio components' API from 7.23 release.

Migrating the app_cc_<command class>_callbacks.c files

There are new source files in every application, named in the pattern app_cc_<command class name>_callback.c. In the new structure, each of those sources contain callbacks for a specific command class. These callbacks can be different in every application, e.g. battery report callback, which can be different in a real life application compared to a sample application. From the 7.23 release, such callbacks are exposed on the application level. In this example, battery command class related

callbacks are in the application project's directory, named **app_cc_battery_callback.c**. If you had any custom implementation of command class specific callbacks in `platform\SiliconLabs\AppsHw\src\<application>\<application>_hw.c` files, in this case **DoorLockKeyPad_hw.c**, please move those implementations next to the application, into the related source file.

Firmware Update Images and Bootloader

Firmware Update Images and Bootloader

General Information

The guide assumes that the user is running Simplicity Studio V5.9.3.1 with the SiSDK 2024.12.0, which at the time of writing this document is the latest version available.

Note: Detailed information regarding bootloaders for the 800 series is available [here](#) and [here](#)

Before going further into this guide, it is important to know a few files and locations that are crucial in following the subsequent steps.

Default Locations

- **Default Simplicity Studio Installation Location:** C:\SiliconLabs\SimplicityStudio\v5
- **Default SDK Installation Location:** C:\Users\{Username}\SimplicityStudio\SDKs
- **Default Workspace Location:** C:\Users\{Username}\SimplicityStudio\v5_workspace
- **SiSDK demo location:** {Simplicity Studio Installation Location}\offline\com.silabs.sdk.stack.sisdk_2024.12.0
- **Sample encryption keys:** - sample_sign.key , sample_sign.key.pub , sample_sign.key-tokens.txt , sample_encrypt.key
- **Sample keys location:** {SDK Installation Location}\gecko_sdk\protocol\z-wave\platform\SiliconLabs\PAL\BootLoader\sample-keys
- **Bootloader location:** {SiSDK demo location}\platform\bootloader\
- **Application location:** {SiSDK demo location}\protocol\z-wave\
- **Commander utility location:** {Simplicity Studio Installation Location}\developer\adapter_packs\commander
- **Sample project location:** {Default Workspace Location}\zwave_soc_switch_on_off\

>**Note:** The above paths are customizable by the user and as such must be adapted if they are different from the default paths given above.

General information

The purpose of this section is to describe how to generate and manage firmware update images. The SDK provides two bootloaders for a given board type - OTA and OTW. The OTA bootloader is needed for all Z-Wave based devices that implement firmware updates; the OTW bootloader is for devices that update firmware using the serial port from another host controller. The OTA bootloader is triggered when an image has been transferred over the air using the FIRMWARE_UPDATE command class. The transferred image must be an image in Gecko Bootloader (GBL) format. The bootloaders provided in the SDK require the GBL image to be signed.

Three steps are needed for performing an OTA update:

1. The OTA bootloader must be flashed.
2. The signing keys and optionally an encryption key must be flashed.
3. A signed image must be transferred using the firmware update command class.

Further information about bootloaders can be found [here](#).

Generate GBL files (manual steps)

To generate the GBL files needed for the OTA/OTW update, a signing keypair must first be created. It is the intention that a vendor will keep the signing keypair for the lifetime of the product. These keys are used to sign all the firmware versions for the whole lifetime of the product. An encryption key must also be created, this key is intended for encrypting the GBL file. Encryption makes it harder for a bootlegger to copy the product.

The signing keys can be created using the `Simplicity Commander` 's command line interface:

```
commander.exe gbl keygen --type ecc-p256 -o vendor_sign.key
```

This step will create 3 files:

vendor_sign.key - This is the private key and must be kept safe by the manufacturer. **vendor_sign.pub.key** - This is the public key **vendor_sign.key-tokens.txt** - This is the public key in another format which can be programmed into the device at manufacturing using simplicity commander. A vendor may choose to have a keypair like this for all his products, one for each product type.

An encryption key can be generated as follows:

```
commander.exe gbl keygen --type aes-ccm -o vendor_encrypt.key
```

Once the two keys have been obtained, a GBL may be produced as follows:

```
commander.exe gbl create appname.gbl --app appname.hex --sign vendor_sign.key --encrypt vendor_encrypt.key --compress lzma
```

This should be done each time a new firmware is produced.

Firmware Update flow (manual)

1. Download the bootloader image files. This is done by running the sample demo in Simplicity Studio. This will download the bootloader images to disk.
 - OTA bootloader images: `bootloader-storage-internal-single-zwave-ota-<board_number>.s37`
 - OTW bootloader images: `bootloader-uart-xmodem-zwave-otw-<board_number>.s37`
2. Create an example project using the same project above as a template.
3. Build the project in SimplicityStudio and generate the hex files.
4. Erase device:

```
commander.exe device masserase -s <board_jlink_serial>
```

5. Reset device:

```
commander.exe device reset -s <board_jlink_serial>
```

6. Flash the appropriate OTA bootloader image:

```
commander.exe flash {Bootloader location}\bootloader-storage-internal-single-zwave-ota-<board_number>.s37 -s <board_jlink_serial>
```

7. Flash initial device firmware built in step 3:

```
commander.exe flash "{sample project location}\zwave_soc_switch_on_off.hex" --address 0x0 -s <board_jlink_serial>
```

8. Flash the encryption keys:

```
commander.exe flash --tokengroup znet --tokenfile sample_encrypt.key --tokenfile sample_sign.keytokens.txt -s <board_jlink_serial>
```

9. Reset device:

```
commander.exe device reset -s <board_jlink_serial>
```

10. Connect a controller or a device running a controller firmware to the PC and start the PC controller application.
11. Include the node into the network and make sure the device is visible.
12. In the PC controller application initiate the OTA update using the OTA gbl file mentioned in the prerequisites.

Firmware update flow (Simplicity Studio)

1. Create a new `Solution Examples` project using the sample project as a template, such as `ZWave_SoC_SwitchOnOff_Solution`. This will create a new workspace with the bootloader and the Z-Wave application.
2. Open the `<solution_name>.slpb` (SL Postbuild Profile) file. This will open a GUI, where the encryption and signing keys can be added.
3. When the `Solution Examples` project is built, the GBL files are generated in the `artifacts` folder with the merged binary of the bootloader and the Z-Wave application.
4. Follow 7-12 from the manual steps above, but use the generated GBL file from the `artifacts` folder.

> **Note:** The automatic generation of GBL files also works with `Example Projects`, but in this case, the bootloader won't be merged with the Z-Wave application.

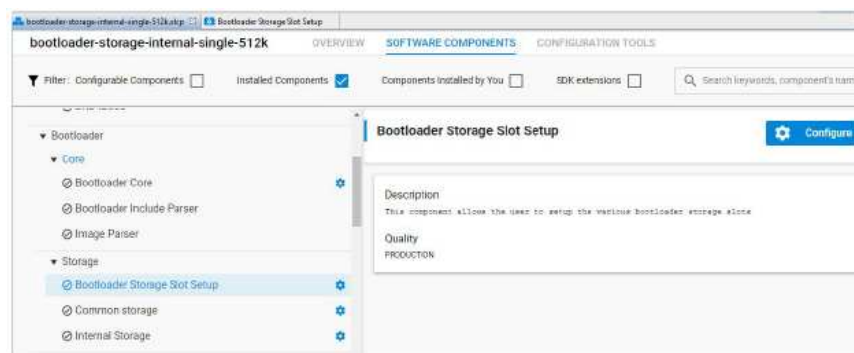
v255 generation

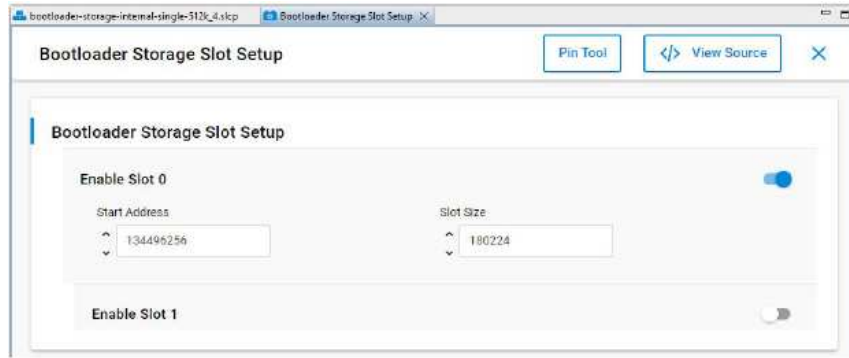
The v255 files are the same as the released firmwares of the given application. The only difference is that the firmware version is set to 255.0.0 to make sure that the image version is greater than the current firmware version on the device, to make the OTA update possible. This file is available as a `Demo` (check the {Application location} and generate the GBL manually) or it can be generated from a `Solution Example` or `Project Example` in Simplicity Studio. The following steps are required to generate the v255 file:

1. Open `zw_version_config.h` in the GUI or in the text editor.
2. Set `USE_USER_APP_VERSION` to `1`.
3. Set `ZAF_VERSION_MAJOR` to `255`.
4. Set `ZAF_VERSION_MINOR` to `0`.
5. Set `ZAF_VERSION_PATCH` to `0`.

Bootloader configuration

The bootloader resides at the start address `0x08000000` of the main flash and a fixed space of `24KB` is reserved for this. Z-Wave applications will start from address `0x08006000`. The bootloader must be flashed first before the Z-Wave sample application is flashed. It is also possible to combine the bootloader and the Z-Wave application into a single image. *One can use the pre-built bootloader images available in Simplicity Studio or build a bootloader image by themselves using the bootloader sample applications in Simplicity Studio. * When building the bootloader, the OTA image storage information must be configured according to subsequent images. Importantly, **this storage slot (slot0), start address and size must not be changed.**





In Simplicity Studio, the `Solution Examples` provide workspaces for the bootloader and the Z-Wave application. In the workspaces' postbuild profile the gbl files are generated and the bootloader and the Z-Wave application are combined into a single image (check `artifacts` folder).

Linker Script for the 800 Series

The Z-Wave sample applications that are available in Simplicity Studio contain linker scripts that have been tuned to accommodate the OTA image-related configuration as well. It is recommended not to modify this linker script when developing applications.

Bootloader Compression for 800 Series

The bootloader compression type used for OTA is **Izma** compression. The corresponding component name to be selected in the studio is `bootloader_compression_Izma`.

Application Upgrade Version

The studio component `bootloader_app_upgrade_version` has to be selected for checking the application version during upgrades.

Erase and Read Manufacturing Tokens for the 800 Series

There is a dedicated space in the flash memory where the Manufacturing Tokens data can be stored. This area can be written once during firmware running. To save the new region, the flash must be erased before.

Read token frequency:

```
commander tokendump --tokengroup znet --token MFG_ZWAVE_COUNTRY_FREQ
```

Write token frequency:

```
commander flash --tokengroup znet --token MFG_ZWAVE_COUNTRY_FREQ 0xFF
```

0xFF means this area is erased.

How to implement a new command class

How to implement a new command class

General informations

An essential feature of the ZAF is the communication through Command Classes. For this purpose, each of the Command Classes has a C module where incoming commands are handled, and outgoing commands are transmitted.

Command classes can have a controlling role and a supporting role. An application typically uses one of the roles. For instance, an LED bulb supports Color Switch CC, but it doesn't control Color Switch CC. The controlling role of Color Switch CC would be suited for a Z-Wave gateway as the gateway would be controlling the color of the LED bulb based on user input in a corresponding mobile app.

Based on the above, best practice would be to split the controlling implementation and the supporting implementation in two different source files like it's done for CC Basic.

Implementation of supporting part

Most command classes require two basic things for the supporting implementation:

1. Define a command handler that parses incoming frames
2. Use `REGISTER_CC_V5()` to register the command handler and the CC version.

Please see [ZAF_types.h](#) for descriptions `REGISTER_CC_V5()` and the corresponding type definition of the command handler.

Association Group Information

<https://sdomembers.z-wavealliance.org/wg/AWG/document/120> defines a list of command class / command pairs that MUST be listed for the Lifeline association group. The Z-Wave Controller will ask for the list using the Association Group Command List Get command.

ZAF will propagate this list automatically, but it requires each command class in the above registry to register the relevant commands. Use `REGISTER_CC_V5()` for this.

The following shows an example of how it's done for CC Door Lock. CC Door Lock must register two different commands which is why the function returns 2. Other command classes register only one command and must then return 1.

```
static uint8_t lifeline_reporting(ccc_pair_t* p_ccc_pair)
{
    p_ccc_pair->cmdClass = COMMAND_CLASS_DOOR_LOCK;
    p_ccc_pair->cmd      = DOOR_LOCK_OPERATION_REPORT;
    p_ccc_pair++;
    p_ccc_pair->cmdClass = COMMAND_CLASS_DOOR_LOCK;
    p_ccc_pair->cmd      = DOOR_LOCK_CONFIGURATION_REPORT;
    return 2;
}

REGISTER_CC_V5(..., lifeline_reporting, ...);
```

Basic Mapping

The Device Type V2 specification dictates how a certain device type (DT) must map CC Basic to other command classes.

For instance, the Lock DT must map CC Basic to CC Door Lock. This mapping is done automatically by ZAF because CC Door Lock has registered mapping functions.

Basic Set Mapping

Basic Set mapping is done by registering a Basic Set mapping function. Please see [basic_set_mapper_t](#) for a description of the function parameters.

Below is an example from CC Door Lock:

```
static void basic_set_mapper(ZW_APPLICATION_TX_BUFFER* pFrame)
{
    pFrame->ZW_DoorLockOperationSetV4Frame.cmdClass = COMMAND_CLASS_DOOR_LOCK;
    pFrame->ZW_DoorLockOperationSetV4Frame.cmd    = DOOR_LOCK_OPERATION_SET;
}

REGISTER_CC_V5(..., basic_set_mapper, ...);
```

Basic Get Mapping

Basic Get mapping is done by registering a Basic Get mapping function. Please see [basic_get_mapper_t](#) for a description of the function parameters.

Below is an example from CC Multilevel Switch:

```
static void basic_get_mapper(uint8_t endpoint, uint8_t * p_current_value, uint8_t * p_target_value, uint8_t * p_duration)
{
    *p_current_value = cc_multilevel_switch_get_current_value_handler(endpoint);
    *p_target_value  = cc_multilevel_switch_get_target_value_handler(endpoint);
    *p_duration      = cc_multilevel_switch_get_duration_handler(endpoint);
}

REGISTER_CC_V5(..., basic_get_mapper, ...);
```

Parsing incoming frames

Each CC implementation defines a handler, e.g. `CC_MultilevelSwitch_handler()`. This function extracts the received frame for a given Command Class. The function must be registered using a `REGISTER_CC_V5` macro. Normally, the frame is carrying a `â€œSetâ€œ` or `â€œGetâ€œ` Command that results in a function call for reading or writing data. All commands are handled by the command class itself.

Example:

```
static received_frame_status_t CC_MultilevelSwitch_handler(
    cc_handler_input_t * input,
    cc_handler_output_t * output)
{
    switch (input->frame->ZW_Common.cmd)
    {
        case SWITCH_MULTILEVEL_GET:
            // Parse SWITCH_MULTILEVEL_GET
            break;
        default:
            // Unknown command
            return RECEIVED_FRAME_STATUS_NO_SUPPORT;
    }
    return RECEIVED_FRAME_STATUS_SUCCESS;
}

REGISTER_CC_V5(COMMAND_CLASS_SWITCH_MULTILEVEL, SWITCH_MULTILEVEL_VERSION_V4, CC_MultilevelSwitch_handler, ...);
```

Implementation of controlling part

A controlling implementation of a CC is rather simple as the functionality is limited to transmission of one or more specific frames.

One example is `CC_Basic_Set_tx()` that can be found in `CC_BasicController.c`:

```
zaf_job_status_t CC_Basic_Set_tx(
    agi_profile_t const * const p_profile,
    const uint8_t source_endpoint,
    const uint8_t value,
    const bool use_supervision,
    ZAF_TX_Callback_t callback)
{
    ccc_pair_t ccc_pair = {COMMAND_CLASS_BASIC, BASIC_SET};

    return cc_engine_multicast_request(
        p_profile,
        source_endpoint,
        &ccc_pair,
        (uint8_t *)&value,
        sizeof(value),
        use_supervision,
        callback);
}
```

A controlling device, e.g., a wall switch, may want to send a Basic Set command. The implementation of Basic Set transmission is shown above. The `value` parameter is specific for the Basic Set command. The remaining four parameters are recommended as generic parameters: `p_profile`, `source_endpoint`, `use_supervision` and `callback`. These parameters are detailed below.

Another slightly more complex example of such a function can be found in `CC_MultilevelSwitch_Control.c`.

Some command classes might not offer the profile and the source endpoint, but instead rely on an index or a pointer to the configuration that contains the AGI profile and the source endpoint. One example of such an API is `CC_Notification_TriggerAndTransmit()`.

`p_profile` (AGI profile)

The `p_profile` is required together with `source_endpoint` for ZAF to be able to look up the right association group. The profile must match one of those defined in the CC AGI configuration. Alternatively, if NULL is passed as the profile, the ZAF will default to the Lifeline profile.

`source_endpoint`

If the application is configured with endpoints, `source_endpoint` must be set to the endpoint that triggers the command. An example could be a device with two temperature sensors represented by endpoint 1 and 2. This device could transmit a Basic Set if the temperature drops under a certain threshold. In that case, `CC_Basic_Set_tx()` would need to be invoked with the endpoint that represents the relevant temperature sensor. If an application does not configure endpoints, `source_endpoint` must be set to zero.

`use_supervision`

This parameter offers the application developer the option to use Supervision for a certain command. Supervision might be important in the case of a window sensor in an alarm system to make sure that a Notification Report, signaling that a window was opened, is received and decrypted by the destination node. An example where Supervision might be of less relevance could be a wall switch sending a Basic Set or a Binary Switch set. In this scenario, an end user would probably just press the button again if a frame was lost and the light didn't turn on in the first attempt.

`callback`

In case the application logic requires a callback whenever a transmission succeeded or failed, the `callback` parameter can be used. This parameter is not required, but offering it in the API adds some flexibility to the application developer.

File structure

Each Command Class should have following folders:

- `inc` for public headers
- `src` for private headers and source files.

Version

The command class version must be set using [REGISTER_CC_V5\(\)](#).

Example:

```
REGISTER_CC_V5(COMMAND_CLASS_DOOR_LOCK, DOOR_LOCK_VERSION_V4, ...);
```

What is a migration code block and how to use them

What is a migration code block and how to use them

Migration code blocks are introduced when there is an incompatible code change between two major releases of the GSDK. This happens when there is a change in the file structure. This change necessitates a migration block. Then the migration code blocks transform the relevant data from the older format to the newer format. Such changes are introduced only on the minor GSDK releases and never on any patch changes. These migration code blocks are relevant only when the user is on an older GSDK version and would like to port or migrate an existing code base to the newer GSDK version.

Migration Code

This is the code that actually does the transformation or migration from the older file version to the newer file version. This is usually implemented in a file by name `<module_name>_nvm.c`. This file has a function by name `<module_name>_migrate` or `<module_name>_Migrate`. In the migrate function, the code that does the actual transformation is wrapped in a `#ifdef ... #endif` guard. The define used is of the form `ZW_MIGRATION_TO_<MAJOR>_<MINOR>` where the MAJOR and MINOR are the GSDK versions.

Sample migration function

```
cc_door_lock_migrate(void)
{
    #ifdef ZW_MIGRATION_TO_7_19
    ...
    ...
    #endif
}
```

Usage

As mentioned in the in the previous section, the migration code blocks are installed by default. In the Studio UI, these are usually found under the Software Components > Z-Wave > Migration section in the project configuration. Each of the component is the shown is the migration code released in a specific GSDK version. If a user is on a later GSDK version, then an older component is redundant and can be safely uninstalled. If the user is on the latest GSDK version, all the older components can be uninstalled to save some flash and ram. If the user is on an older GSDK version and would like to build the project with the latest GSDK, then the user should select all those components that are greater than the GSDK version at the time of the project creation. This ensures that any incompatible file structures are automatically transformed to the latest GSDK version.

True Status Module

True Status Module

Function for registering state change events and triggering the report to be sent to lifeline group members after a predefined delay. The delay is currently defined to 250 ms. The delay is a means to prevent network collisions and to avoid generation of redundant reports from rapid state changes. The function also takes care of sending the report to the relevant lifeline group members only i.e., it will not send the report to a lifeline destination that issued the command causing the state change. The current implementation can hold up to 3 different state change requests in a queue awaiting the predefined delay to expire. Additional requests during this period will be discarded.

General Information

The True Status module implements the requirements for Lifeline Reports as specified in document "ZWave Plus v2 Device Type Specification".

The main components of the True Status module are described in the following sub-sections.

True Status Engine

The True Status Engine component is located in the `ZAF/ApplicationUtilities/TrueStatusEngine` folder and consists of the following source files:

1. `ZAF_TSE.c`
2. [ZAF_TSE.h](#)

TSE implements the functionality for registering and handling the state change events that a node wants to report to its lifeline association group members. A state change can be triggered by a command from a remote node, or by a local change (e.g., a button press).

Public functions

[ZAF_TSE_Init](#) Returns

- bool

[ZAF_TSE_Trigger](#)

True Status Callback Functions

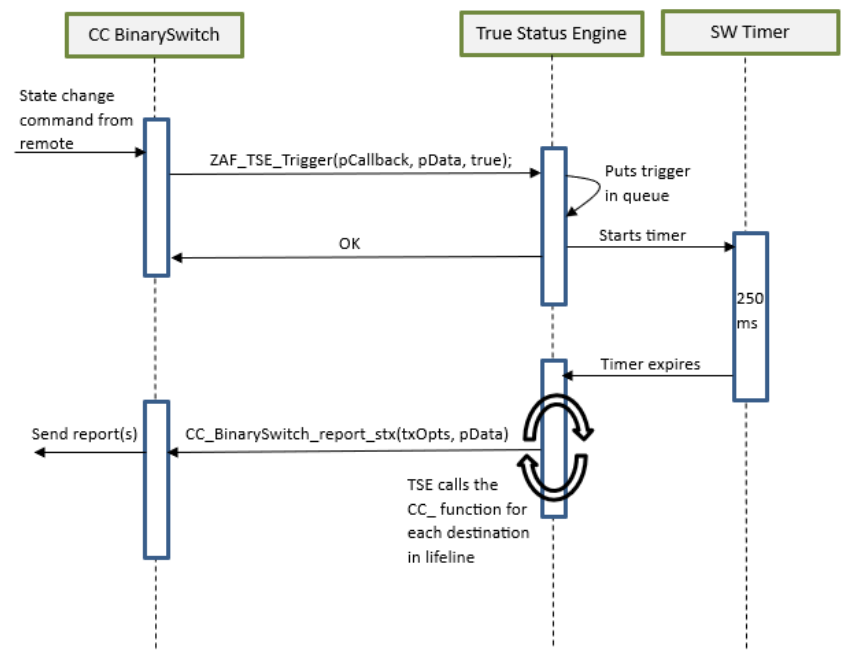
The True Status Callback Functions consist of several functions that implement the functionality for sending the actual state change reports to the lifeline association group members. These functions are implemented in each of the relevant command class modules. The True Status Callback Function is one of the arguments passed to the `ZAF_TSE_Trigger()` function (described in the previous section), and it will be executed by the True Status Engine for sending the state change reports to each of the relevant members of the lifeline group, one at a time. An example of a True Status Callback Function implementation can be found in the Command Class `BinarySwitch`:

True Status Sequence Flows

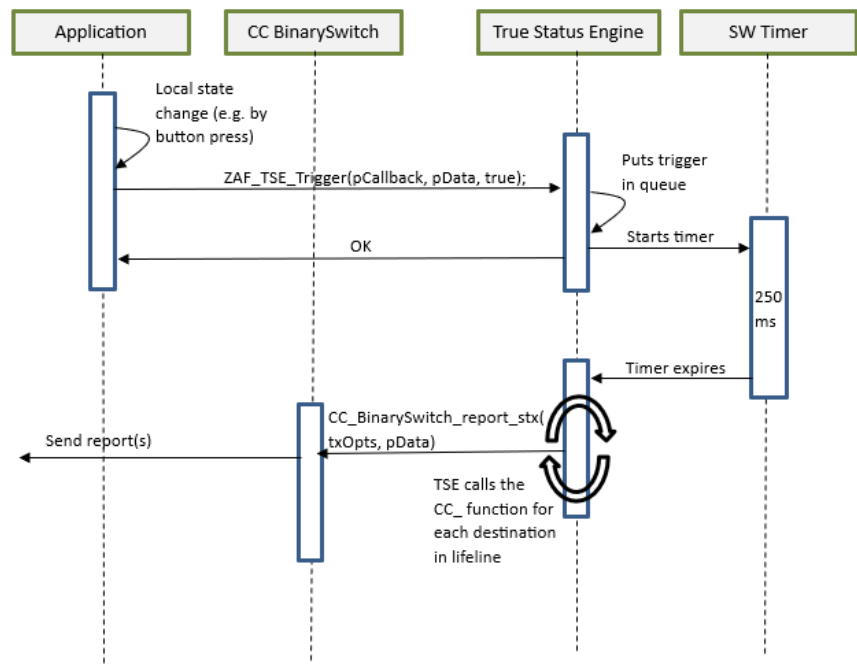
The following diagrams show the function call flows for a `BinarySwitch` example for the two use cases.

1. State change triggered by a command from a remote node
2. State change triggered by a local change (e.g., a button press)

Use Case 1 - State Change Triggered by a Command from a Remote Note



Use Case 2 - State Change Triggered by a Local Change



When and how to create a Multi Channel application

When and how to create a Multi Channel application

Creating a Multi Channel application might be relevant in several use cases.

Some use cases might be:

- A power strip where each outlet can be controlled individually
- Two or more identical sensors each reporting their value, for instance temperature. Multi Channel is a requirement when the sensors are of the same type.

What to configure?

The following describes the minimum configuration that is required.

The Power Strip application shows an example of how the following can be configured and adds application specific configuration on top as in Binary Switch CC, Multilevel Switch CC and Notification CC.

Multi Channel Command Class

The Multi Channel Command Class component must be installed in Simplicity Studio and comes with some configuration parameters.

Example from Power Strip configuration:

```
zw_cc_multi_channel:
  endpoints:
    - endpoint: 1
      generic_type: GENERIC_TYPE_SWITCH_BINARY
      specific_type: SPECIFIC_TYPE_NOT_USED
    - endpoint: 2
      generic_type: GENERIC_TYPE_SWITCH_MULTILEVEL
      specific_type: SPECIFIC_TYPE_NOT_USED
```

Z-Wave Plus Info Command Class

This configuration defines the icon types for each endpoint so that a controller would know which icons to display in a mobile app.

Example from Power Strip configuration:

```
zw_cc_zwaveplusinfo:
  endpoints:
    - endpoint: 1
      installer_icon_type: ICON_TYPE_GENERIC_ON_OFF_POWER_SWITCH
      user_icon_type: ICON_TYPE_GENERIC_ON_OFF_POWER_SWITCH
    - endpoint: 2
      installer_icon_type: ICON_TYPE_GENERIC_LIGHT_DIMMER_SWITCH
      user_icon_type: ICON_TYPE_GENERIC_LIGHT_DIMMER_SWITCH
```

Association Group Information Command Class

This configuration defines what the application will report unsolicited.

Endpoint specific configuration is not mandatory as all reporting can be done via Lifeline, but configuring this for endpoints opens up for a more flexible installation at the end user.

In case of Power Strip, the application can transmit a Notification Report from each endpoint if any nodes are associated in those association groups.

It's up to the application developer to decide on the level of flexibility.

Example from Power Strip configuration:

```
zw_cc_agi:
endpoints:
- endpoint: 1
  groups:
    - name: Alarm EP 1
      profile_msb: ASSOCIATION_GROUP_INFO_REPORT_AGLPROFILE_NOTIFICATION
      profile_lsb: NOTIFICATION_REPORT_POWER_MANAGEMENT_V4
      commands:
        - command_class: COMMAND_CLASS_NOTIFICATION_V3
          command: NOTIFICATION_REPORT_V3
- endpoint: 2
  groups:
    - name: Alarm EP 2
      profile_msb: ASSOCIATION_GROUP_INFO_REPORT_AGLPROFILE_NOTIFICATION
      profile_lsb: NOTIFICATION_REPORT_POWER_MANAGEMENT_V4
      commands:
        - command_class: COMMAND_CLASS_NOTIFICATION_V3
          command: NOTIFICATION_REPORT_V3
```

CLI Manipulator Board Application

CLI Manipulator Board Application

The CLI Manipulator Board application allows to emulate all functionality of the SiliconLabs's BRD8029 expansion board including 4 LEDs, 4 push buttons and a slider switch. Button push/release/press, slider on/off events and led state queries are performed through CLI commands. The GPIO pins are mapped according to the Simple Button and LED SiSDK components, except for the slider which is defined in `cli_manipulator.h`.

You can use the command line interface over the VCOM serial port. The app provides four commands:

- `ext_led`: Reads a specified LED's state given in an integer argument from 0 to 3.
- `ext_button`: Simulates a button push, release or press. The first input identifies the button as an integer from 0 to 3 while the second argument specifies a 'push', 'release' or 'press' (push and release) action.
- `ext_slider`: Simulates switching the slider to the on or the off position. The only string input argument is 'on' or 'off' respectively.

Requirements

Silicon Labs board with 16-Pin expansion header.

Connect the pins of the two boards marked with an `o` in the figure below:

```
.....o.. pin 20
pin 1 .oooooooo..
```

See also the SQA intranet page on this topic: <https://confluence.silabs.com/x/mCYKEw>

How to use the .hts scripts

How to use the .hts scripts:

Setup

1. install the hterm PC application from [here](#)
2. Right click on the Sequence overview window on the left, and click load file
3. select the railtest.hts file

Available commands:

- rx 0
-

- setzwavemode 1 3
-

- setzwaveregion 11
 - setzwaveregion 1
-

- setchannel 0
 - setchannel 1
 - setchannel 2
 - setchannel 3
-

- setpower 155 raw
-

- settxtone 1
- settxtone 0

Applications

Applications

This folder contains test applications for the Open Z-Wave. Each application is placed in a subfolder.

Usage on Linux platform

Command line arguments

Argument	Description
-debug	Don't fork process (useful for debug purposes)
-storage <path>	Path to storage (application and stack filesystem, manufacturing tokens and retention registers)
-port <port>	Port for TCP server mode of UART module (default 4901)
-region <region>	Simulated radio region. One of [REGION_EU, REGION_US, REGION_ANZ, REGION_HK, REGION_IN, REGION_IL, REGION_RU, REGION_CN, REGION_US_LR, REGION_JP, REGION_KR, REGION_DEFAULT]
-pty	Use PTY mode of UART module
-id <ID>	Use routed ZNE with ID
-zne-port <PORT>	Port for routed ZNE
-fifo	Use FIFO based communication
-tmp-path <PATH>	Path to a temporary directory unique for a node. Used to store FIFO and Unix Domain Socket files.

Additional arguments for Emulated Tick mode

Argument	Description
-tick-server <PATH>	Tick Server socket path

Application control

Applications use single character commands read from standard input.

Character	Description
h	Display usage info
d	Print DSK
n	Print Node ID
l	Start Learn Mode
L	Stop Learn Mode
f	Factory reset
p	Print PTY
r	Restart application
q	Quit application

Usage on other platforms

Please refer to documentation of platform specific implementation in `platform/` directory.

Serial API

Serial API

The Serial Applications Programming Interface (Serial API) allows a host to communicate with a Z-Wave chip. The host may be PC or a less powerful embedded host CPU, e.g., in a remote control or in a gateway device. Depending on the chip family, the Serial API may be accessed via RS-232 or USB physical interfaces.

There are 2 available versions of SerialAPI:

- SerialAPI Controller
- SerialAPI End Device

The following host-based sample applications are available in the SDK:

- Z/IP Gateway – Gateway application using Serial API features of the bridge controller API
- PC Controller – Demonstrates Serial API features of the bridge controller API

The host-based sample applications are described in the respective SDK overview documents.

The Serial API leverages the Z-Wave Protocol API. The Serial API introduces additional messages related to inter-host communications.

Known Issues

OTW Update

The following update paths for Serial API Controller are not possible using the pre-built binaries because the defines `ZAF_CONFIG_GENERIC_TYPE` and `ZAF_CONFIG_SPECIFIC_TYPE` differ across these versions:

- 7.19 or lower to 7.20
- 7.20 to 7.21 or higher

However, it is possible to create a custom firmware that is compatible with earlier versions of the application by building the application manually after editing `application_properties_config.h`.

If you want to update a device running the pre-built 7.19 or lower version firmware to 7.20, change both `PRODUCT_ID_GENERIC_TYPE` and `PRODUCT_ID_SPECIFIC_TYPE` to `0`.

If you want to update a device running the pre-built 7.20 version firmware to 7.21 or higher, remove the following conditional block from the file:

```
#if (ZAF_CONFIG_GENERIC_TYPE == GENERIC_TYPE_STATIC_CONTROLLER && \
    ZAF_CONFIG_SPECIFIC_TYPE == SPECIFIC_TYPE_PC_CONTROLLER)
(...)\n#endif
```

Changing these values will only affect the Product ID, stored in the struct `sLapp_properties`. The application will still report the values specified in the defines `ZAF_CONFIG_GENERIC_TYPE` and `ZAF_CONFIG_SPECIFIC_TYPE` in its Node Information Frame.\ Note that these values can be changed on the fly using the Serial API command `SERIAL_API_APPL_NODE_INFORMATION`.

Zniffer

Zniffer

The Zniffer is a development tool for capturing Z-Wave RF communication and sending the frames to a host application.

Z-Wave Zniffer is a PC side host application which communicates with the Zniffer via a USB serial connection and presents the frames in a graphical user interface. This enables Z-Wave developers to analyze all RF communication between Z-Wave nodes.

The host application can connect to the Zniffer through either:

- a virtual COM port over a USB serial connection (or through port 4901 when an Ethernet connection is available)
- or via the Packet Trace Interface (PTI) over TCP/IP on port 4905.

Usage on Linux platform - zwave_soc_door_lock_keypad

Usage on Linux platform - zwave_soc_door_lock_keypad

Application uses single character commands read from standard input.

Character	Description
1	Send battery report
2	Activate/deactivate outside door handle with User Code
3	Activate/deactivate outside door handle with User Credential
4	Start Credential Learn process for UUID=1 Type=PIN Slot=2
5	Enter hard-coded data (1667) for the Credential Learn process

Door Lock

Door Lock

The Z-Wave certified Door Lock application shows a lock implementation. It supports user codes to open a door and thereby eliminate the need for traditional keys. It is possible to both lock and unlock the door remotely through the Z-Wave protocol.

The Door Lock application is based on:

Role Type	Listening Sleeping End device (LSS / FLiRS)
Supporting Device Type	Actuator
Device Type	Lock
Generic Type	Entry Control
Specific Type	Door Lock
Requested security keys	S0 and S2_ACCESS The Door Lock features depend on the security level

The application will only work when securely added to a network by a controller supporting security. The controller **MUST** support security S2_Access_Control or S0 in order to be able to control the door lock.

Not implemented Door lock functionality:

- Timed Operation mode
- Door Lock condition
- Inside Door Handle State
- Functionality handling Lock timeout
- Target mode
- Auto-Relock, Hold And Release, Block to Block, and Twist Assist

Supported Command Classes

The Door Lock application implements mandatory and some optional command classes. The table below lists the supported Command Classes, their version, and their required Security class, if any.

Command Class	Version	Required Security Class
Association	2	S0 or Access Control
Association Group Info	3	S0 or Access Control
Basic	2	S0 or Access Control
Battery	1	S0 or Access Control
Device Reset Locally	1	S0 or Access Control
Door Lock	4	S0 or Access Control
Firmware Update Meta Data	5	S0 or Access Control
Indicator	3	S0 or Access Control
Manufacturer Specific	2	S0 or Access Control
Multi-Channel Association	3	S0 or Access Control
Powerlevel	1	S0 or Access Control

Command Class	Version	Required Security Class
Security 0	1	None
Security 2	1	None
Supervision	1	None
Transport Service	2	None
User Code	1	S0 or Access Control
User Credential*	1	S0 or Access Control
Version	3	S0 or Access Control
Z-Wave Plus Info	2	None

*: The User Credential Command Class is only available on specific boards. See the section User Credential Command Class.

User Credential Command Class

The User Credential Command Class is enabled by default on the following boards:

- BRD2705A
- BRD4400B
- BRD4400C
- BRD4401B
- BRD4401C

Support for this command class can be disabled if the User Code Command Class is sufficient, by disabling the **User Credential Command Class** component.

To enable this command class on other boards, due to flash memory constraints, it is recommended to first disable the **Z-Wave CLI Common** component before enabling the **User Credential Command Class** component.

The default credentials are created when the inclusion process finishes.

- If the User Code Command Class is enabled:
 - User Code 1 is created with a value of 3494.
- If the User Credential Command Class is enabled:
 - A Programming User with the unique identifier 1, called "Admin" is created.
 - A PIN Code associated with user 1 is created with a value of 3494.

To operate the door bolt with the default credentials, the end device must first be included in a Z-Wave network.

Basic Command Class mapping

The Basic Command Class is mapped according to the table below.

Basic Command	Mapped Command
Basic Set (Value)	Door Lock Operation Set (Door Lock Mode)
Basic Report (Current Value = 0x00)	Door Lock Operation Report (Door Lock Mode = 0x00)
Basic Report (Current Value = 0xFF)	Door Lock Operation Report (Door Lock Mode = 0xFF)

Association Groups

The table below shows the available association groups, including supported command classes for Z-Wave and Z-Wave Long Range respectively.

ID	Name	Node Count	Description
1	Lifeline	X	Supports the following command classes: • Device Reset Locally: triggered upon reset. • Battery: Triggered upon low battery. • Door Lock: Triggered upon a change in door lock configuration. • Door Lock: Triggered upon a change in door lock operation. • Indicator Report: Triggered when LED0 changes state. • User Code: Triggered when a user code record is modified. • User Credential*: Triggered when a Duress User enters a Credential.

X: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 1.\ *: The User Credential Command Class is only available on specific boards. See the section User Credential Command Class.

Usage of Buttons and LED Status

We are differentiating four different types of button presses. The following types are the same for the BTN0 and BTN1 on the WSTK board. The duration values can be configured under the config directory in app_button_press_config.h file in each generated application/project.

Please note external wakeup is not supported on button 1 in case of brd2603a and brd2603b.

Press Type	Duration
Short Press	0 - 400 ms
Medium Press	401 - 1500 ms
Long Press	1501 - 5000 ms
Very Long Press	Every press longer than Long Press

Button	Action	Description
RST	Press	Resets the firmware of an application (like losing power). All volatile memory will be cleared.
BTN0*	Short Press	Simulates entering a PIN Code on a keypad via the User Credential API. The entered PIN Code is hard-coded with the value of 3494. A valid Credential entry (i.e., button press) toggles the Door Lock Mode between Secured and Unsecured, with a delay of 2 seconds, to simulate timed hardware operation. If Credential data for User UID 1, Slot 1 is changed to something other than the default PIN Code, the Door Lock Mode can no longer be toggled by pressing this button.
Medium Press	Sends Battery Report	Enter "learn mode" (sending node info frame) to add/remove the device. Removing the device from a network will reset it.
Long Press	Changes the outside door handle mode to active if it was inactive and vice versa.	
BTN1	Short Press	
Very Long Press	Perform a reset to factory default operation of the device, and a Device Reset Locally Notification Command is sent via Lifeline.	

LED	Functionality	Description
LED0	Indicator:	Blinks with 1 Hz when learn mode is active. Used for Indicator Command Class.
LED1	Bolt:	• Led off -> bolt unlocked • Led on -> bolt locked

*: There is a limitation of BTN0 functionality on BRD2705A. On this board, BTN0 is routed to PC00 pin, which is incapable of triggering an interrupt from EM2. Since Door Lock Keypad mostly operates in EM2, sometimes, this could result in an incorrect behaviour, i.e. a short press on BTN0 may trigger Battery Report, and a medium press may enter the PIN code.

Firmware Update

This section will describe backward compatibility when upgrading the Door Lock application from one SDK to a newer version.

CLI Support

In case CLI support is needed, please install `zw_cli_common` component to the project. Please note that the `zw_cli_common` component will modify the power consumption for every application.

The default baud rate is 9600, which allows the underlying EUSART module to work in EM2. On some development kits (e.g. BRD2603A and BRD2705A), the following command should be applied in Simplicity Studio via the Admin console to adjust the baudrate: `serial vcom config speed 9600`.

Available CLI commands

Command	Arguments	Description
help	-	Printing all supported CLI commands and their description.
set_learn_mode	-	Toggling the learn mode functionality. In case the learn mode started, but inclusion does not happen, learn mode will automatically stop after about 30 seconds
factory_reset	-	Executing factory reset
get_dsk	-	Printing out the generated DSK of the device
get_region	-	Printing out the set region of the application
battery_report	-	Sending a battery report
enter_user_code	Your current user code. It must be between 4 and 10 digits.	Entering a minimum 4 digit long user code
set_new_user_code	Your new user code. It must be between 4 and 10 digits.	Setting a new user code wich's lenght must be between 4 and 10 digits
set_doorhandle_state	State of the door handle as string. Possible values are deactivate and activate	Updating the doorhandle state to locked or unlocked
get_doorhandle_state	-	Shows the state of the door handle. Can be "pressed" or "released"
get_doorbolt_state	-	Shows the state of the door bolt. Can be "unlocked" or "locked"
get_doorlatch_state	-	Shows the state of the door latch. Can be "closed" or "open"

User Credential Related CLI commands

The following commands exist only if User Credential Command class is available. See the section User Credential Command Class.

Command	Arguments	Description
u3c_add_user	[string] Name of the user	This command will create a new user in U3C database. The user's UUID will be autogenerated.
u3c_add_credential	[uint16] Associated User UID [string] Type of the new credential. Can be "pin" or "password" [uint16] Slot number of the new credential [string] Credential data. E.g. 3494	This command will create a new credential and assign it to an existing user.
u3c_modify_user	[uint16] User UID to modify [string] Attribute to modify. Can be "type", "name" or "active" [string] New value of the attribute	This command modifies an attribute of an existing user
u3c_modify_credential	[string] Type of the credential. Can be "pin" or "password" [uint16] Slot of the credential [string] Attribute to change. Can be "data", "uuid" or "slot" [string] New value of the attribute	This command modifies an existing credential.
u3c_delete_user	[uint16] User UID of an existing user to delete	This command deletes an existing user and all of the assigned credentials.
u3c_delete_credential	[string] Type of the credential. Can be "pin" or "password" [uint16] Slot of the credential	This command deletes an existing credential from the database.
u3c_get_user	[uint16] User UID of an existing user	This command prints the information of an existing user.
u3c_get_credential	[string] Type of the credential. Can be "pin" or "password" [uint16] Slot of the credential	This command prints the information of an existing credential.
u3c_enter_credential	[string] Type of the credential. Can be "pin" or "password" [string] Credential data	This command enters an existing credential and validates it.

Command	Arguments	Description
u3c_credential_learn_start	[uint16] Learn target UUID [string] Type of the credential. Can be "pin" or "password" [uint16] Learn target slot	Initiates the Credential Learn process locally.
u3c_credential_learn	[string] Credential data. E.g. 3494	Enter credential data for User Credential Learn process

Usage on Linux platform - zwave_soc_led_bulb

Usage on Linux platform - zwave_soc_led_bulb

Application uses single character commands read from standard input.

Note: This application supports only general commands common for all applications.

LED Bulb

LED Bulb

The Z-Wave LED Bulb application shows an implementation of a remotely controlled color switch. An example is a LED bulb or any other color capable device.

The LED Bulb application is based on:

Role Type	Always On End device (AOS)
Supporting Device Type	Actuator
Device Type	Color Switch
Generic Type	Switch Multilevel
Specific Type	Color Tunable Multilevel
Requested security keys	S0, S2_UNAUTHENTICATED, and S2_AUTHENTICATED

Supported Command Classes

The LED Bulb application implements mandatory and some optional command classes. The table below lists the supported Command Classes, their version, and their required Security class

Command Class	Version	Required Security Class
Association	2	Highest granted Security Class
Association Group Info	3	Highest granted Security Class
Basic	2	Highest granted Security Class
Color Switch	3	Highest granted Security Class
Device Reset Locally	1	Highest granted Security Class
Firmware Update Meta Data	5	Highest granted Security Class
Indicator	3	Highest granted Security Class
Manufacturer Specific	2	Highest Granted Security Class
Multi-Channel Association	3	Highest granted Security Class
Multilevel Switch	4	Highest granted Security Class
Powerlevel	1	Highest granted Security Class
Security 0	1	None
Security 2	1	None
Supervision	1	None
Transport Service	2	None
Version	3	Highest granted Security Class
Z-Wave Plus Info	2	None

Basic Command Class Mapping

The Basic Command Class is mapped according to the table below.

Basic Command	Mapped Command
Basic Set (Value)	Multilevel Switch Set (Value)
Basic Report (Current Value, Duration)	Multilevel Switch Report (Value, Duration)

Association Groups

The following association groups shown in the table below are available for Z-Wave and Z-Wave Long Range respectively.

ID	Name	Node Count	Description
1	Lifeline	X	Supports the following command classes: • Device Reset Locally: triggered upon reset. • Color Switch Report: Triggered when the switch changes state. • Multilevel Switch Report: Triggered when the switch changes state. • Indicator Report: Triggered when LED0 changes state.

X: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 1.

Usage of Buttons and LED Status

We are differentiating four different types of button presses. The following types are the same for the BTN0 and BTN1 on the WSTK board. The duration values can be configured under the config directory in app_button_press_config.h file in each generated application/project.

Please note external wakeup is not supported on button 1 in case of brd2603a and brd2603b.

Press Type	Duration
Short Press	0 - 400 ms
Medium Press	401 - 1500 ms
Long Press	1501 - 5000 ms
Very Long Press	Every press longer than Long Press

Button	Action	Description
RST	Press	Resets the firmware of an application (like losing power). All volatile memory will be cleared.
BTN1	Short Press	Enter "learn mode" (sending node info frame) to add/remove the device. Removing the device from a network will reset it.
Very Long Press	Perform a reset to factory default operation of the device, and a Device Reset Locally Notification Command is sent via Lifeline.	

LED	Description
Boards with an RGB LED	Boards without an RGB LED

LED	Description	
LED0	Blinks with 1 Hz when learn mode is active. Used for Indicator Command Class.	
LED1	-	Controlled by CC Color Switch and CC Multilevel Switch (monochrome, brightness only)
RGB	Controlled by CC Color Switch (colors) and CC Multilevel Switch (brightness)	-

Firmware Update

SDK 7.14.0 is the first SDK, including LED Bulb. When there are new versions of LED Bulb application, backward compatibility notes will be added in this section.

CLI Support

In case CLI support is needed, please install zw_cli_common component to the project. Please note that the zw_cli_common component will modify the power consumption for every application.

Available CLI commands

Command	Arguments	Description
help	-	Printing all supported CLI commands and their description.
set_learn_mode	-	Toggling the learn mode functionality. In case the learn mode started, but inclusion does not happen,learn mode will automatically stop after about 30 seconds
factory_reset	-	Executing factory reset
get_dsk	-	Printing out the generated DSK of the device
get_region	-	Printing out the set region of the application
get_rgb_values	-	Shows the current Red Green and Blue values. The range is 0-65535 for each component. In case the target does not have RGB LED it shows the LED intensity on a scale 0-255.
toggle_led_bulb	-	Toggle "LED Bulb" on/off

Usage on Linux platform - zwave_soc_multilevel_sensor

Usage on Linux platform - zwave_soc_multilevel_sensor

Application uses single character commands read from standard input.

Character	Description
1	Send battery report

Multilevel Sensor

Multilevel Sensor

Shows the ability to advertise numerical sensor readings, such as temperature, and humidity. Multiple parameters can be set for the minimum and maximum values.

The Multilevel Sensor application is based on:

Role Type	Reporting Sleeping End Device (RSS)
Supporting Device Type	Data reporting
Device Type	Notification sensor
Generic Type	Sensor Notification
Specific Type	Notification Sensor
Requested security keys	S2_UNAUTHENTICATED and S2_AUTHENTICATED

Multilevel Sensor transmits the following events:

- Environment monitoring
 - Temperature, and humidity measure

Supported Command Classes

Multilevel Sensor implements mandatory and some optional command classes. The table below lists the supported Command Classes, their version, and their required Security class, if any.

Command Class	Version	Required Security Class
Association	2	Highest granted Security Class
Association Group Info	3	Highest granted Security Class
Device Reset Locally	1	Highest granted Security Class
Firmware Update Meta Data	5	Highest granted Security Class
Indicator	3	Highest granted Security Class
Manufacturer Specific	2	Highest Granted Security Class
Multi-Channel Association	3	Highest granted Security Class
Powerlevel	1	Highest granted Security Class
Security 2	1	None
Supervision	1	None
Transport Service	2	None
Version	3	Highest granted Security Class
Wake Up	2	Highest granted Security Class
Z-Wave Plus Info	2	None
Configuration	4	Highest granted Security Class
Multilevel-Sensor	11	Highest granted Security Class

Basic Command Class Mapping

Basic Command Class is not mapped to any of the supported command classes.

Association Groups

The following table shows the available association groups.

ID	Name	Node Count	Description
1	Lifeline	5	Supports the following command classes: • Device Reset Locally: triggered upon reset. • Indicator Report: Triggered when LED0 changes state. • Configuring parameters: Minimum and maximum temperature levels can be set, errors can be detected if measured temperature is out of range • Environmental measurements: Temperature and humidity values can be read, triggered from other Z-Wave devices

Usage of Buttons and LED Status

We are differentiating four different types of button presses. The following types are the same for the BTN0 and BTN1 on the WSTK board. The duration values can be configured under the config directory in app_button_press_config.h file in each generated application/project.

Please note external wakeup is not supported on button 1 in case of brd2603a and brd2603b.

Press Type	Duration
Short Press	0 - 400 ms
Medium Press	401 - 1500 ms
Long Press	1501 - 5000 ms
Very Long Press	Every press longer than Long Press

Button	Action	Description
RST	Press	Resets the firmware of an application (like losing power). All volatile memory will be cleared.
BTN0	Short Press	Sends Battery Report, temperature, and humidity data
BTN1	Short Press	Enter "learn mode" (sending node info frame) to add/remove the device. Removing the device from a network will reset it.
Very Long Press	Perform a reset to factory default operation of the device, and a Device Reset Locally Notification Command is sent via Lifeline.	

LED	Description
LED0	Blinks with 1 Hz when learn mode is active. Used for Indicator Command Class.

Firmware Update

This section will describe backward compatibility when upgrading the MultilevelSensor application from one SDK to a newer version.

CLI Support

In case CLI support is needed, please install `zw_cli_common` component to the project. Please note that the `zw_cli_common` component will significantly increase the power consumption of this application, as it must always be in an active state in order to receive CLI commands. The CLI cannot function during EM4 sleep mode. After a reset, the application stays awake for 10 seconds. During this timeframe, the sleeping disable CLI command can be issued to keep the application awake.

Available CLI commands

Command	Arguments	Description
help	-	Printing all supported CLI commands and their description.
set_learn_mode	-	Toggling the learn mode functionality. In case the learn mode started, but inclusion does not happen, learn mode will automatically stop after about 30 seconds
factory_reset	-	Executing factory reset
get_dsk	-	Printing out the generated DSK of the device
get_region	-	Printing out the set region of the application
send_reports	-	Sending battery and sensor reports
sleeping	[string] "enable" or "disable"	Enable or disable sleeping. After pushing the reset button (or resetting with commander) the device will be awake for a given amount of time if the CLI component is added to the project. During this time the user can prevent the sleeping. For more information check the <code>zw_cli_sleeping</code> component.

Usage on Linux platform - zwave_soc_power_strip

Usage on Linux platform - zwave_soc_power_strip

Application uses single character commands read from standard input.

Character	Description
1	Outlet 1 toggle on/off
2	Outlet 2 dim/on/off
3	Dim up/down start
4	Dim up/down stop
5	Toggle notification every 30 sec

Power Strip

Power Strip

The Z-Wave certified Power Strip application shows an extension block implementation used to turn on several devices that are connected to power. Examples include lights, appliances, etc.

The Power Strip Z-Wave certified application implements several endpoints, which can be controlled individually.

The Power Strip application is based on:

Role Type	Always On End device (AOS)
Supporting Device Type	Actuator
Device Type	Binary Switch
Generic Type	Switch Binary
Specific Type	Not Used
Requested security keys	S0, S2_UNAUTHENTICATED, and S2_AUTHENTICATED
Endpoint 1 Device Type	Binary Switch
Endpoint 1 Generic Type	Switch Binary
Endpoint 1 Specific Type	Not Used
Endpoint 2 Device Type	Multilevel Switch
Endpoint 2 Generic Type	Switch Multilevel
Endpoint 2 Specific Type	Not Used

Power Strip supports the "push mode" only of Notification CC and transmits the following notification types/events:

- Power Management
 - Overload detected (both endpoint 1 and 2)

Supported Command Classes

The Power Strip application implements mandatory and some optional command classes. The table below lists the supported command classes, their version, and their required Security class, if any.

Command Class	Version	Required Security Class
Association	2	Highest Granted Security Class
Association Group Info	3	Highest Granted Security Class
Basic	2	Highest Granted Security Class
Device Reset Locally	1	Highest Granted Security Class
Firmware Update Meta Data	5	Highest Granted Security Class
Indicator	3	Highest Granted Security Class
Manufacturer Specific	2	Highest Granted Security Class
Multi-Channel	4	Highest Granted Security Class
Multi-Channel Association	3	Highest Granted Security Class
Notification	8	Highest Granted Security Class

Command Class	Version	Required Security Class
Powerlevel	1	Highest Granted Security Class
Security 0	1	None
Security 2	1	None
Supervision	1	None
Switch Binary	2	Highest Granted Security Class
Transport Service	2	None
Version	3	Highest Granted Security Class
Z-Wave Plus Info	2	None

Supported Command Classes: Endpoint 1

Endpoint 1 implements the following command classes.

Command Class	Version	Required Security Class
Association	2	Highest Granted Security Class
Association Group Info	3	Highest Granted Security Class
Basic	2	Highest Granted Security Class
Multi-Channel Association	3	Highest Granted Security Class
Notification	8	Highest Granted Security Class
Security 0	1	None
Security 2	1	None
Supervision	1	None
Switch Binary	2	Highest Granted Security Class
Z-Wave Plus Info	2	None

Supported Command Classes: Endpoint 2

Endpoint 2 implements the following command classes.

Command Class	Version	Required Security Class
Association	2	Highest Granted Security Class
Association Group Info	3	Highest Granted Security Class
Basic	2	Highest Granted Security Class
Multi-Channel Association	3	Highest Granted Security Class
Notification	8	Highest Granted Security Class
Security 0	1	None
Security 2	1	None
Supervision	1	None
Multilevel Switch	4	Highest Granted Security Class
Z-Wave Plus Info	2	None

Basic Command Class mapping

The Basic Command Class is mapped according to the next table.

Endpoint	Basic Command	Mapped Command
1	Basic Set (Value)	Binary Switch Set (Value)
1	Basic Report (Current Value, Duration)	Binary Switch Report (Value, Duration)
2	Basic Set (Value)	Multilevel Switch Set (Value)
2	Basic Report (Current Value, Duration)	Multilevel Switch Report (Value, Duration)

Association Groups

The following three tables show the available association groups for Z-Wave and Z-Wave Long Range respectively. Backward compatibility for non-Multi Channel devices forces the root device AGI table to contain all the association groups mentioned in each of the endpoints AGI tables except from group 1, the Lifeline group.

Association groups for Root Device:

ID	Name	Node Count	Description
1	Lifeline	X	Supports the following command classes: - Device Reset Locally: triggered upon reset. - Binary Switch Report: Triggered when the switch changes state. - Notification: triggered by the endpoints - Indicator Report: Triggered when LED0 changes state.
2	alarm EP 1	Y	Mirror of endpoint 1, group 2
3	alarm EP 2	Y	Mirror of endpoint 2, group 2

X: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 1.

Y: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 0.

Association groups for Endpoint 1:

ID	Name	Node Count	Description
1	Lifeline	0	Supports the following command classes: - Binary Switch Report: Triggered when the switch changes state. - Notification: triggered by the endpoints
2	alarm EP 1	Y	Notification report on overload.

Y: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 0.

Association groups for Endpoint 2:

ID	Name	Node Count	Description
1	Lifeline	0	Supports the following command classes: - Multilevel Switch Report: Triggered when the switch changes state. - Notification: triggered by the endpoints
2	alarm EP 2	Y	Notification report on overload.

Y: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 0.

Usage of Buttons and LED Status

We are differentiating four different types of button presses. The following types are the same for the BTN0 and BTN1 on the WSTK board. The duration values can be configured under the config directory in app_button_press_config.h file in each generated application/project.

Please note external wakeup is not supported on button 1 in case of brd2603a and brd2603b.

Press Type	Duration
Short Press	0 - 400 ms
Medium Press	401 - 1500 ms
Long Press	1501 - 5000 ms
Very Long Press	Every press longer than Long Press

Button	Action	Description
RST	Press	Resets the firmware of an application (like losing power). All volatile memory will be cleared.
BTN0	Short Press	Toggles the transmission of an "Overload detected" notification. The first transmission, when enabled, will always be the first in the following list. The notifications will be transmitted in the following pattern: 1. Notification from endpoint 1: Overload detected 2. Notification from endpoint 1: No event 3. Notification from endpoint 2: Overload detected 4. Notification from endpoint 2: No event One notification will be transmitted every 30 seconds.
Medium Press	Toggles the binary switch	
BTN1	Short Press	Enter "learn mode" (sending node info frame) to add/remove the device. Removing the device from a network will reset it.
Very Long Press	Perform a reset to factory default operation of the device, and a Device Reset Locally Notification Command is sent via Lifeline.	

LED	Description for boards with RGB LED	Description for boards without RGB LED ¹
LED0	Blinks with 1 Hz when learn mode is active. Used for Indicator Command Class.	
LED1	Endpoint 1 (switch on/off)	Endpoint 2 (dimmer, average of the three colors)
RGB	Endpoint 2 (dimmer)	-

¹: Some boards do not have an RGB LED. On such boards, the command line interface can be used to monitor the binary switch endpoint and all the levels of the multilevel switch endpoint.

Firmware Update

This section will describe backward compatibility when upgrading the Power Strip application from one SDK to a newer version.

CLI Support

In case CLI support is needed, please install `zw_cli_common` component to the project. Please note that the `zw_cli_common` component will modify the power consumption for every application.

Available CLI commands

Command	Arguments	Description
<code>help</code>	-	Printing all supported CLI commands and their description.
<code>set_learn_mode</code>	-	Toggling the learn mode functionality. In case the learn mode started, but inclusion does not happen, learn mode will automatically stop after about 30 seconds
<code>factory_reset</code>	-	Executing factory reset
<code>get_dsk</code>	-	Printing out the generated DSK of the device
<code>get_region</code>	-	Printing out the set region of the application
<code>toggle_endpoint</code>	Number of your endpoint you want to toggle. Can be 1 or 2	Toggling endpoint 1 or endpoint 2
<code>dim_endpoint</code>	New dimming level, it can be minimum 0 and maximum 99	Setting the dimmin for endpoint 2
<code>toggle_notification_sending</code>	-	Toggles the transmission of an "Overload detected" notification. The first transmission, when enabled, will always be the first in the following list.
<code>get_led_state</code>	-	Get the state of the LED1. Can be "on" or "off". Only available on hardware with an RGB LED (otherwise, LED1 is assigned to the multilevel switch and its state can be queried with <code>get_rgb_values</code>).
<code>get_rgb_values</code>	-	Shows the current Red Green and Blue values. The range is 0-65535 for each component. In case the target does not have RGB LED it shows the LED intensity on a scale 0- 255.

Usage on Linux platform - zwave_soc_sensor_pir

Usage on Linux platform - zwave_soc_sensor_pir

Application uses single character commands read from standard input.

Character	Description
1	Send battery report
2	Activate sensor
3	Deactivate sensor

Sensor PIR

Sensor PIR

The Z-Wave certified Sensor PIR application shows a presence/movement detector implementation for controlling other devices and for sending notifications.

Role Type	Reporting Sleeping End device (RSS)
Supporting Device Type	Data reporting
Device Type	Notification sensor
Generic Type	Sensor Notification
Specific Type	Notification Sensor
Requested security keys	S2_UNAUTHENTICATED, and S2_AUTHENTICATED

Sensor PIR transmits the following notification types/events:

- Home Security
 - Motion detection (unknown location)

Sensor PIR supports the "push mode" only of Notification CC.

Supported Command Classes

Sensor PIR implements mandatory and some optional command classes. The table below lists the supported Command Classes, their version, and their required Security class, if any.

Command Class	Version	Required Security Class
Association	2	Highest granted Security Class
Association Group Info	3	Highest granted Security Class
Battery	1	Highest granted Security Class
Device Reset Locally	1	Highest granted Security Class
Firmware Update Meta Data	5	Highest granted Security Class
Indicator	3	Highest granted Security Class
Manufacturer Specific	2	Highest Granted Security Class
Multi-Channel Association	3	Highest granted Security Class
Notification	8	Highest granted Security Class
Powerlevel	1	Highest granted Security Class
Security 2	1	None
Supervision	1	None
Transport Service	2	None
Version	3	Highest granted Security Class
Wake Up	2	Highest granted Security Class
Z-Wave Plus Info	2	None

Basic Command Class mapping

Basic Command Class is not mapped to any of the supported command classes

Association Group configuration

Application Association Group configuration

ID	Name	Node Count	Description
1	Lifeline	X	Supports the following command classes: • Device Reset Locally: triggered upon reset. • Battery: triggered upon low battery. • Notification: triggered upon movement detection (simulated by Medium Press of BTN1). After a while, a cancel notification will be issued. • Indicator Report: Triggered when LED0 changes state.
2	Basic Set	Y	Upon a movement detection (simulated by Medium Press of BTN1), nodes associated in this group will first receive a Basic Set with 0xFF (turn on) and after a while receive a Basic Set with 0x00 (turn off).

X: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 1. Y: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 0.

Usage of Buttons and LED Status

We are differentiating four different types of button presses. The following types are the same for the BTN0 and BTN1 on the WSTK board. The duration values can be configured under the config directory in app_button_press_config.h file in each generated application/project.

Please note external wakeup is not supported on button 1 in case of brd2603a and brd2603b.

Press Type	Duration
Short Press	0 - 400 ms
Medium Press	401 - 1500 ms
Long Press	1501 - 5000 ms
Very Long Press	Every press longer than Long Press

Button	Action	Description
RST	Press	Resets the firmware of an application (like losing power). All volatile memory will be cleared.
BTN0	Short Press	Sends Battery Report
Medium Press	Toggles "motion detected" event simulation.	
BTN1	Short Press	Enter "learn mode" (sending node info frame) to add/remove the device. Removing the device from a network will reset it.
Very Long Press	Perform a reset to factory default operation of the device, and a Device Reset Locally Notification Command is sent via Lifeline.	

LED	Description
LED0	Blinks with 1 Hz when learn mode is active. Used for Indicator Command Class.

Firmware Update

This section will describe backward compatibility when upgrading the Switch On/Off application from one SDK to a newer version.

CLI Support

In case CLI support is needed, please install `zw_cli_common` component to the project. Please note that the `zw_cli_common` component will significantly increase the power consumption of this application, as it must always be in an active state in order to receive CLI commands. The CLI cannot function during EM4 sleep mode. After a reset, the application stays awake for 10 seconds. During this timeframe, the sleeping disable CLI command can be issued to keep the application awake.

Available CLI commands

Command	Arguments	Description
help	-	Printing all supported CLI commands and their description.
set_learn_mode	-	Toggling the learn mode functionality. In case the learn mode started, but inclusion does not happen, learn mode will automatically stop after about 30 seconds
factory_reset	-	Executing factory reset
get_dsk	-	Printing out the generated DSK of the device
get_region	-	Printing out the set region of the application
battery_report	-	Sending a battery report
motion_detected	-	Triggering a motion detected event
sleeping	[string] "enable" or "disable"	Enable or disable sleeping. After pushing the reset button (or resetting with commander) the device will be awake for a given amount of time if the CLI component is added to the project. During this time the user can prevent the sleeping. For more information check the <code>zw_cli_sleeping</code> component.

Usage on Linux platform - zwave_soc_switch_on_off

Usage on Linux platform - zwave_soc_switch_on_off

Application uses single character commands read from standard input.

Character	Description
1	Toggle switch on/off
2	Send Included NIF

Switch On/Off

Switch On/Off

The Z-Wave certified Switch On/Off application shows a switch implementation, that turns on any device that is connected to power. Examples include lights, appliances, etc.

Role Type	Always On End device (AOS)
Supporting Device Type	Actuator
Device Type	Binary Switch
Generic Type	Switch Binary
Specific Type	Not Used
Requested security keys	S0, S2_UNAUTHENTICATED, and S2_AUTHENTICATED

Supported Command Classes

The Switch On/Off application implements mandatory and some optional command classes. The table below lists the supported Command Classes, their version, and their required Security class.

Command Class	Version	Required Security Class
Association	2	Highest granted Security Class
Association Group Info	3	Highest granted Security Class
Basic	2	Highest granted Security Class
Binary Switch	2	Highest granted Security Class
Device Reset Locally	1	Highest granted Security Class
Firmware Update Meta Data	5	Highest granted Security Class
Indicator	3	Highest granted Security Class
Manufacturer Specific	2	Highest Granted Security Class
Multi-Channel Association	3	Highest granted Security Class
Powerlevel	1	Highest granted Security Class
Security 0	1	None
Security 2	1	None
Supervision	1	None
Transport Service	2	None
Version	3	Highest granted Security Class
Z-Wave Plus Info	2	None

Basic Command Class mapping

Basic Command Class is mapped to Binary Switch CC

Basic Command	Mapped Command
Basic Set (Value)	Binary Switch Set (Value)
Basic Report (Current Value, Duration)	Binary Switch Report (Value, Duration)

Association Group configuration

ID	Name	Node Count	Description
1	Lifeline	X	Supports the following command classes: • Device Reset Locally: triggered upon reset. • Binary Switch Report: Triggered when the switch changes state. • Indicator Report: Triggered when LED0 changes state.

X: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 1.

Usage of Buttons and LED Status

We are differentiating four different types of button presses. The following types are the same for the BTN0 and BTN1 on the WSTK board. The duration values can be configured under the config directory in app_button_press_config.h file in each generated application/project.

Please note external wakeup is not supported on button 1 in case of brd2603a and brd2603b.

Press Type	Duration
Short Press	0 - 400 ms
Medium Press	401 - 1500 ms
Long Press	1501 - 5000 ms
Very Long Press	Every press longer than Long Press

Button	Action	Description
RST	Press	Resets the firmware of an application (like losing power). All volatile memory will be cleared.
BTN0	Short Press	Toggles LED1
BTN1	Short Press	Enter "learn mode" (sending node info frame) to add/remove the device. Removing the device from a network will reset it.
Very Long Press	Perform a reset to factory default operation of the device, and a Device Reset Locally Notification Command is sent via Lifeline.	

LED	Description
LED0	Blinks with 1 Hz when learn mode is active. Used for Indicator Command Class.
LED1	Switch: • LED ON indicates SWITCH ON • LED OFF indicates SWITCH OFF

Firmware Update

This section will describe backward compatibility when upgrading the Switch On/Off application from one SDK to a newer version.

CLI Support

In case CLI support is needed, please install zw_cli_common component to the project. Please note that the zw_cli_common component will modify the power consumption for every application.

Available CLI commands

Command	Arguments	Description
help	-	Printing all supported CLI commands and their description.
help	-	Printing all supported CLI commands and their description.
set_learn_mode	-	Toggling the learn mode functionality. In case the learn mode started, but inclusion does not happen,learn mode will automatically stop after about 30 seconds
factory_reset	-	Executing factory reset
get_dsk	-	Printing out the generated DSK of the device
get_region	-	Printing out the set region of the application
toggle_led	-	Toggling LED1
send_nif	-	Sending Node Information Frame
get_led_state	-	Get the state of the LED1. Can be "on" or "off"

Usage on Linux platform - zwave_soc_wall_controller

Usage on Linux platform - zwave_soc_wall_controller

Application uses single character commands read from standard input.

Character	Description
1	KEY01 press
2	KEY01 hold
3	KEY01 release
4	KEY02 press
5	KEY02 hold
6	KEY02 release
7	KEY03 press
8	KEY03 hold
9	KEY03 release

Wall Controller

Wall Controller

The Z-Wave certified Wall Controller application shows a push-button switch panel implementation to control devices in the Z-Wave network from push buttons (physical or virtual) on a device that is meant to be mounted on a wall. Examples include scene and zone controller and wall-mounted AV controllers.

The Wall controller application implements a wall-mounted switch that has three individually configurable buttons. Each button has support for on/off and dimming.

The Wall Controller application is based on:

Role Type	Always On End device (AOS)
Supporting Device Type	Actuator
Device Type	Central Scene
Generic Type	Wall Controller
Specific Type	Not Used
Requested security keys	S0, S2_UNAUTHENTICATED, and S2_AUTHENTICATED The Wall Controller features do not depend on the security level.

Supported Command Classes

Command Class	Version	Required Security Class
Association	2	Highest granted Security Class
Association Group Info	3	Highest granted Security Class
Central Scene	3	Highest granted Security Class
Device Reset Locally	1	Highest granted Security Class
Firmware Update Meta Data	5	Highest granted Security Class
Indicator	3	Highest granted Security Class
Manufacturer Specific	2	Highest Granted Security Class
Multi-Channel Association	3	Highest granted Security Class
Powerlevel	1	Highest granted Security Class
Security 0	1	None
Security 2	1	None
Supervision	1	None
Transport Service	2	None
Version	3	Highest granted Security Class
Z-Wave Plus Info	2	None

Use case:

The wall controller's three buttons are configurable to control nodes. Each button action is sent over the Lifeline as scene actions.

Basic Command Class mapping

Basic Command Class is not mapped to any of the supported command classes.

Association Groups

The following association groups are available for Z-Wave and Z-Wave Long Range respectively.

ID	Name	Node Count	Description
1	Lifeline	X	Supports the following command classes: • Device Reset Locally: triggered upon reset. • Central Scene: triggered by button press. • Indicator Report: Triggered when LED0 changes state.
2	KEY1	Y	Basic Set Command
3	Y	Switch Multilevel CC	
4	KEY2	Y	Basic Set Command
5	Y	Switch Multilevel CC	
6	KEY3	Y	Basic Set Command
7	Y	Switch Multilevel CC	

X: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 1.

Y: For Z-Wave node count is equal to 5 and for Z-Wave Long Range it is 0.

Usage of Buttons and LED Status

We are differentiating four different types of button presses. The following types are the same for the BTN0 and BTN1 on the WSTK board. The duration values can be configured under the config directory in app_button_press_config.h file in each generated application/project.

Please note external wakeup is not supported on button 1 in case of brd2603a and brd2603b.

Press Type	Duration
Short Press	0 - 400 ms
Medium Press	401 - 1500 ms
Long Press	1501 - 5000 ms
Very Long Press	Every press longer than Long Press

Button	Action	Description
RST	Press	Resets the firmware of an application (like losing power). All volatile memory will be cleared.
BTN0	Short Press	Triggers association group 2: Basic Set
Hold longer than Short Press	Association group 3: Multilevel Switch Start Level Change	
Released after being held longer than Short Press	Association group 3: Multilevel Switch Stop Level Change	

Button	Action	Description
BTN1	Short Press	Enter "learn mode" (sending node info frame) to add/remove the device. Removing the device from a network will reset it.
Very Long Press	Perform a reset to factory default operation of the device, and a Device Reset Locally Notification Command is sent via Lifeline.	

LED	Description
LED0	Blinks with 1 Hz when learn mode is active. Used for Indicator Command Class.

Firmware Update

This section will describe backward compatibility when upgrading the Wall Controller application from one SDK to a newer version.

Firmware update from previous versions should work without any additional steps.

CLI Support

In case CLI support is needed, please install `zw_cli_common` component to the project. Please note that the `zw_cli_common` component will modify the power consumption for every application.

Available CLI commands

Command	Arguments	Description
help	-	Printing all supported CLI commands and their description.
set_learn_mode	-	Toggling the learn mode functionality. In case the learn mode started, but inclusion does not happen, learn mode will automatically stop after about 30 seconds
factory_reset	-	Executing factory reset
get_dsk	-	Printing out the generated DSK of the device
get_region	-	Printing out the set region of the application
send_central_scene_key	First argument is the number of the key, which could be 1, 2 or 3. Second argument is used to determine the action of the specified key, which can be press, hold or release.	Triggering central scene key

Z-Wave Command Class Configurator

Z-Wave Command Class Configurator

Some command classes require more advanced configuration parameters and for those an advanced configurator was created.

The advanced configurator will be executed for all projects (`slcp`) which contain a `configuration_file` with the extension (`cc_config`)

Specification

The `cc_config` file is a yaml file with a defined format per command class. The basic format is

```
<CC_1>:
...
<CC_2>:
...
<CC_n>:
...
```

Where `CC_n`, is the component (`slcc`) id.

Each command class defines it's properties based on the requirements for it.

Command Classes - Specification

Below each Command Class properties are defined Non mandatory fields are displayed within parentheses

zw_cc_multi_channel_support

This section configures the generic type and specific type for each endpoint.

This configuration is only relevant for a multi channel device like a power strip or a sensor with multiple sensors of the same type, for example two temperature sensors.

```
zw_cc_multi_channel_support:
  endpoints:
    - endpoint: 1
      generic_type: <generic type for endpoint 1>
      specific_type: <specific type for endpoint 1>
    - endpoint: 2
      generic_type: <generic type for endpoint 2>
      specific_type: <specific type for endpoint 2>
    - endpoint: <N-127>
      generic_type: <generic type>
      specific_type: <specific type>
```

Please see `PowerStrip.cc_config` for an example.

zw_cc_binary_switch

The Command Class Binary Switch advanced configurator is defined as follows:

```
zw_cc_binary_switch:
  binary_switches:
    - endpoint: <Integer 0-255>
    ...
    - endpoint: <Integer 0-255>
```

zw_cc_color_switch

The Command Class Color Switch advanced configurator is defined as follows:

```
zw_cc_color_switch:
  colors:
    - id: <EColorComponents>
      (end_point: <Integer 0-255>)
    ...
    - id: <EColorComponents>
      (end_point: <Integer 0-255>)
```

zw_cc_multilevel_switch_support

The Command Class Multilevel Switch Support advanced configurator is defined as follows:

```
zw_cc_multilevel_switch_support:
  switches:
    - end_point: <Integer 0-255>
    ...
    - end_point: <Integer 0-255>
```

zw_cc_multilevel_sensor

The Command Class Multilevel Sensor advanced configurator is defined as follows:

```
zw_cc_multilevel_sensor:
  sensors:
    <sensor_instance>:
      name: <sensor_name_t>
      scales:
        - <SENSOR_SCALE_*>
        - <SENSOR_SCALE_*>
    ...
    <sensor_instance>:
      name: <sensor_name_t>
      scales:
        - <SENSOR_SCALE_*>
```

zw_cc_multilevel_sensor

The Command Class Configuration configurator is defined as follows:

```

zw_cc_configuration:
  configurations:
    - name: <String>
      number: <1-65535>
      file_id: <0-99>
      info: <String>
      size: <cc_config_parameter_size_t>
      format: <cc_config_parameter_format_t>
      min_value: <Integer - Range depends on size and format>
      max_value: <Integer - Range depends on size and format>
      default_value: <Integer - Range depends on size and format>
      altering_capabilities: <Bit>
      read_only: <Bit>
      advanced: <Bit>
    ...
    - name: <String>
      number: <1-65535>
      file_id: <0-99>
      info: <String>
      size: <cc_config_parameter_size_t>
      format: <cc_config_parameter_format_t>
      min_value: <Integer - Range depends on size and format>
      max_value: <Integer - Range depends on size and format>
      default_value: <Integer - Range depends on size and format>
      altering_capabilities: <Bit>
      read_only: <Bit>
      advanced: <Bit>

```

zw_cc_door_lock

The Command Class Configuration configurator is defined as follows:

```

zw_cc_configuration:
  configuration:
    operation_type: <cc_door_lock_operation_type_t>
    supported_inside_handles: <cc_door_lock_handle_t>
    supported_outside_handles: <cc_door_lock_handle_t>
    options_flags: <Integer 0-255>
    max_auto_relock_time: <Integer 0-65535>
    max_hold_and_release_time: <Integer 0-65535>

```

zw_cc_notification

The Command Class Notification configurator is defined as follows:

```

zw_cc_notification:
  notifications:
    - type: <notification_type_t>
      (endpoint: <Integer 0-127>)
      agi_profile: <agi_profile_t>
      state_events:
        - <notification_event_state 1>
        - <notification_event_state 2>
        - <notification_event_state N>
    ...
    - type: <notification_type_t>
      (endpoint: <Integer 0-127>)
      agi_profile: <agi_profile_t>
      state_events:
        - <notification_event_state 1>
        - <notification_event_state 2>
        - <notification_event_state N>

```

zw_cc_agi

The Command Class Association Group Information configurator is defined as follows

```

zw_cc_association:
  endpoints:
    - endpoint: <Integer 0-127>
  groups:
    - name: <String>
      profile_msb: <Integer>
      profile_lsb: <Integer>
  commands:
    - command_class: <Integer>
      command: <Integer>

```

The endpoints have to be specified in ascending order.

zw_cc_central_scene

The Command Class Central Scene configurator is defined as follows

```

zw_cc_central_scene:
  central_scene:
    identical: <Integer 0-1 - Mandatory field only if all scenes are identical>
    number_of_scenes: <1-255 - Mandatory field if and only if all scenes are identical>
    scenes:
      - scene: <Integer 0 - number_of_scenes>
        attributes:
          - attribute: CENTRAL_SCENE_NOTIFICATION_KEY_ATTRIBUTES_KEY_PRESSED_1_TIME_V2
          - attribute: CENTRAL_SCENE_NOTIFICATION_KEY_ATTRIBUTES_KEY_RELEASED_V2
        ...
      - scene: <Integer 0-127 First scene is mandatory. The rest of scenes are mandatory if and only if scenes are not identical>
        attributes:
          - attribute: CENTRAL_SCENE_NOTIFICATION_KEY_ATTRIBUTES_KEY_RELEASED_V2
          - attribute: CENTRAL_SCENE_NOTIFICATION_KEY_ATTRIBUTES_KEY_HELD_DOWN_V2
        ...

```

Z-Wave Platform Abstraction Layer

Z-Wave Platform Abstraction Layer

This folder contains the definition and implementations of the Z-Wave Platform Abstraction Layer (ZPAL).

Includes

ZPAL consists of a number of header files each declaring an interface. They are located in `inc/`.

Source files

Implementations are located in `../platform/<platform name>/PAL/src/*`.

Documentation

Coding standard is described by `doc/coding-standard.md`. Doxygen related files are in `doc/doxygen/`.

Mocks

Implementation of mocks for each ZPAL module are located in `mocks/`.

Templates

Folder `templates` contains files that can be used as starting point when porting ZPAL to a new platform.

ZPAL declaration

ZPAL declaration

This folder contains the declaration of the Z-Wave Platform Abstraction Layer (ZPAL).

Silicon Labs ZPAL

Silicon Labs ZPAL

This section covers information about the ZPAL specific to the Silicon Labs implementation.

UART

ZPAL_UART0 is hardcoded to use Silabs USART0.

Any other UART is not supported via ZPAL. If you want to use any other UART/USART, please use the Silabs API for that. Alternatively, request support for an additional UART in ZPAL.

Deprecated List

Deprecated List

- Global [E_SYSTEM_TYPE](#)
Use [Application Node Info](#) instead
Use [APPLICATION_NODEINFO_LISTENING](#) instead
- Global [FUNC_ID_APPLICATION_SLAVE_COMMAND_HANDLER](#)
OBSOLETE: In DevKit 4.5x/6.0x Controller Bridge applications, this is obsoleted by the [FUNC_ID_APPLICATION_COMMAND_HANDLER_BRIDGE](#)
- Global [FUNC_ID_ZW_APPLICATION_CONTROLLER_UPDATE](#)
Use [ZW_APPLICATION_UPDATE](#)
- Global [FUNC_ID_ZW_SET_EXT_INT_LEVEL](#)
Use [FUNC_ID_ZW_INT_EXT_LEVEL_SET](#)
- Global [SYSTEM_TYPE_LISTENING_MASK](#)
Use [APPLICATION_FREQ_LISTENING_MODE_1000ms](#) or [APPLICATION_FREQ_LISTENING_MODE_250ms](#) instead
- Global [Transport_ApplicationCommandHandlerEx](#) ([RECEIVE_OPTIONS_TYPE_EX](#) *rxOpt, [ZW_APPLICATION_TX_BUFFER](#) *pCmd, uint8_t cmdLength)
If all command classes use one of the [REGISTER_CC](#) macros, this function is not required to be implemented in the application. There exists a weak definition of it. [ZAF_CC_Invoker](#) will take care of invoking the right command class handler.
- Global [Transport_OnLearnCompleted](#) (node_id_t nodeID)
Use [ZAF_Transport_OnLearnCompleted\(\)](#).

Overview

Examples Overview

This section contains:

- [How to Use Certified Apps \(PDF\)](#): Describes how to use the Z-Wave certified applications, which come as part of the Z-Wave SDK. The document explains the real-life application that a specific certified app is modelling, the Z-Wave specific configuration being used, and the functionality of buttons and LEDs on the expansion board.
- [How to Use Pre-Certified Apps \(PDF\)](#): Describes how to use the Z-Wave pre-certified applications, which come as part of the Z-Wave SDK. They differ from the certified applications in that these applications are not formally certified but they do pass the CTT 3 test suite. The document explains the real-life application that a specific pre-certified app is modelling, the Z-Wave specific configuration being used, and the functionality of buttons and LEDs on the expansion board.

Best Practices for Z-Wave Door Locks

This multi-part series is written by Dr. ZWave, Eric Ryherd. It brings together the many issues unique to door locks and guides the developer toward the most robust and interoperable implementation. These are mostly recommendations, not requirements, and do not guarantee Z-Wave certification.

- [Best Practices for Z-Wave Door Locks \(Part 1\)](#)
- [Best Practices for Z-Wave Door Locks \(Part 2\)](#)
- [Best Practices for Z-Wave Door Locks \(Part 3\)](#)
- [Best Practices for Z-Wave Door Locks \(Part 4\)](#)
- [Best Practices for Z-Wave Door Locks \(Part 5\)](#)
- [Best Practices for Z-Wave Door Locks \(Part 6\)](#)

Additional Topics

Examples: Additional Topics

[Missing Command Classes in Application Framework](#)