



Telink

Application Note

Telink Zigbee SDK Developer Manual

AN-19052900-E6

Ver1.2.2
2023.7.24

Keyword

Zigbee

Brief

This document is a development guide for the Telink Zigbee SDK.

Published by
Telink Semiconductor

Bldg 3, 1500 Zuchongzhi Rd,
Zhangjiang Hi-Tech Park, Shanghai, China

© Telink Semiconductor
All Rights Reserved

Legal Disclaimer

This document is provided as-is. Telink Semiconductor reserves the right to make improvements without further notice to this document or any products herein. This document may contain technical inaccuracies or typographical errors. Telink Semiconductor disclaims any and all liability for any errors, inaccuracies or incompleteness contained herein.

Copyright © 2023 Telink Semiconductor (Shanghai) Co., Ltd.

Information

For further information on the technology, product and business term, please contact Telink Semiconductor Company www.telink-semi.com

For sales or technical support, please send email to the address of:

telinksales@telink-semi.com

telinksupport@telink-semi.com

Revision History

Versions	Major changes
V1.0.0	Initial release
V1.1.0	Added 2.2 TLSR9 RISC-V SDK Installation. Added 2.3 App Version Management. Updated 2.4.2 Flash Address Assignment Instructions. Added 4.3.4 Network Security. Add 4.4.1 AF.
V1.1.1	Classified 8269 (b86), 8258 (b85), 8278 (b87) as B85m and 9518 (b91) as B91m. Added 4.3.11 NV Storage, 4.3.12 Exception Processing. Updated section 4.4 for common APIs.
V1.2.0	Added 2.4 Operational Mode. Added 4.1 operation Mode Selection and 4.2 Chip Type Selection. Updated 4.5.8 Software Timer Task and 4.5.9 Sleep and Wake up. Added 4.7.4 OTA Image.
V1.2.1	Updated section 7 HCI message format for tl_zigbee_SDK version after 3.6.6.0.
V1.2.2	Updated section 2.5 Flash allocation description. Updated section 3.2 Abstract layer driver. Updated section 3.3 Memory management. Updated section 3.4 Task management. Updated section 4.4.3 Network security. Updated section 6.4 OTA image. Added chapter 7 Bootloader.

Contents

Revision History	3
1 Overview	11
1.1 Zigbee brief	11
1.1.1 Device type	11
1.1.2 Network type	11
1.1.3 Basic concept	11
1.2 Telink Zigbee SDK brief	13
1.2.1 Software development environment for Telink Zigbee SDK	13
1.2.2 Hardware platforms supported by Telink Zigbee SDK	14
2 Telink Zigbee SDK	15
2.1 TLSR8 TC32 SDK Installation	15
2.1.1 Project Import	15
2.1.2 Project directory structure for TLSR8	18
2.1.3 Compile options	19
2.1.4 Add new project	20
2.1.5 Project configuration description	22
2.2 TLSR9 RISC-V SDK installation	26
2.2.1 Project import	26
2.2.2 Project directory structure for TLSR9	29
2.2.3 Compile options	30
2.2.4 Add new project	31
2.2.5 Project configuration description	32
2.3 App version management	36
2.3.1 Manufacturer Code	37
2.3.2 Image Type	37
2.3.3 File Version	37
2.4 Operation mode	37
2.4.1 Comparison of the advantages and disadvantages of the two modes	38
2.4.2 Multi-address boot mode flash allocation	38
2.4.3 Bootloader boot mode flash allocation	40
2.5 Flash Allocation Description	41
2.6 Firmware burning	42
3 Software Architecture	44
3.1 Directory description	44
3.1.1 Hardware platform directory	44
3.1.2 Common function directory	45
3.1.3 Zigbee protocol stack directory	45
3.1.4 Application layer directory	46
3.1.5 Project compiling directory	47
3.2 Abstract layer driver	47
3.2.1 Platform initialization	47
3.2.2 Radio Frequency (RF)	47
3.2.3 GPIO	49
3.2.4 UART	50

3.2.5	ADC	51
3.2.6	PWM	51
3.2.7	Timer	52
3.2.8	Watchdog	52
3.2.9	System Tick	53
3.2.10	Voltage detection	53
3.2.11	Sleep and wake-up	54
3.3	Memory management	55
3.3.1	Dynamic memory management	55
3.3.2	NV Management	55
3.4	Task management	57
3.4.1	Single task queue	57
3.4.2	Standing task queue	57
3.4.3	Software timer task	58
3.4.3.1	Interface functions	58
3.4.3.2	Attentions	58
3.4.3.3	Examples	58
4	Zigbee SDK Application Development	60
4.1	Operation mode selection	60
4.2	Hardware selection	60
4.2.1	Chip model confirmation	60
4.2.1.1	comm_cfg.h chip type definition	60
4.2.1.2	version_cfg.h chip type selection	60
4.2.2	Target board selection	61
4.3	Print debug configuration	62
4.3.1	UART print	62
4.3.2	USB print	62
4.4	Zigbee network development process	63
4.4.1	Application layer initialization (user_init)	63
4.4.2	BDB process	65
4.4.2.1	BDB initialization (bdb_init)	66
4.4.2.2	BDB Commissioning	67
4.4.3	Network security	67
4.4.3.1	Network access process	67
4.4.3.2	Default TCLK	69
4.4.3.3	Install Code	70
4.4.3.4	Pre-configured NWK Key	70
4.4.4	Data interaction	70
4.4.4.1	Data interaction using ZCL layer	70
4.4.4.2	Data interaction using AF layer	71
4.4.5	Network parameters configuration	72
4.4.6	System exception processing	72
5	Commonly Used APIs	74
5.1	Application Framework (AF)	74
5.1.1	af_endpointRegister()	74
5.1.2	af_dataSend()	74

5.2	Base Device Behaviour (BDB)	75
5.2.1	bdb_init()	75
5.2.2	bdb_networkFormationStart()	75
5.2.3	bdb_networkSteerStart()	76
5.2.4	bdb_networkTouchLinkStart()	76
5.2.5	bdb_findAndBindStart()	76
5.2.6	bdb_defaultReportingCfg()	77
5.3	Network management	77
5.3.1	zb_init()	77
5.3.2	zb_zdoCbRegister()	77
5.3.3	zb_setPollRate()	78
5.3.4	zb_rejoinReq()	78
5.3.5	zb_factoryReset()	79
5.3.6	zb_mgmtPermitJoinReq()	79
5.3.7	zb_mgmtLeaveReq()	79
5.3.8	zb_mgmtLqiReq()	80
5.3.9	zb_mgmtBindReq()	81
5.3.10	zb_mgmtNwkUpdateReq()	81
5.3.11	zb_zdoBindUnbindReq()	82
5.3.12	zb_zdoNwkAddrReq()	82
5.3.13	zb_zdoIeeeAddrReq()	83
5.3.14	zb_zdoSimpleDescReq()	83
5.3.15	zb_zdoNodeDescReq()	84
5.3.16	zb_zdoPowerDescReq()	84
5.3.17	zb_zdoActiveEpReq()	85
5.3.18	zb_zdoMatchDescReq()	85
5.3.19	zb_isDeviceFactoryNew()	86
5.3.20	zb_isDeviceJoinedNwk()	86
5.3.21	zb_getMacAssocPermit()	86
5.3.22	zb_apsExtPanidSet()	87
5.4	ZCL	87
5.4.1	zcl_init()	87
5.4.2	zcl_register()	87
5.5	OTA	88
5.5.1	ota_init()	88
5.5.2	ota_queryStart()	88
6	OTA	90
6.1	OTA initialization	90
6.2	OTA Server	90
6.3	OTA Client	90
6.4	OTA Image	91
7	Bootloader	93
7.1	Application Upgrade	93
8	Extended Function HCI Interface	95
8.1	Control flow chart	95
8.2	Command frame format	95

8.3	Acknowledge format	96
8.3.1	Message Type	96
8.3.2	Payload	96
8.4	BDB commands	97
8.4.1	Message Type	97
8.4.2	Payload	98
8.5	Network management command	99
8.5.1	Message Type (Host)	99
8.5.2	Payload (Host)	100
8.5.3	Message Type (Slave)	107
8.5.4	Payload (Slave)	107
8.6	ZCL Cluster commands	115
8.6.1	ZCL command header format	115
8.6.2	General cluster command	116
8.6.2.1	Message Type (Host)	116
8.6.2.2	Payload (Host)	117
8.6.2.3	Message Type (Slave)	119
8.6.2.4	Payload (Slave)	119
8.6.3	Basic cluster command	123
8.6.3.1	Message Type (Host)	123
8.6.3.2	Payload (Host)	123
8.6.4	Group cluster command	123
8.6.4.1	Message Type (Host)	123
8.6.4.2	Payload (Host)	124
8.6.4.3	Message Type (Slave)	126
8.6.4.4	Payload (Slave)	126
8.6.5	Identify cluster command	128
8.6.5.1	Message Type (Host)	128
8.6.5.2	Payload (Host)	128
8.6.5.3	Message Type (Slave)	128
8.6.5.4	Payload (Slave)	129
8.6.6	On/Off cluster command	129
8.6.6.1	Message Type (Host)	129
8.6.6.2	Payload (Host)	129
8.6.7	Level cluster command	130
8.6.7.1	Message Type (Host)	130
8.6.7.2	Payload (Host)	130
8.6.8	Scene cluster command	133
8.6.8.1	Message Type (Host)	133
8.6.8.2	Payload (Host)	133
8.6.8.3	Message Type (Slave)	136
8.6.8.4	Payload (Slave)	136
8.6.9	OTA cluster command	139
8.6.9.1	Message Type (Host)	139
8.6.9.2	Payload (Host)	139
8.7	HCI serial port upgrade command	140
8.7.1	Message Type (Host)	140

8.7.2	Payload (Host)	140
8.7.3	Message Type (Slave)	141
8.7.4	Payload (Slave)	141
9	Appendix: Zigbee Alliance Pro R21 Certification	143

Telink Semiconductor

List of Figures

Figure 1.1	Mesh network	11
Figure 1.2	The relationship chart between Endpoint, Cluster and Node	13
Figure 2.1	Select import projects	16
Figure 2.2	Complete project import	17
Figure 2.3	Project directory structure	18
Figure 2.4	Select and compile sample project	19
Figure 2.5	Output file	20
Figure 2.6	Adding a new project	21
Figure 2.7	Configuring a new project	22
Figure 2.8	Symbol definition	23
Figure 2.9	Library files and paths	24
Figure 2.10	Link file pre-compiling settings	25
Figure 2.11	image check settings	26
Figure 2.12	Select import project file	27
Figure 2.13	Complete project import	28
Figure 2.14	Project directory structure	29
Figure 2.15	Select and compile	30
Figure 2.16	Output file	30
Figure 2.17	Adding a new project	31
Figure 2.18	Configuring a new project	32
Figure 2.19	Symbol definition	33
Figure 2.20	Library files and paths	34
Figure 2.21	Link file pre-compiling settings	35
Figure 2.22	image check settings	36
Figure 2.23	Binary file of image	37
Figure 2.24	512KB Flash space allocation chart	38
Figure 2.25	1MB Flash space allocation chart	39
Figure 2.26	512KB Flash space allocation chart	40
Figure 2.27	1MB Flash space allocation chart	41
Figure 2.28	Burning connection	42
Figure 2.29	Burning tool	43
Figure 3.1	Hardware platform directory	44
Figure 3.2	Common functions directory	45
Figure 3.3	Zigbee protocol stack directory	45
Figure 3.4	Application layer directory	46
Figure 3.5	Project compiling directory	47
Figure 4.1	Application layer initialization flow	64
Figure 4.2	BDB initialization flow	66
Figure 4.3	Direct access to the network	68
Figure 4.4	Indirect access to the network	69
Figure 6.1	File import	92
Figure 7.1	Bootloader upgrade	94
Figure 8.1	Control flow chart	95
Figure 8.2	Command frame format	97

Figure 9.1 Zigbee Alliance	143
--------------------------------------	-----

Telink Semiconductor

1 Overview

1.1 Zigbee brief

1.1.1 Device type

Zigbee 3.0 supports the following device types.

- **Router:** a device with routing functions
- **Coordinator:** Router with network management capabilities
- **End Device:** A device that can support low-power hibernation mode, but without routing function, and should interact with other nodes in the network through its parent node.
- **GPD (Green Power Device):** Low-power passive device. Not introduced in this document.

1.1.2 Network type

Zigbee 3.0 network belongs to Mesh network, which can be categorized into two types, Central and Distribute network, depending on the creator, which differ in the maintenance of link key and nwk key in the network.

- **Central network:** A network that is usually formed and managed by a Coordinator as a trust center.
- **Distribute network:** A network which is formed by nodes with Router function.

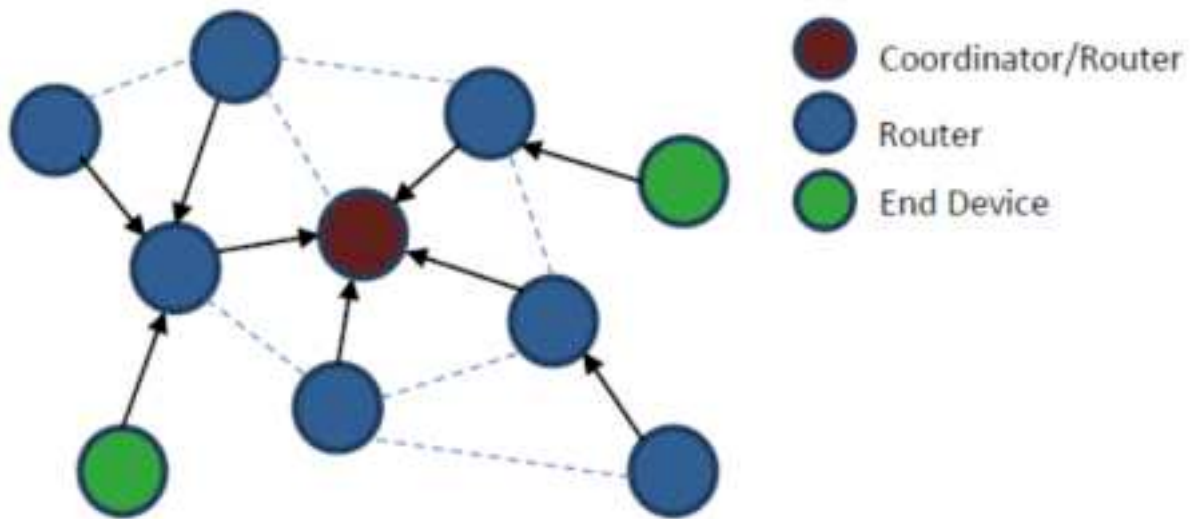


Figure 1.1: Mesh network

1.1.3 Basic concept

1) **PAN ID:** Personal Area Network ID.

The PAN ID is to identify the formed network. The PAN ID is assigned by the node responsible for forming the network, and can be assigned directly or generated randomly, but active scan is required to avoid PAN ID conflicts.

2) **Channel:** frequency point/channel.

The operating frequency points permitted by Zigbee to work at (2.4G): $2405 + (N-11) \times 5$ (mHz), (channel N = 11~26).

Once connected to a network, it uses a single frequency point of operation mode, will not actively jump frequency.

3) **Node:** a physical device.

Each node has a unique 64-bit MAC address, and after forming or joining in a network, it has a unique 16-bit short network address in the PAN. The short address can be directly used in subsequent data transfer.

4) **Endpoint:** port, corresponding to specific applications (Apps).

A Node can contain multiple applications (multiple ports). The applications in user layer are all based on the operation of different ports.

Port number range: 0~255. The Zigbee Alliance has specific rules for the use of port number.

- 0: Used for ZDO (Zigbee Device Object), which is mandatory for every Node.
- 1~240: User-level application use, that is, the port that can be randomly used in application development.
- 241~254: Some applications specified by the Zigbee Alliance, such as 242 for Green Power only.
- 255: Broadcast port, some data is to be broadcast to all application ports.

5) **Clusters:**

A specific application (port) is composed of a set of different clusters that perform different behaviors. ZCL (Zigbee Cluster Library) defines a set of behavioral rules for the clusters.

6) **Attributes:**

ZCL defines the attributes supported by each cluster and also defines the permissions to operate on the values of these attributes.

7) **BDB:** Basic Device Behavior

It regulates the behavior of node startup, network forming, network entry, key management and reset.

In essence, the operation object of the application layer is an attribute of a Cluster on a certain port (Endpoint). The corresponding relationship is as below.

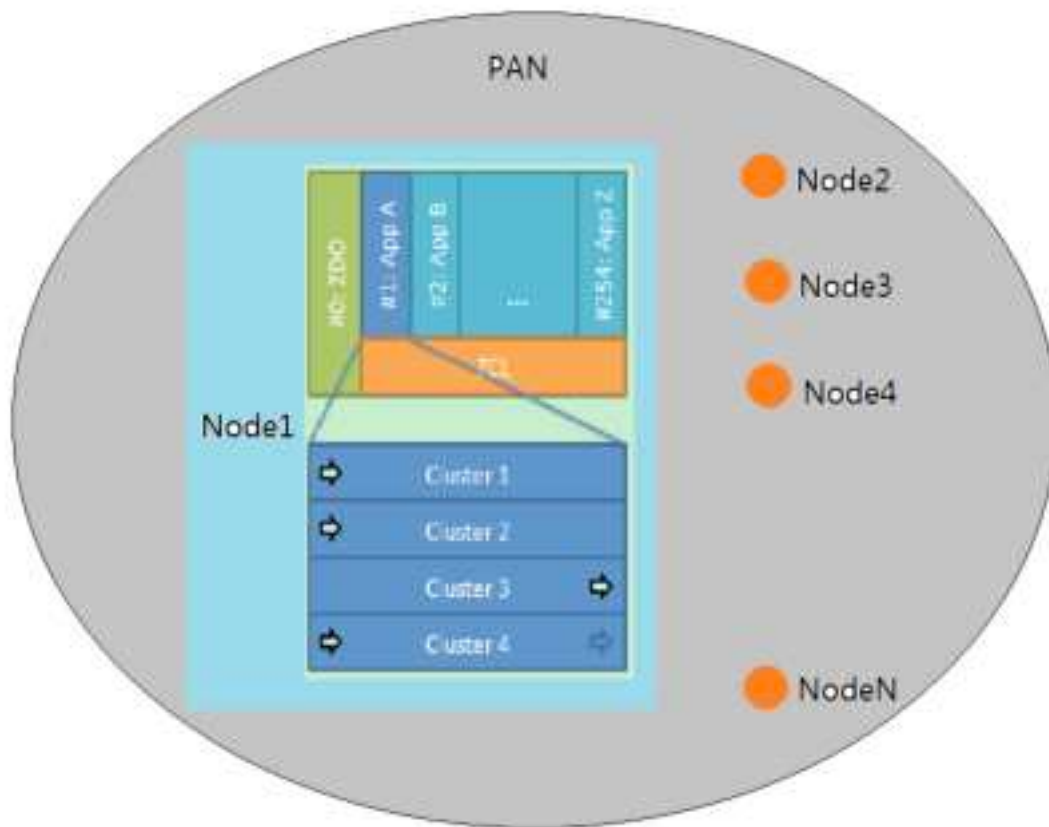


Figure 1.2: The relationship chart between Endpoint, Cluster and Node

1.2 Telink Zigbee SDK brief

Telink Zigbee SDK is a set of Zigbee protocol stack developed on the basis of Zigbee PRO specification, and it has been certified by the Alliance platform Zigbee Pro R21, supports Pro R22 and complies with Zigbee 3.0 application specification.

1.2.1 Software development environment for Telink Zigbee SDK

1) **Essential software tools** (download at: <http://wiki.telink-semi.cn/>)

- Integrated development environment.
 TSLR8 Chips: Telink IDE for TC32
 TSLR9 Chips: Telink RDS IDE for RISC-V
- Download debugging tools: Telink Burning and Debugging Tools
- OTA code conversion tool: tl_ota_tool

2) **Auxiliary tool to capture and analyze packet** (download or purchase as you need)

- TI Packet Sniffer

- Ubiqua

3) **Software development kit** (download at: <http://wiki.telink-semi.cn/>)

- tl_zigbee_sdk_v3.x.x.zip

4) **Auxiliary control software on PC (ZGC)**

- Zigbee_gateway_controller.exe

1.2.2 Hardware platforms supported by Telink Zigbee SDK

- B85m (TC32 platform)
8269: 8269 EVK Board and 8269 USB Dongle
8258: 8258 EVK Board and 8258 USB Dongle
8278: 8278 EVK Board and 8278 USB Dongle
- B91m (RISC-V platform)
B91 series: B91 EVK Board and B91 USB Dongle

Telink Semiconductor

2 Telink Zigbee SDK

Before installing the SDK, please install the appropriate Telink IDE for TC32 or Telink RDS IDE for RISC-V according to [\(section 1.2.1.\)](#)

2.1 TLSR8 TC32 SDK Installation

2.1.1 Project Import

- 1) Start the IDE and go to the interface File -> Import -> Existing Projects into Workspace in order.
- 2) Select tlr_tc32 in the tl_zigbee_sdk/build directory -> click "OK".

Telink Semiconductor

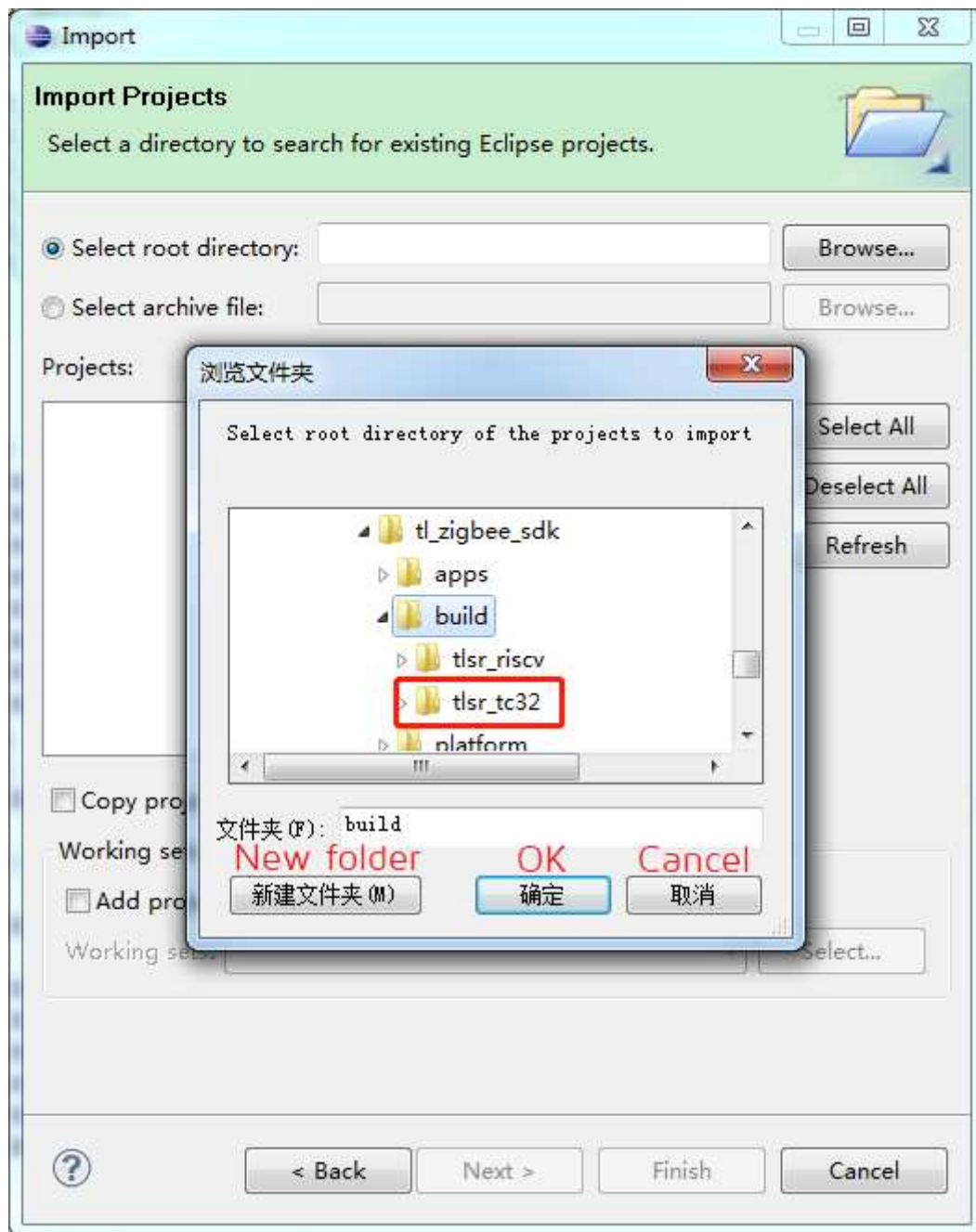


Figure 2.1: Select import projects

3) Click "Finish" to complete the project import.

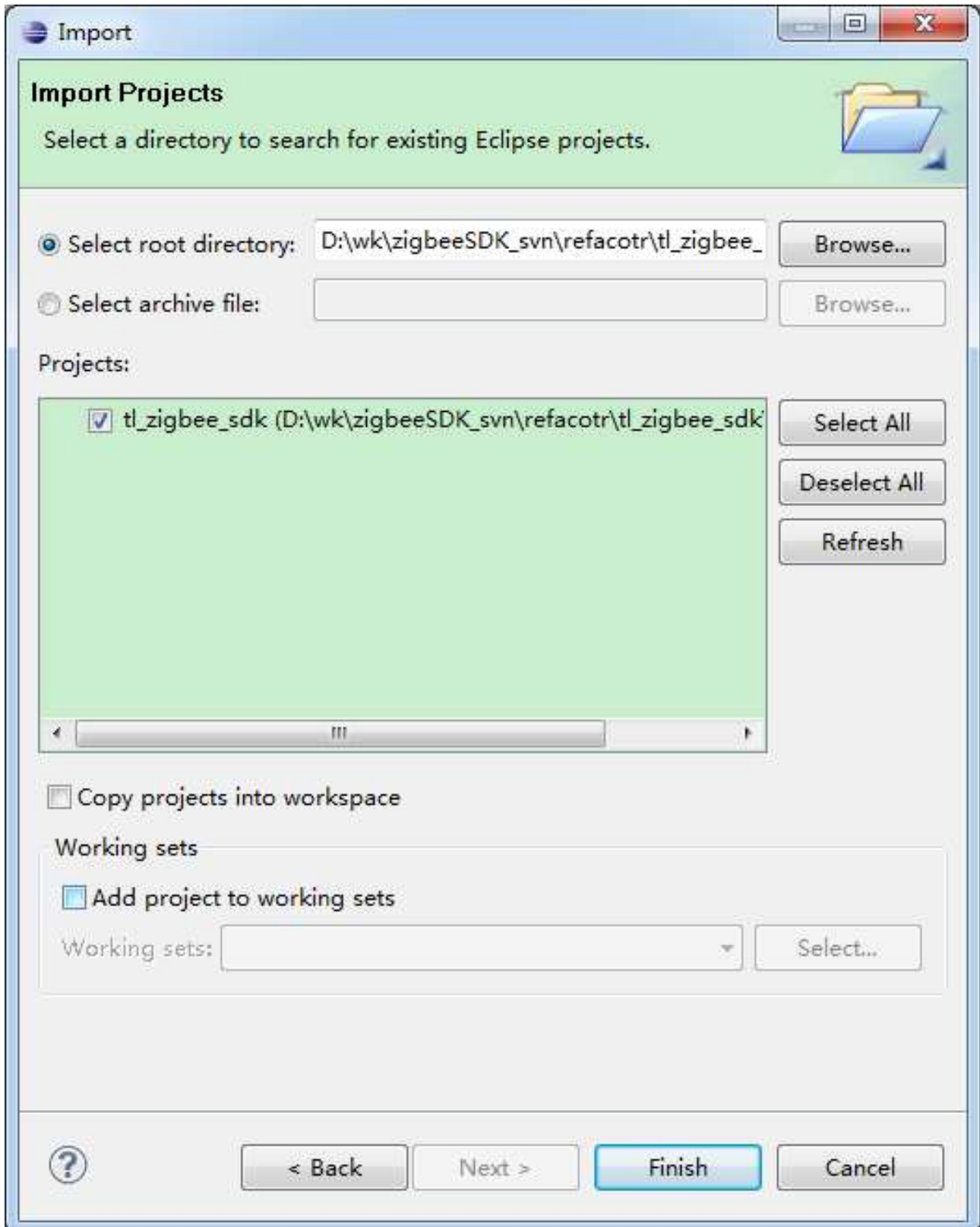


Figure 2.2: Complete project import

2.1.2 Project directory structure for TLSR8

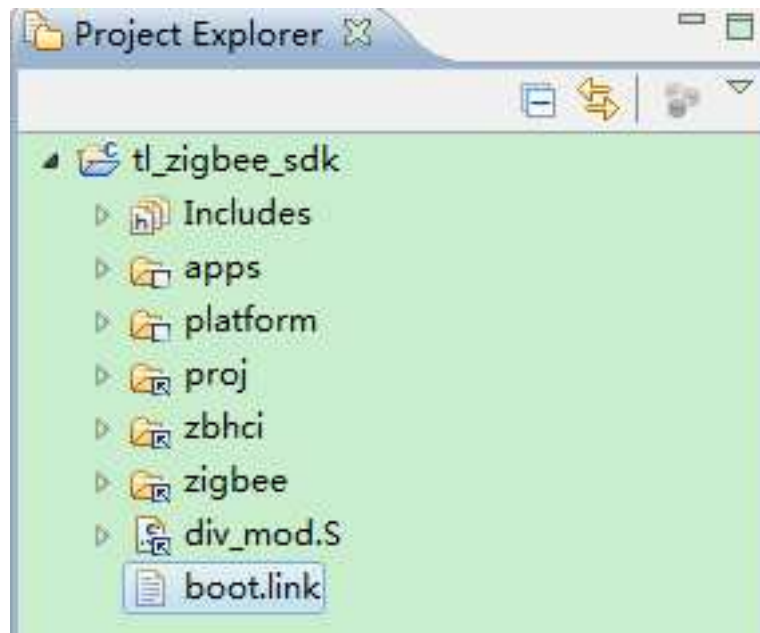
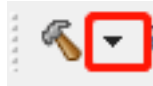
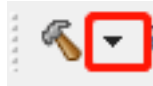


Figure 2.3: Project directory structure

- **/apps**: user project directory.
 - /common: application-level public code directory, including main functions (main.c), module test functions (module_test.c), and version and mode configuration (comm_cfg.h), etc.
 - /sampleGW: Gateway (Coordinator) samples.
 - /sampleLight: Light (Router) samples.
 - /sampleSwitch: Switch (End Device) samples.
 - /bootLoader: Bootloader.
- **/platform**: operation platform directory.
 - /boot: boot and link files.
 - /chip_xx: chip driver files.
 - /services: interrupt service function files.
- **/proj**: project code directory.
 - /common: common code directory.
 - /drivers: abstraction layer driver file.
 - /os: task events, buffer management function files.
- **/zigbee**: protocol stack related directory.
- **/zbhci**: hci command processing related directory.
- **div_mod.S**: division and remainder related assembly functions. (TLSR8 does not support hardware dividers)

2.1.3 Compile options



Click the drop-down icon , you can see all the current compiling options, select the sample project you need to compile as below, and wait for the compiling to complete.

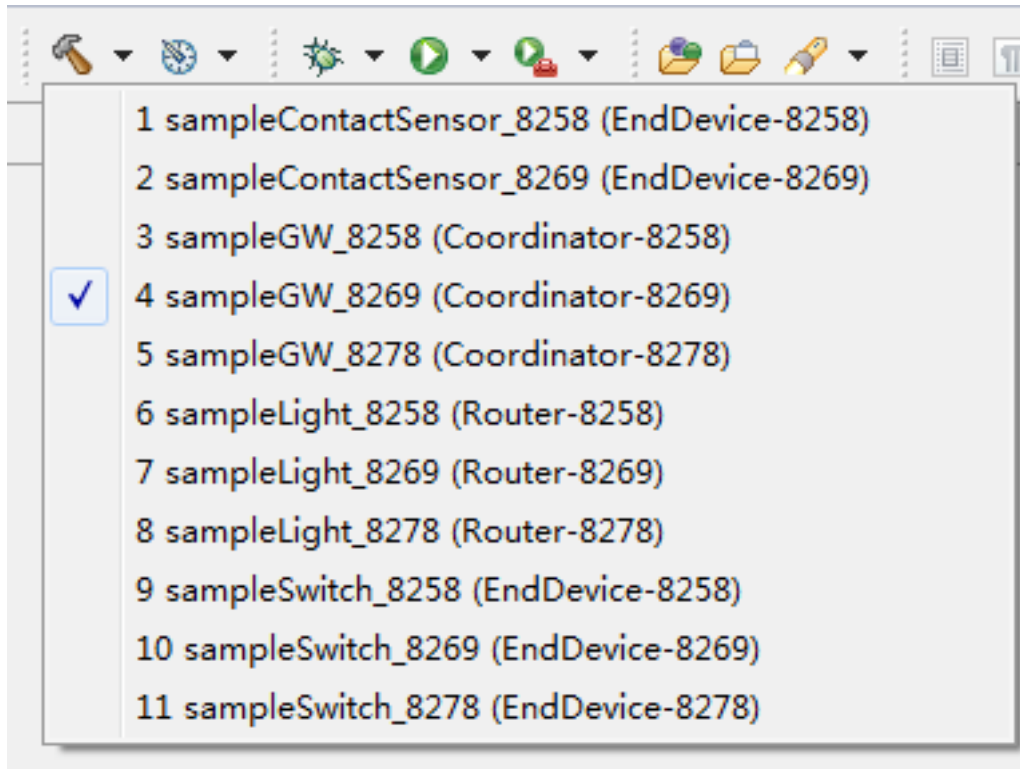


Figure 2.4: Select and compile sample project

After the compiling is completed, the compiled folder will appear in the "Project Explorer" window, which contains the compiled firmware, as shown below.

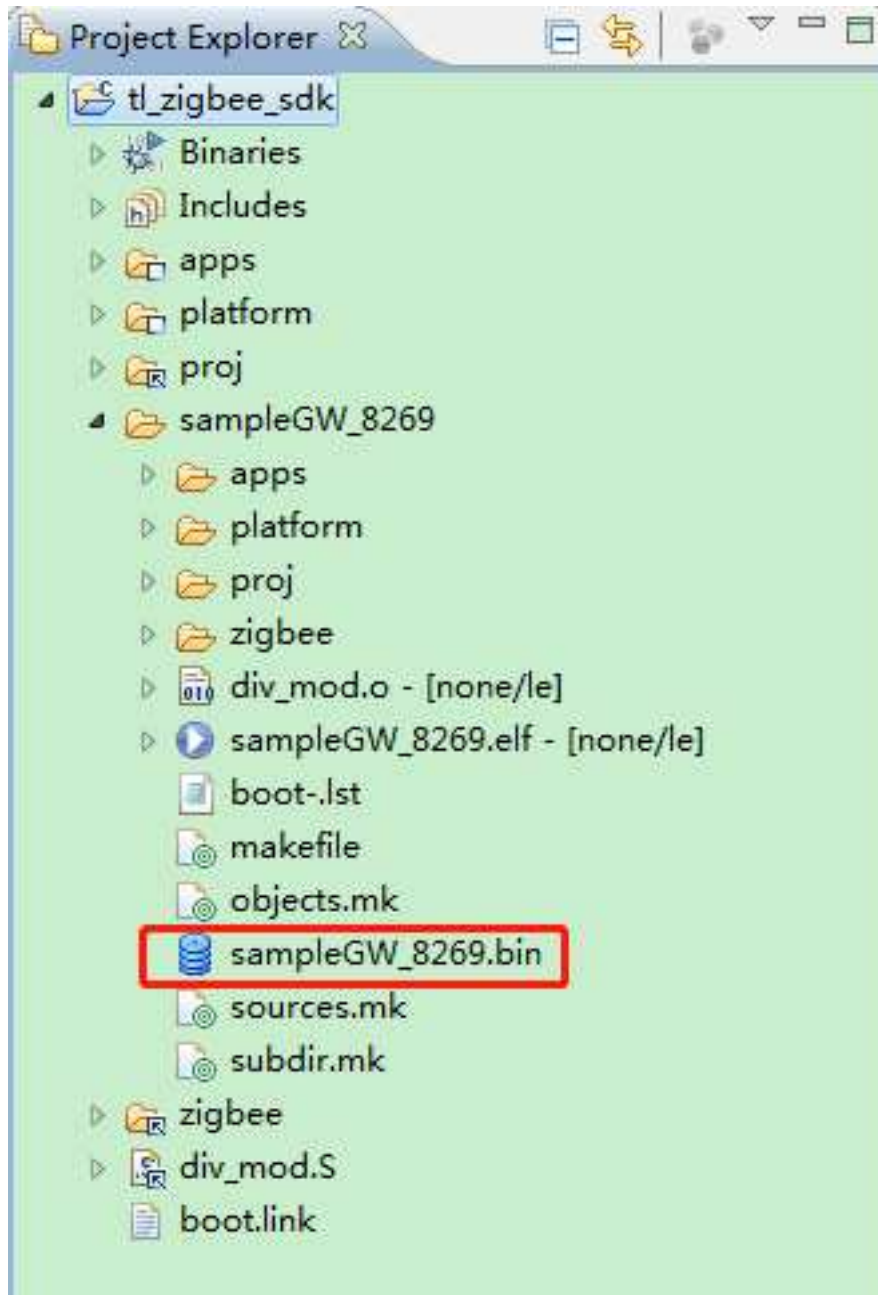


Figure 2.5: Output file

2.1.4 Add new project

In the provided SDK, only some simple use cases are listed. Users can add their own application projects and compiling options according to their needs.

The detailed steps are as follows.

- 1) Step 1: Project Explorer -> tl_zigbee_sdk -> Properties -> Manage Configurations

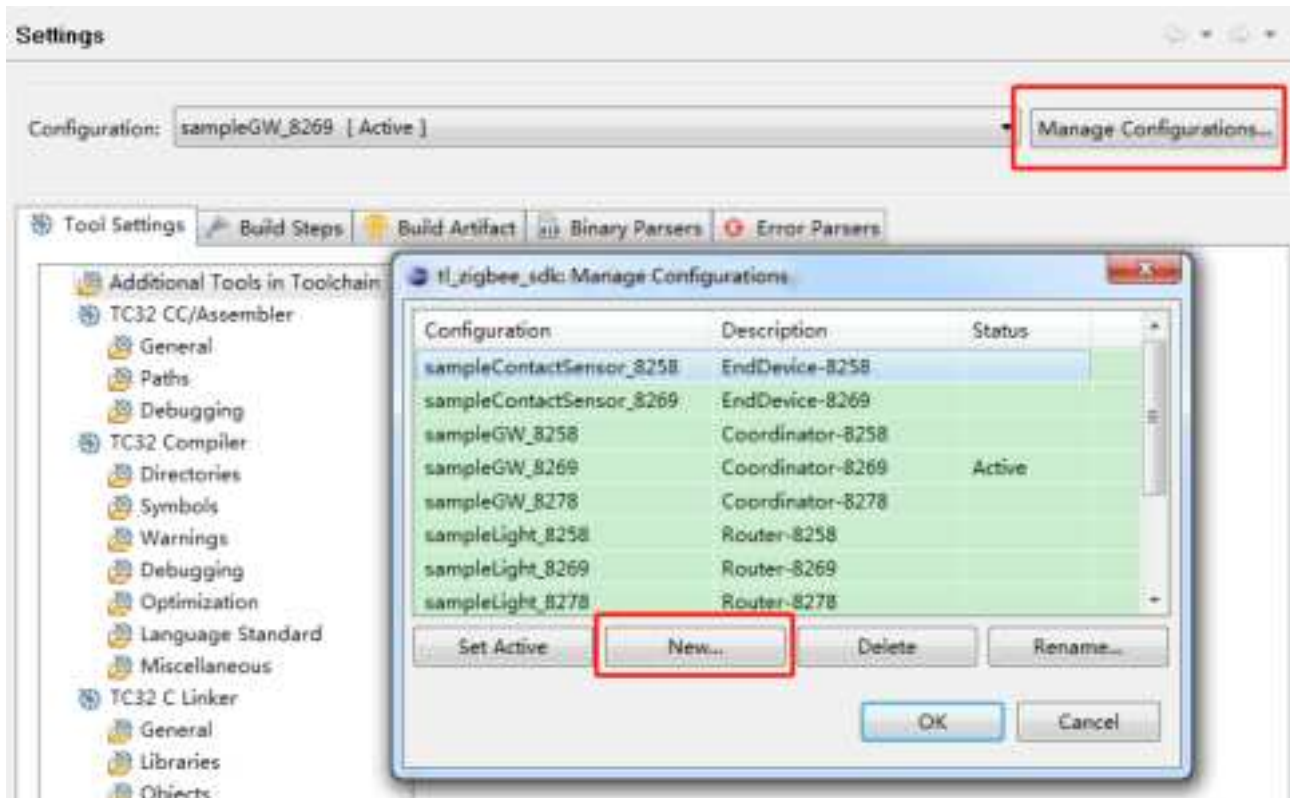


Figure 2.6: Adding a new project

2) Step 2: Click New, the following window will pop up.

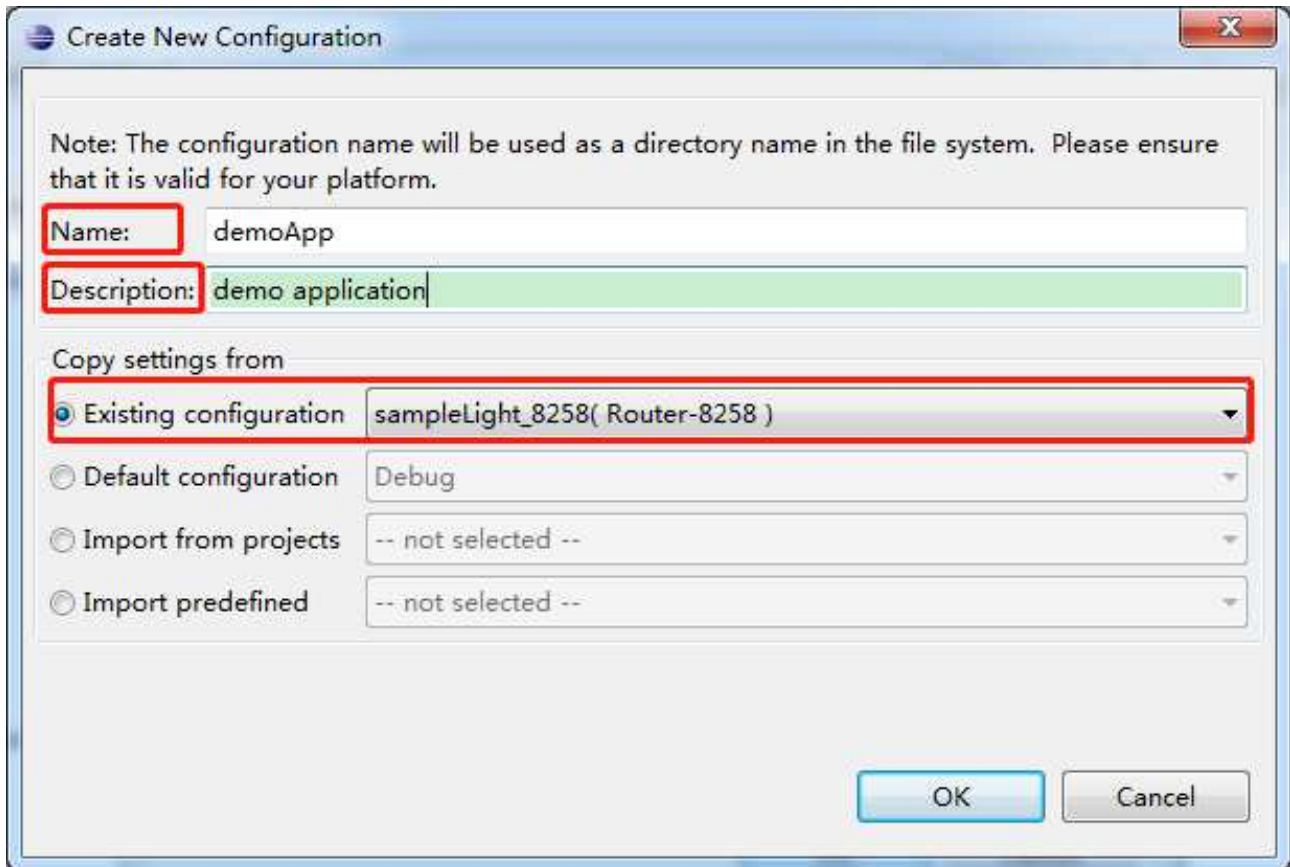


Figure 2.7: Configuring a new project

- Name: project name
- Description: brief project description
- Copy settings from: It is recommended to choose Existing configuration, which can simplify the configuration process.

The principle of selecting Existing configuration is as below.

- If using 8269 platform, select xxx_8269; if using 8258 platform, select xxx_8258.
- Project for Gateway development, select sampleGw_xxxx; project for Router device development, select sampleLight_xxxx; project for End Device device development, select sampleSwitch_xxxx.

For example, suppose you need to develop a light project with routing function based on 8258, according to this principle, you should choose the configuration of the project "sampleLight_8258" as the configuration of this new project.

If you need to modify the configuration, you can follow the next section (2.1.5 Project configuration description) and adjust accordingly.

2.1.5 Project configuration description

Enter in order: Project Explorer -> tl_zigbee_sdk -> Properties -> Settings (take the project sample-Light_8258 for example)

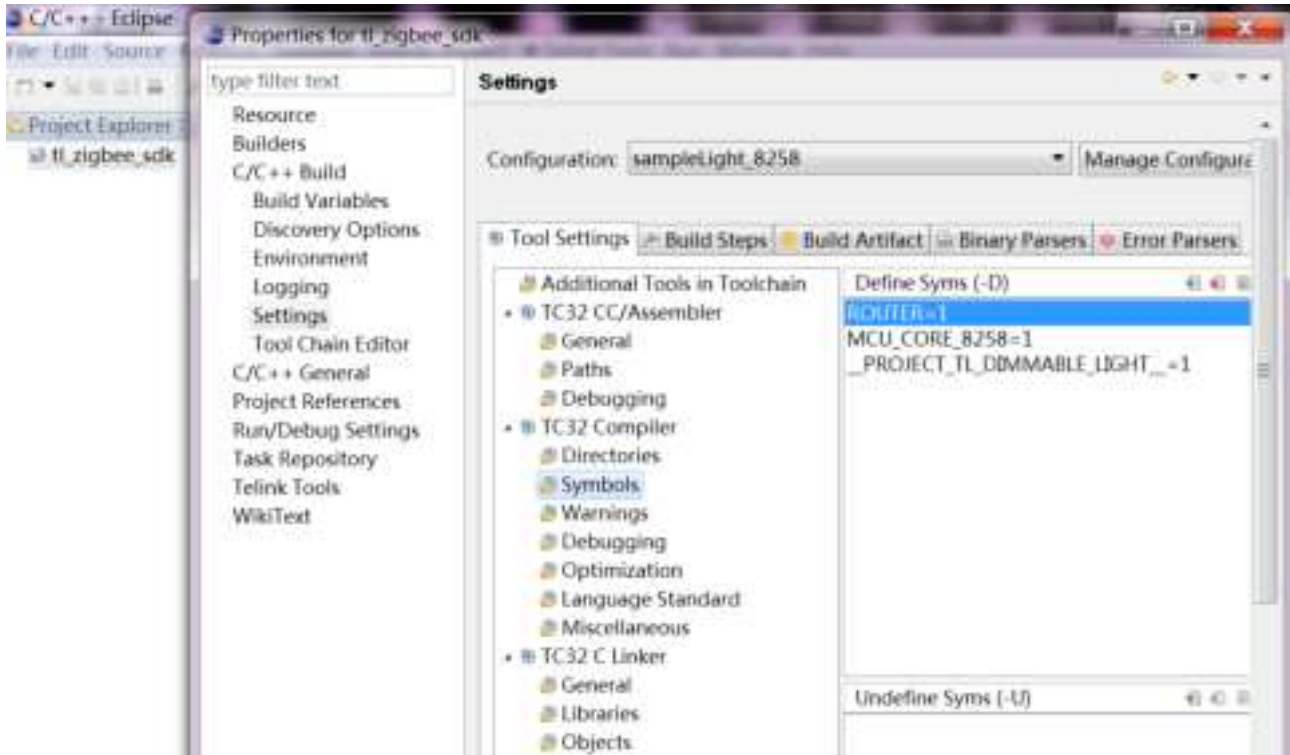


Figure 2.8: Symbol definition

The “Tool Settings” contains some pre-defined configurations for current project:

1) Device type pre-definition

-DROUTER=1: indicates that the project is a device with router function

Following shows the device setting for coordinator and end device:

-DEND_DEVICE=1: indicates that the project is a device with end device function

-DCOORDINATOR=1: indicates that the project is a device with coordinator function

2) Platform selection

- 8269 platform: -DMCU_CORE_826x=1 and -DCHIP_8269=1

The startup code cstartup_826x.S is located in the \tl_zigbee_sdk\platform\boot\826x directory.

- 8258 platform: -DMCU_CORE_8258=1

The startup code cstartup_8258.S is located in the \tl_zigbee_sdk\platform\boot\8258 directory.

- 8278 platform: -DMCU_CORE_8278=1

The startup code cstartup_8278.S is located in the \tl_zigbee_sdk\platform\boot\8278 directory.

3) Links for lib files

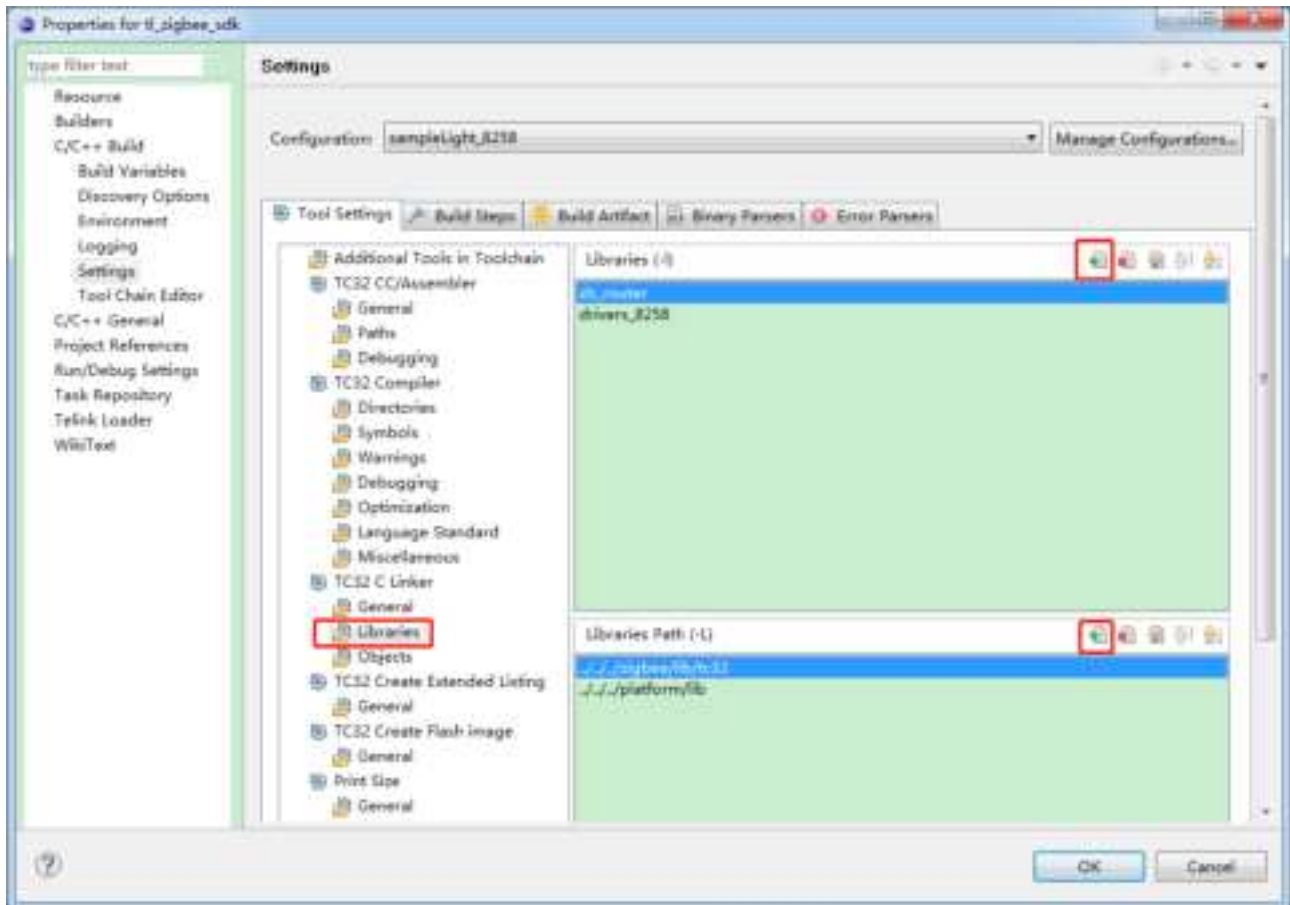


Figure 2.9: Library files and paths

The current SDK includes 2 types of library files: the Zigbee stack library and the platform driver library.

- Zigbee stack library: libzb_router.a, libzb_coordinator.a, libzb_ed.a
Available in the \tl_zigbee_sdk\zigbee\lib\tc32 directory.
- Platform driver library: libdrivers_826x.a, libdrivers_8258.a, libdrivers_8278.a
Available in the \tl_zigbee_sdk\platform\lib directory.

4) Link file

According to the actual application requirements, the user can adjust the appropriate link file as per the different platforms chosen and the memory requirements.

The current SDK provides the default link files boot_826x.link, boot_8258.link and boot_8278.link for the 826x, 8258 and 8278 platforms, which are available in the corresponding directories of \tl_zigbee_sdk\platform\boot.

As shown in the figure below, the link file to be used can be selected by invoking the script "tl_link_load.sh" (in the directory of \tl_zigbee_sdk\tools) during the pre-compiling.

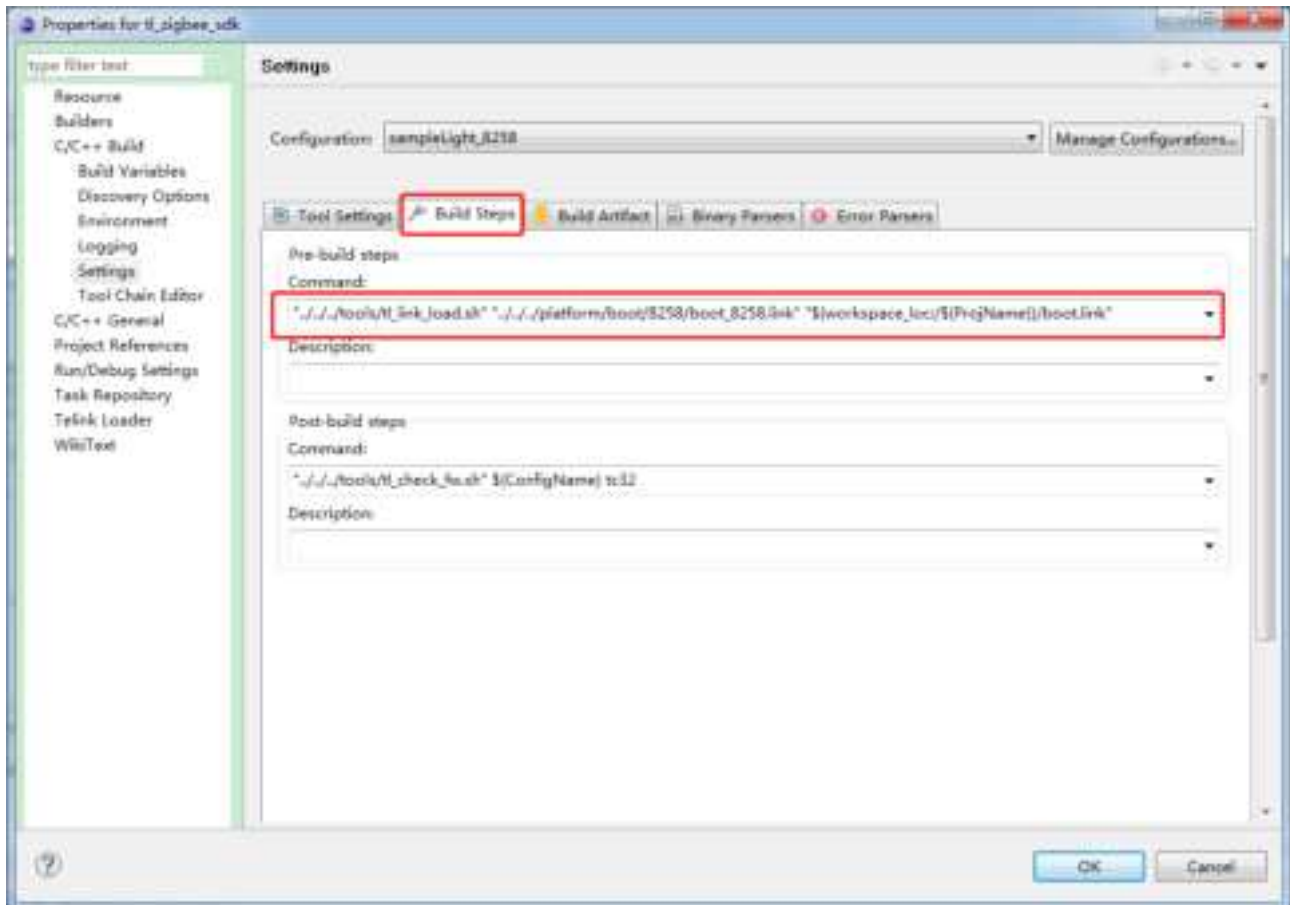


Figure 2.10: Link file pre-compiling settings

5) .image file check

In order to ensure the reliability of the downloaded file, a check field is added to the generated image file by the script `tl_check_fw.sh` (in the directory of `\tl_zigbee_sdk\tools`), and after the OTA process it will determine whether to run the new image by checking whether the check field matches or not.

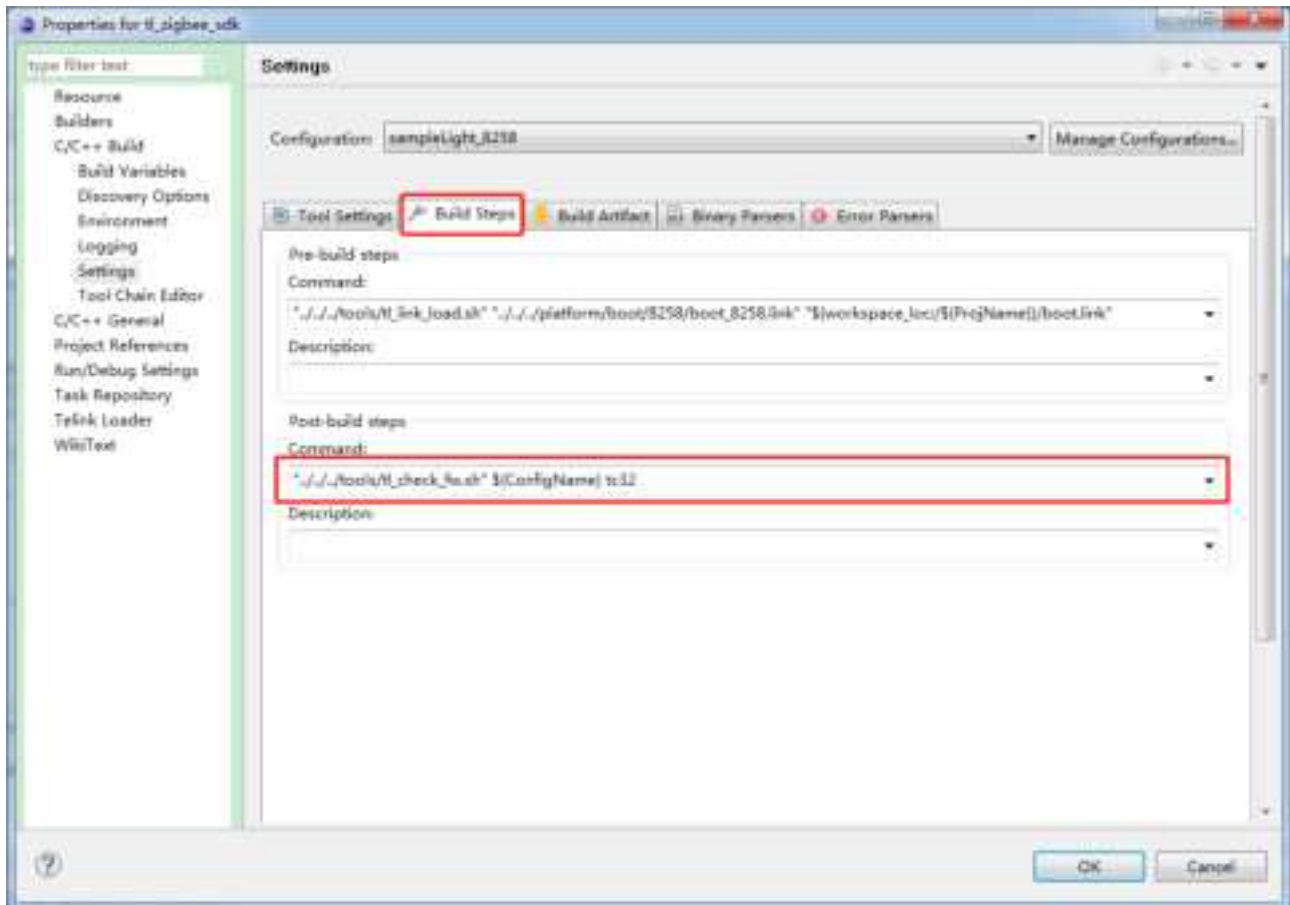


Figure 2.11: image check settings

2.2 TLSR9 RISC-V SDK installation

2.2.1 Project import

- 1) Start the IDE and go to the interface File -> Import -> Existing Projects into Workspace in order.
- 2) Select tlsr_riscv in the tl_zigbee_sdk/build directory -> click "OK".

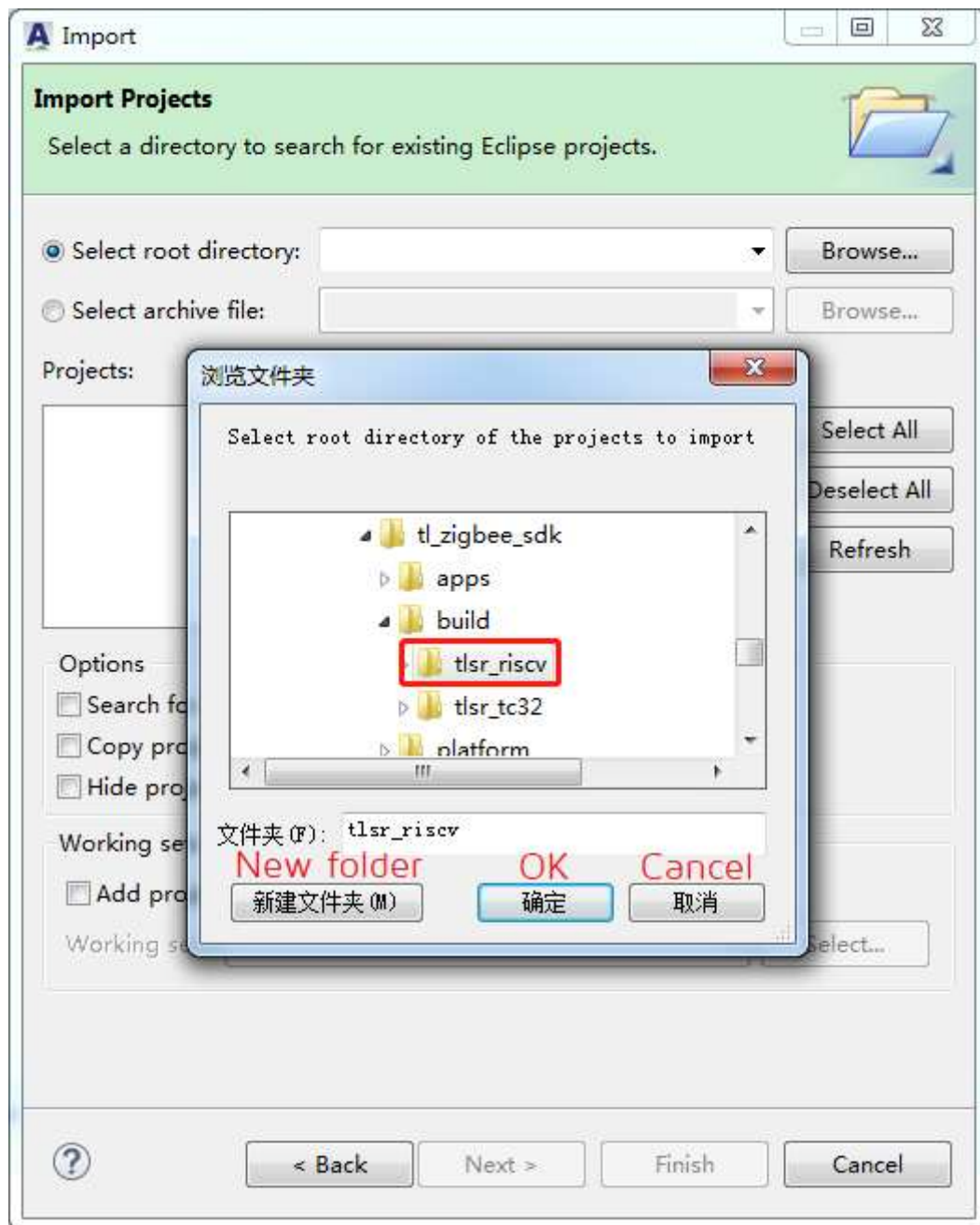


Figure 2.12: Select import project file

3) Click "Finish" to complete the project import.

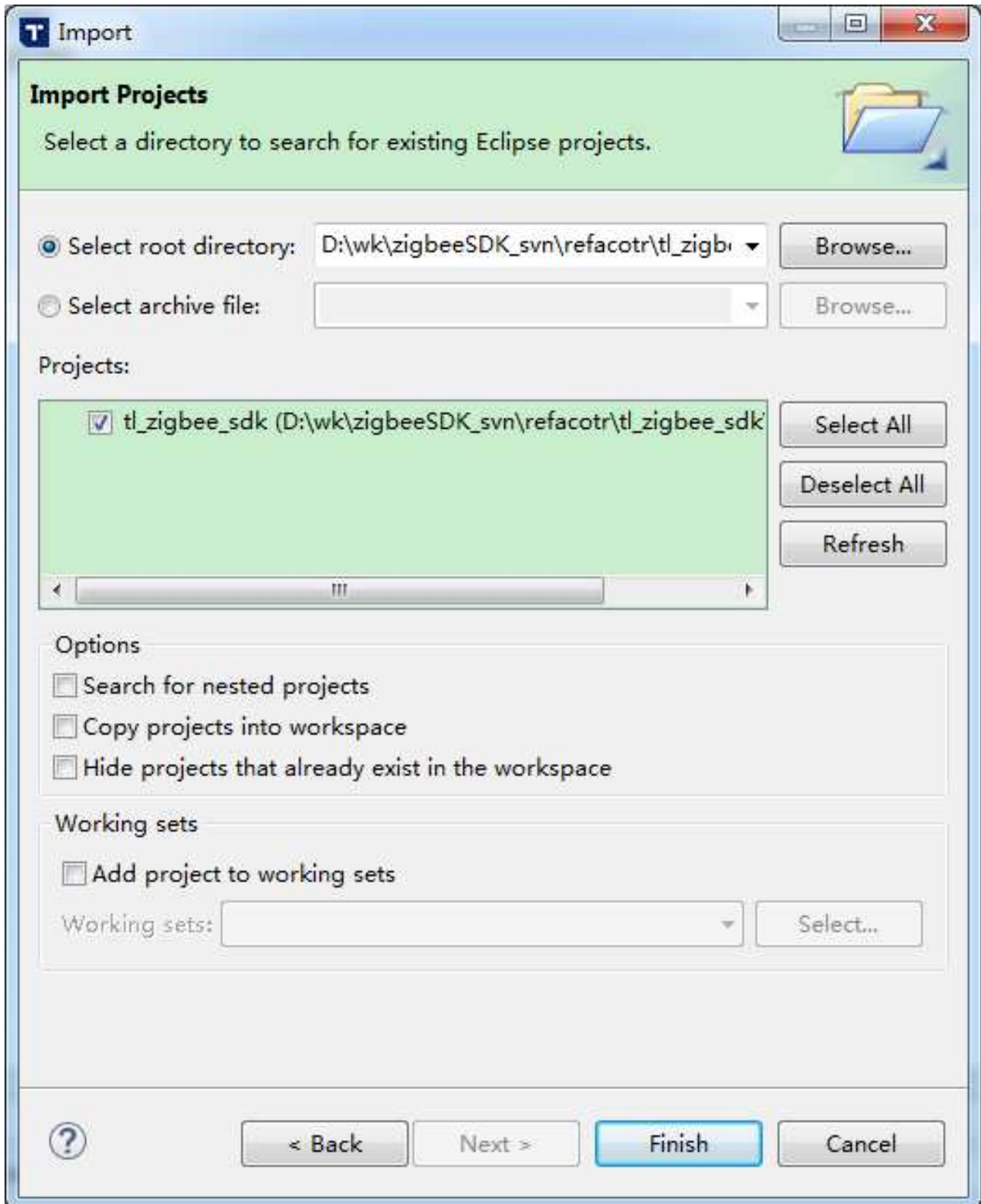


Figure 2.13: Complete project import

2.2.2 Project directory structure for TLSR9

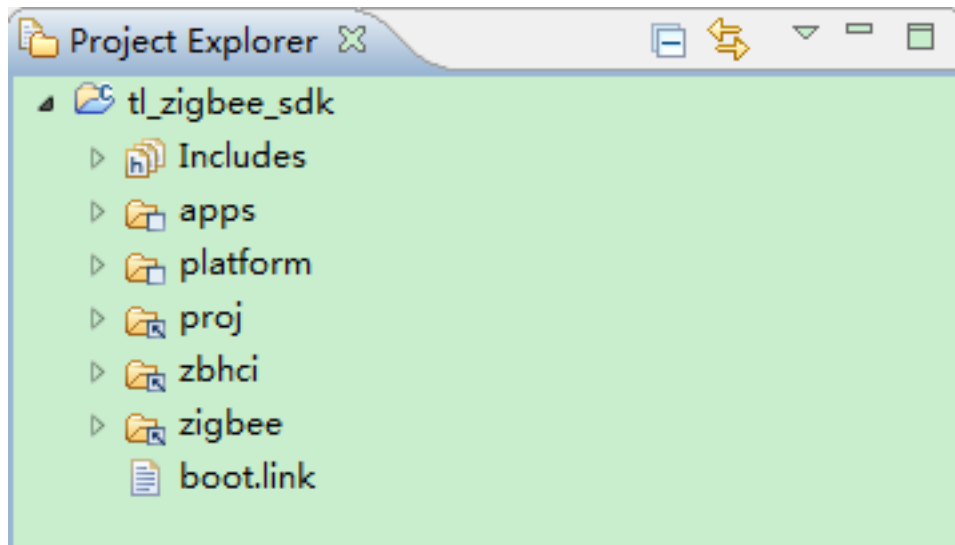


Figure 2.14: Project directory structure

- **/apps**: user project directory.
 - /common: application-level public code directory, including main functions (main.c), module test functions (module_test.c), and version and mode configuration (comm_cfg.h), etc.
 - /sampleGW: Gateway (Coordinator) samples.
 - /sampleLight: Light (Router) samples.
 - /sampleSwitch: Switch (End Device) samples.
 - /bootLoader: Bootloader.
- **/platform**: operation platform directory.
 - /boot: boot and link files.
 - /chip_xx: chip driver file.
 - /services: interrupt service function file.
- **/proj**: project code directory.
 - /common: common code directory.
 - /drivers: abstraction layer driver file.
 - /os: task events, buffer management function files.
- **/zigbee**: protocol stack related directory.
- **/zbhci**: hci command processing related directory.

2.2.3 Compile options



Click the drop-down icon “”, you can see all the current compiling options, select the sample project you need to compile as below, and wait for the compiling to complete.

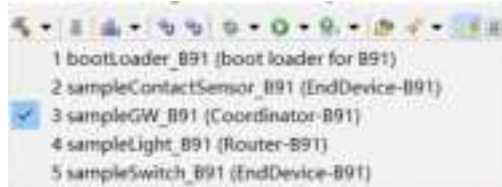


Figure 2.15: Select and compile

After the compiling is completed, the compiled folder will appear in the “Project Explorer” window, which contains the compiled firmware, as shown below.



Figure 2.16: Output file

2.2.4 Add new project

In the provided SDK, only some simple use cases are listed. Users can add their own application projects and compiling options according to their needs.

The detailed steps are as follows.

1) Step 1: Project Explorer -> tl_zigbee_sdk -> Properties -> Manage Configurations

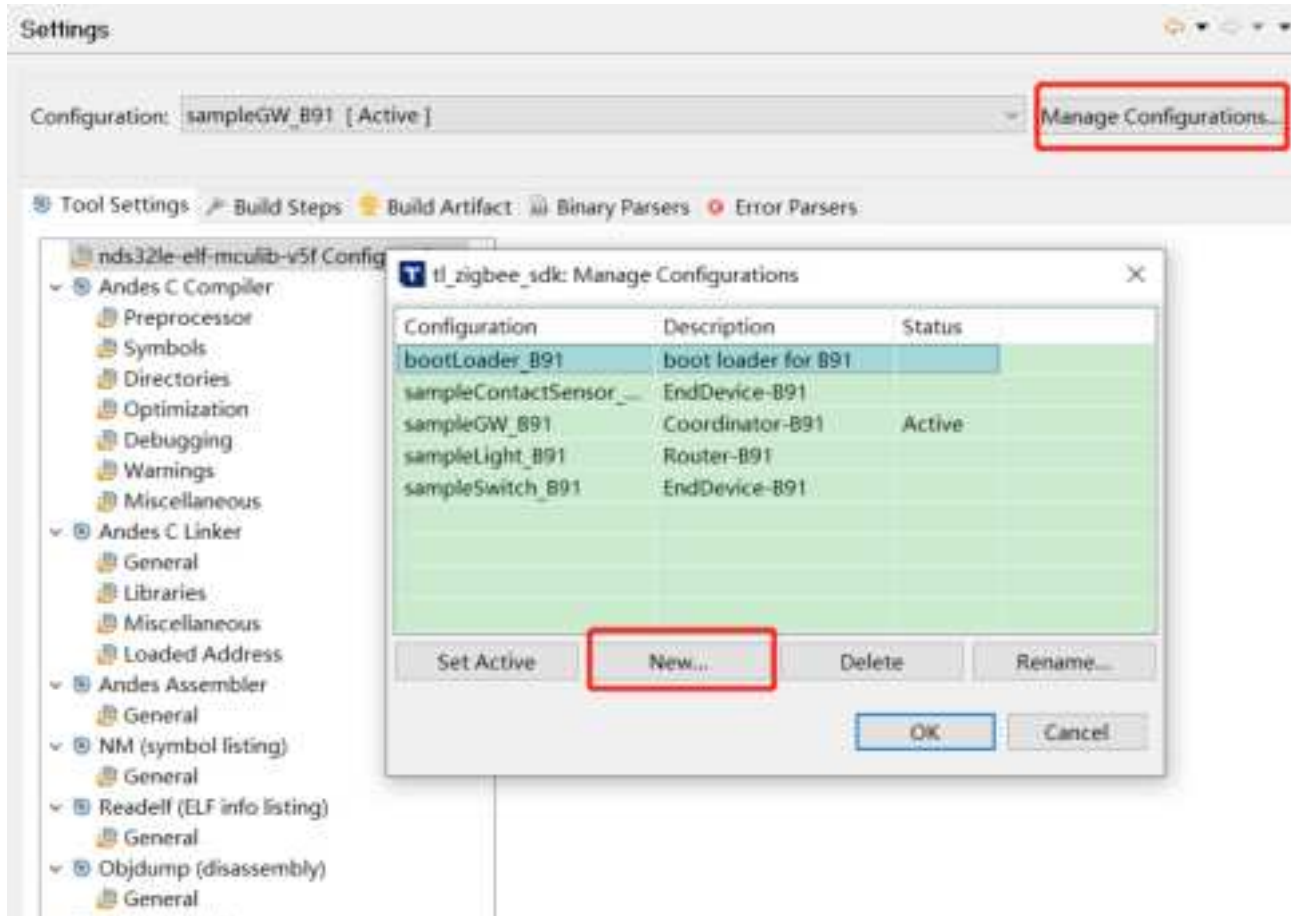


Figure 2.17: Adding a new project

2) Step 2: Click New, the following screen will pop up.

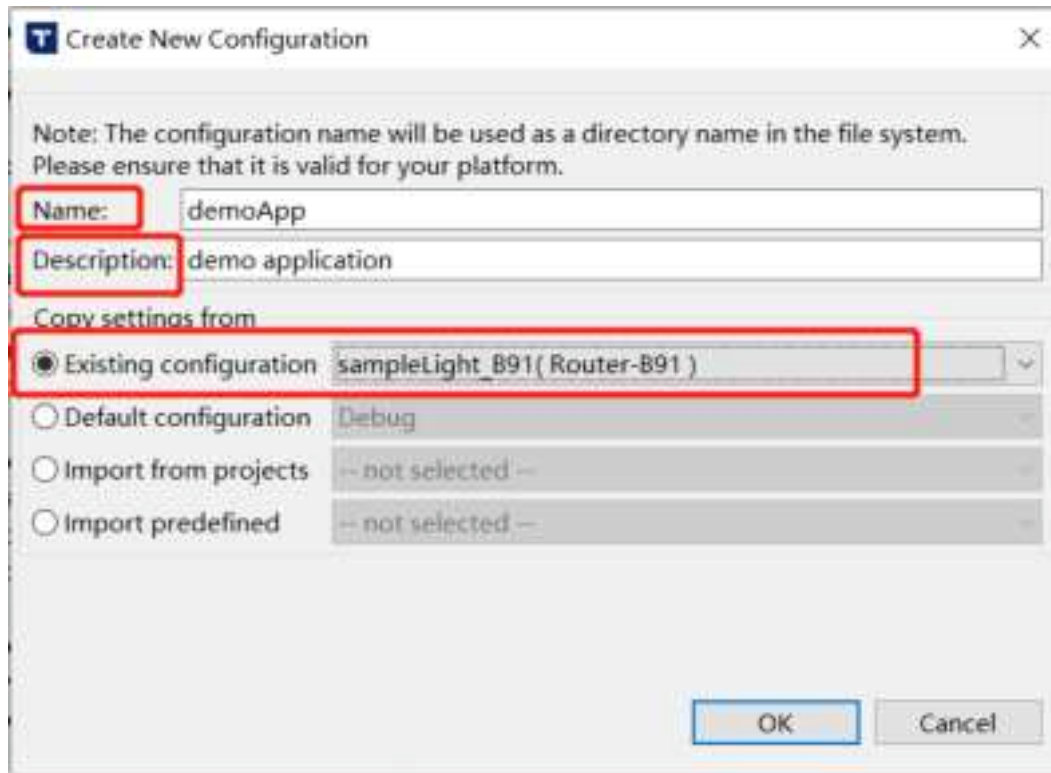


Figure 2.18: Configuring a new project

- Name: project name
- Description: brief project description
- Copy settings from: It is recommended to choose Existing configuration, which can simplify the configuration process.

The principle of selecting Existing configuration is as below.

- If using B91 series chip, select xxx_B91.
- Project for Gateway development, select sampleGw_xxxx; project for Router device development, select sampleLight_xxxx; project for End Device device development, select sampleSwitch_xxxx.

For example, suppose you need to develop a light project with routing function based on B91 series chip, according to this principle, you should choose the configuration of the project "sampleLight_B91" as the configuration of this new project.

If you need to modify the configuration, you can follow the next section (2.2.5 Project configuration description) and adjust it accordingly.

2.2.5 Project configuration description

Enter in order: Project Explorer -> tl_zigbee_sdk -> Properties -> Settings (take the project sampleLight_B91 for example)

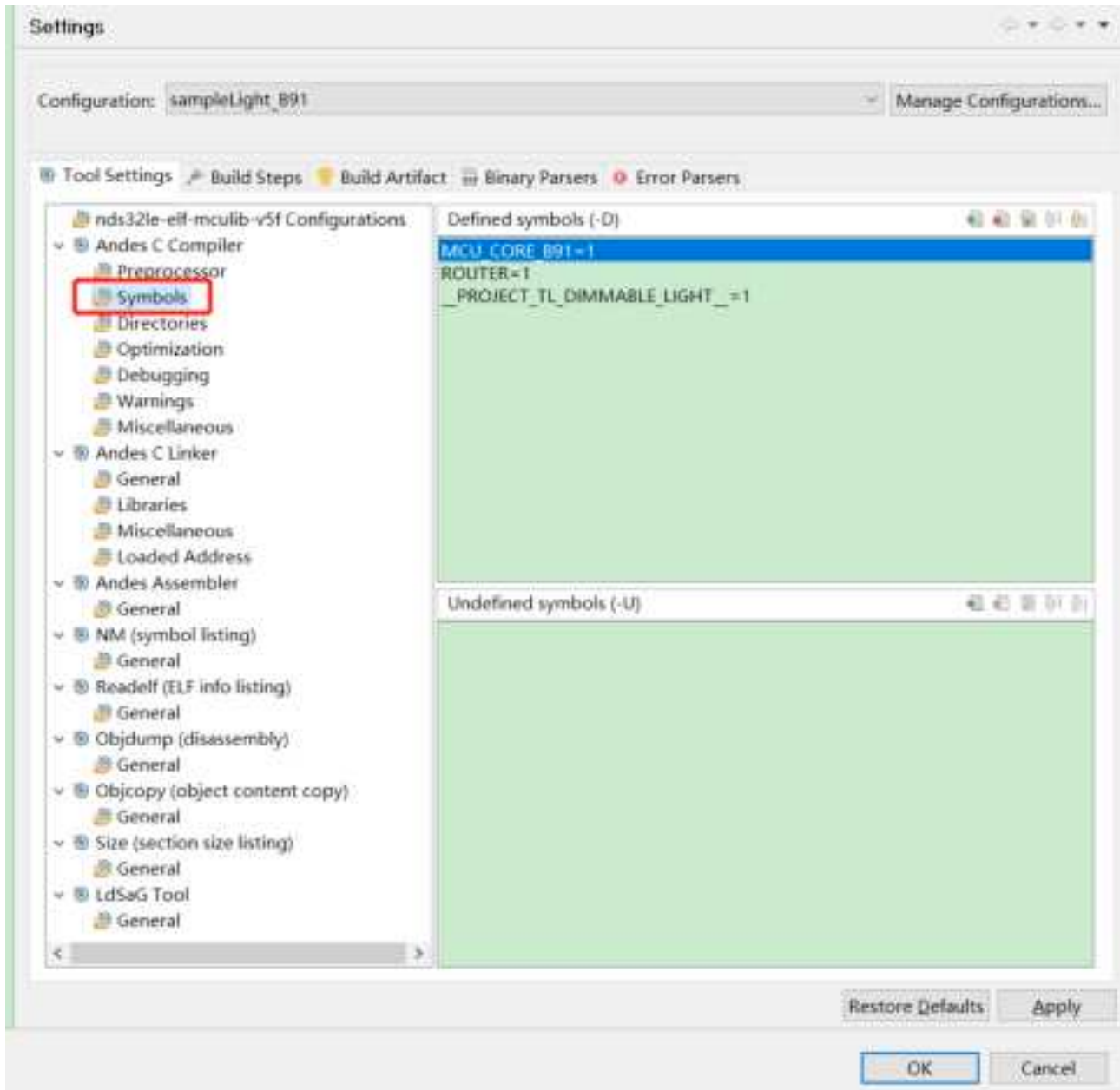


Figure 2.19: Symbol definition

The “Tool Settings” contains some pre-defined configurations for current project:

1) Device type pre-definition

-DROUTER=1: indicates that the project is a device with routing function

Following shows the device setting for coordinator and end device:

-DEND_DEVICE=1: indicates that the project is a device with end device function

-DCOORDINATOR=1: indicates that the item is a device with coordinator function

2) Platform selection

- b91 platform: -DMCU_CORE_B91=1

The startup code cstartup_b91.S is located in the \tl_zigbee_sdk\platform\boot\b91 directory.

3) Links for lib files

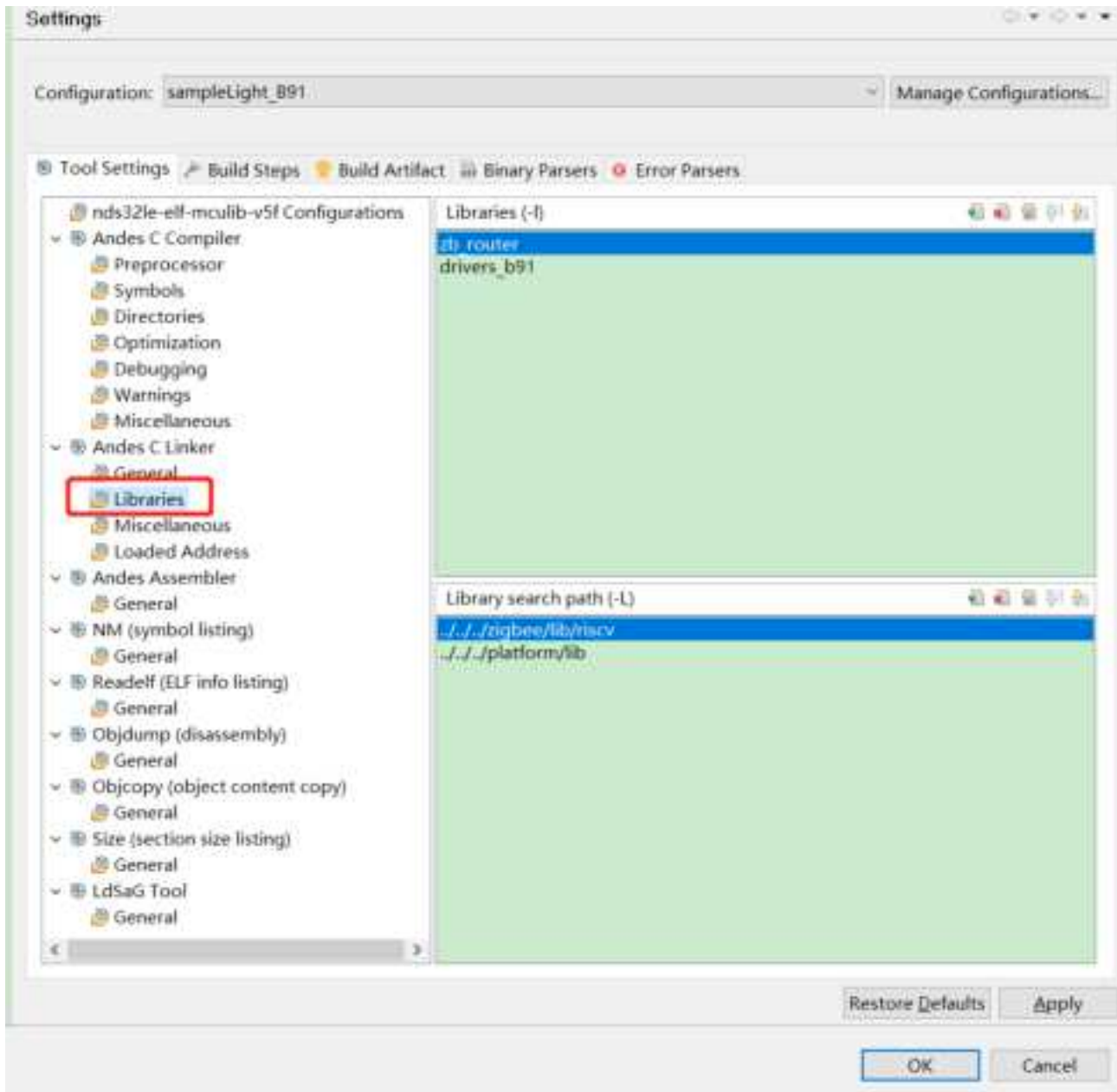


Figure 2.20: Library files and paths

The current SDK includes 2 types of library files: the Zigbee stack library and the platform driver library.

- Zigbee stack libraries: libzb_router.a, libzb_coordinator.a, libzb_ed.a
Available in the \tl_zigbee_sdk\zigbee\lib\riscv directory.
- Platform driver library: libdrivers_b91.a
Available in the \tl_zigbee_sdk\platform\lib directory.

4) Link file

According to the actual application requirements, the user can adjust the appropriate link file as per the different platforms chosen and the memory requirements.

The current SDK provides the default link file boot_b91.link for the b91 platform in the directory corresponding to \tl_zigbee_sdk\platform\boot.

As shown in the figure below, the link file to be used can be selected by invoking the script tl_link_load.sh (in the directory of \tl_zigbee_sdk\tools) during the pre-compiling.

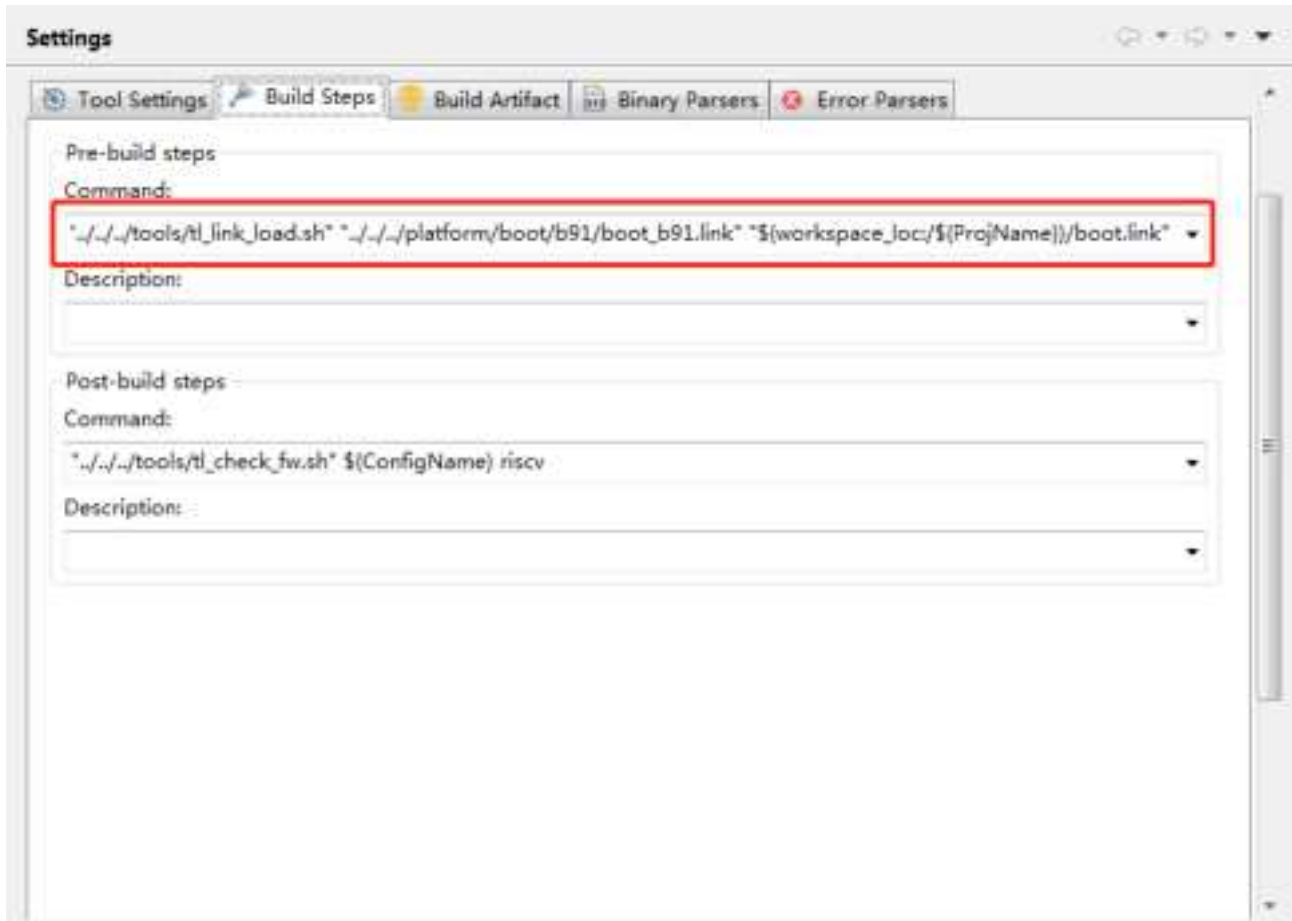


Figure 2.21: Link file pre-compiling settings

5) .image file check

In order to ensure the reliability of the downloaded file, a check field is added to the generated image file by the script tl_check_fw.sh (in the directory of \tl_zigbee_sdk\tools), and after the OTA process it will determine whether to run the new image by checking whether the check field matches or not.

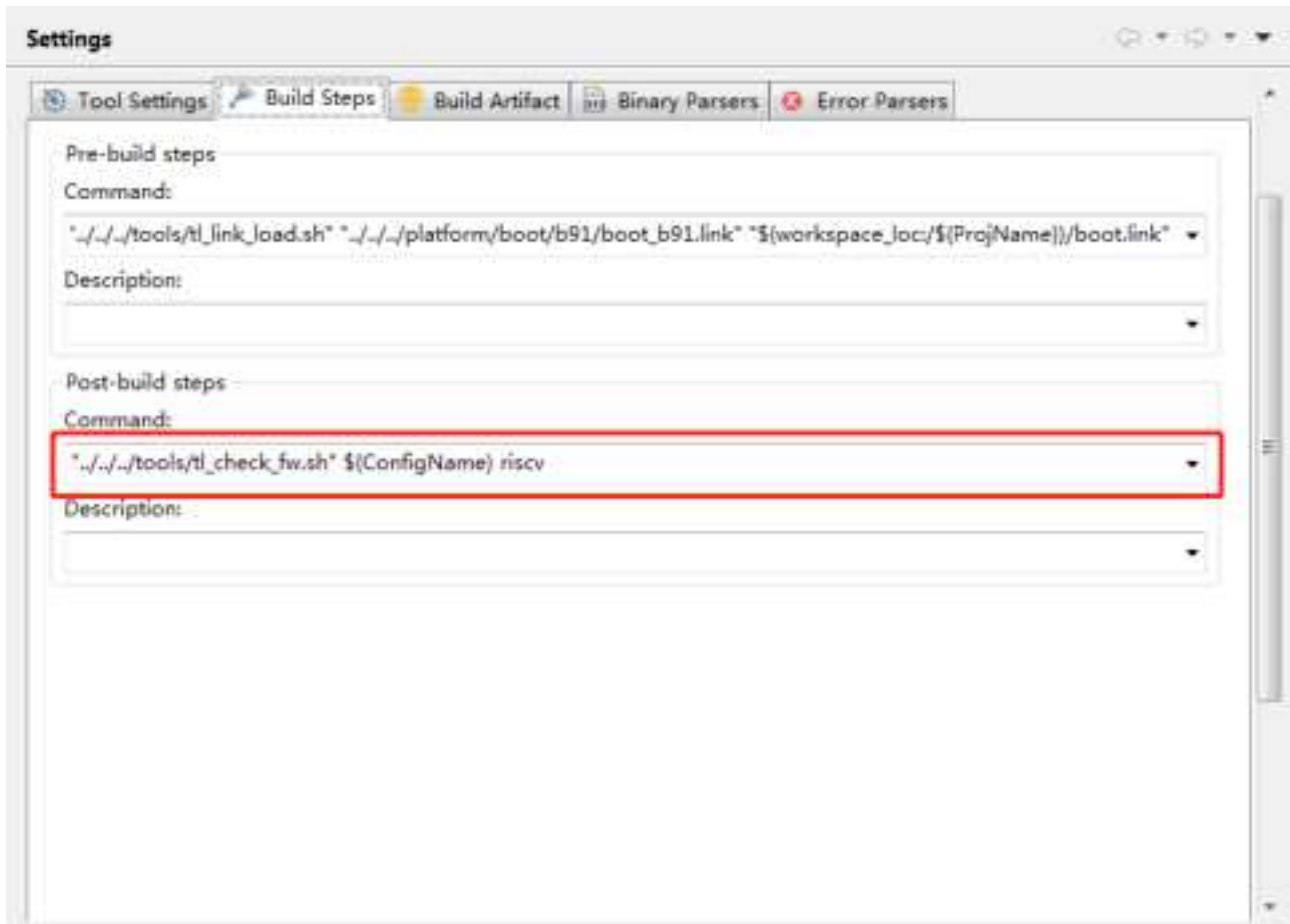


Figure 2.22: image check settings

2.3 App version management

Starting from SDK V3.6.0, there is a version_cfg.h file in each demo directory to manage the app version. It is very necessary to manage the version of the app, especially during OTA to prevent the risk of being bricked due to incorrect upgrade.

App version is composed of three key character fields, namely Manufacturer Code, Image Type and File Version, which will be stored in a fixed location in the firmware in the form of a small end, as shown below. (The field definitions can be found in Zigbee Cluster Library Specification).

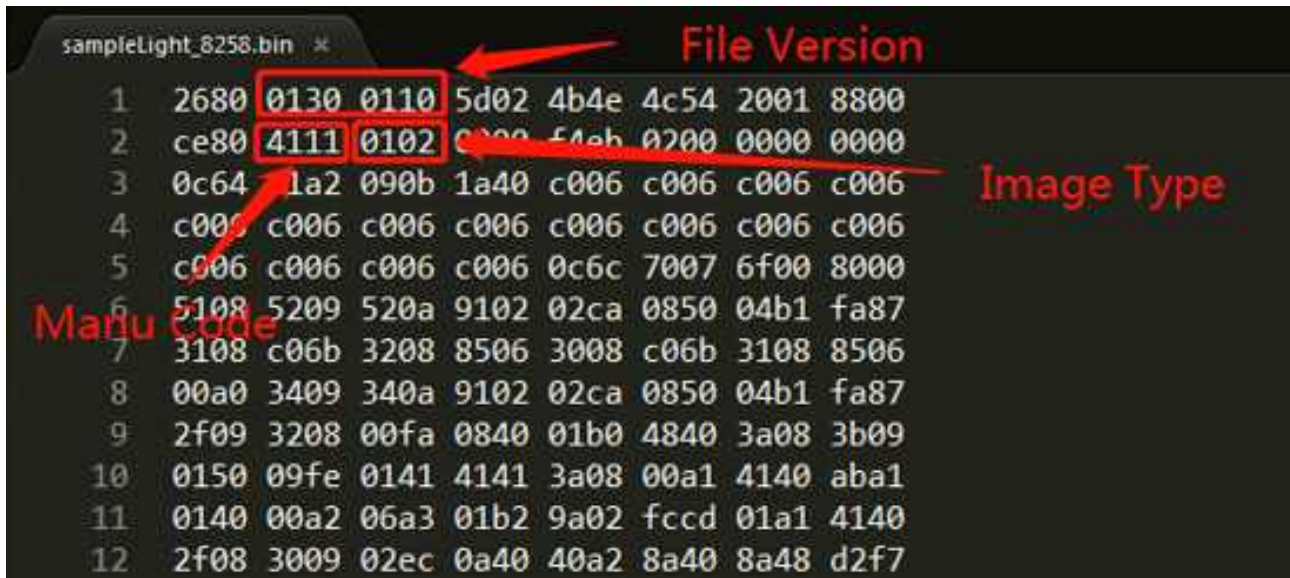


Figure 2.23: Binary file of image

2.3.1 Manufacturer Code

MANUFACTURER_CODE: The manufacturer code is a 2-byte-long identifier assigned by the Zigbee Alliance to each member company, for example, 0x1141 is the manufacturer code of Telink. Users can modify it to their own manufacturer code.

2.3.2 Image Type

IMAGE_TYPE: The image type is a 2-byte constant, the high byte represents the Chip type and the low byte represents the Image type. The Chip type and Image type are defined in the version_comm.h file, which can be modified or added by users according to their needs.

2.3.3 File Version

FILE_VERSION: The file version is the version number that reflects the firmware release and compiling, and is a constant of 4 bytes in length. The file version should be managed in incremental form, e.g. the current file version number should be greater than the previous release number, because when OTA, the upgrade will be performed only if the new version number is checked to be greater than the previous version number.

2.4 Operation mode

Telink Zigbee SDK provides two modes of operation: multi-address boot mode (boot from 0x0 or 0x40000 address) and BootLoader boot mode (boot from BootLoader and jump to App).

The user can change the operation mode by modifying the macro definition BOOT_LOADER_MODE in the comm_cfg.h file. The SDK uses multi-address boot mode by default.

```
#define BOOT_LOADER_MODE    0//1
```

It should be noted that the SDK will use different Flash layouts for different operation modes and different Flash size capacities.

2.4.1 Comparison of the advantages and disadvantages of the two modes

Multi-address boot mode

Advantages: fast boot; no need to transfer image again after OTA, fast boot after correct verification

Disadvantage: image can only be located at address 0x0 or 0x40000, which will cause discontinuity in flash space allocation; in addition, the size of image (if OTA is supported) can only be less than 208KB.

Bootloader boot mode

Advantages: usually the bootloader is located at address 0x0, the image file can be flexibly selected to address the space, so that users can define their own address space.

Disadvantages: slower boot speed than multi-address boot mode; additional flash consumption for boot-loader code; additional work required for transferring images during OTA.

2.4.2 Multi-address boot mode flash allocation

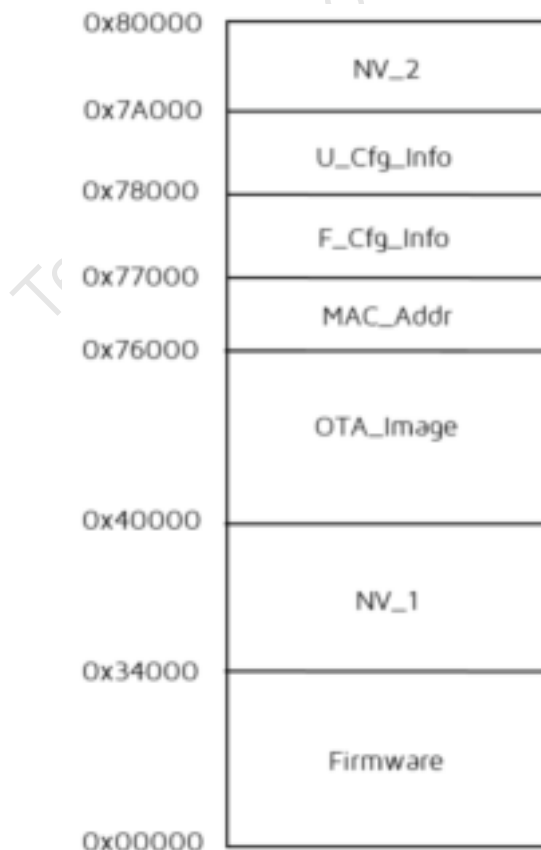


Figure 2.24: 512KB Flash space allocation chart

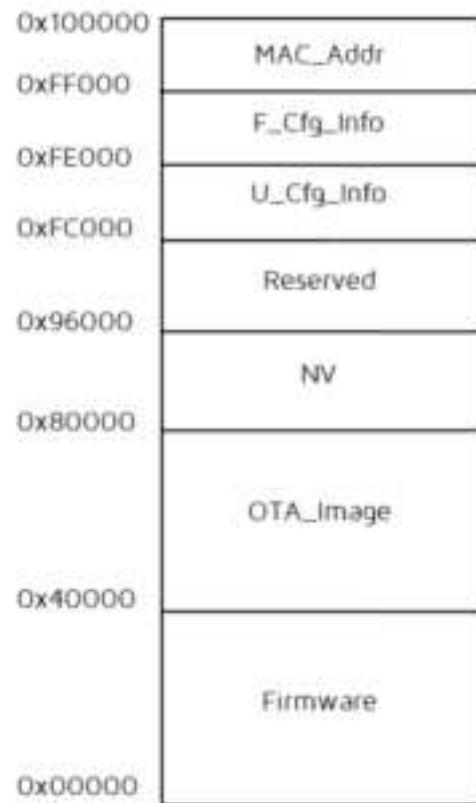


Figure 2.25: 1MB Flash space allocation chart

2.4.3 Bootloader boot mode flash allocation

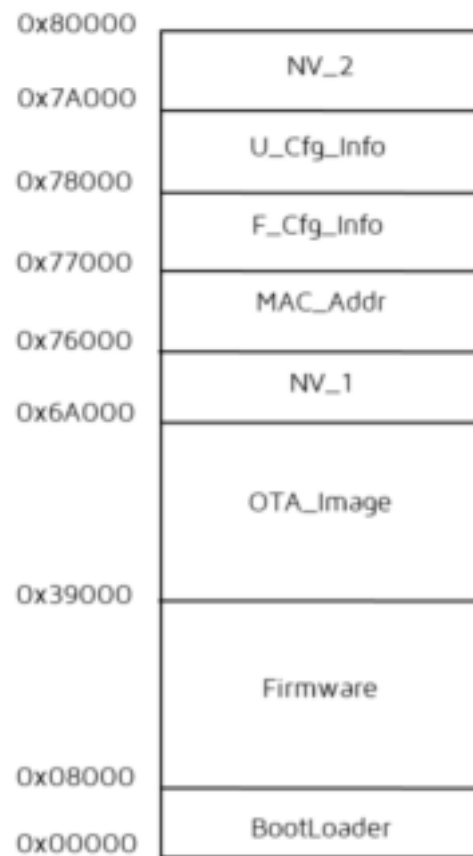


Figure 2.26: 512KB Flash space allocation chart

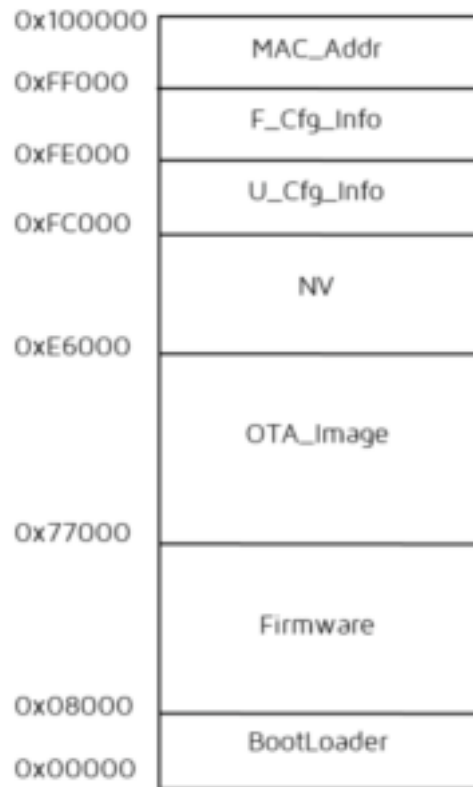


Figure 2.27: 1MB Flash space allocation chart

2.5 Flash Allocation Description

1) MAC_Addr

The MAC address is stored in 8 bytes location starting from the 0x76000 or 0xFF000 in the flash. The system will check the MAC address information after startup, and if it is found to be 0xFFFFFFFFFFFFFFFF, a MAC address will be generated randomly. A MAC address is pre-written when the chip is out of factory, so erase it carefully.

2) F_Cfg_Info

Factory configuration parameter information. The chip will be pre-written with some calibration parameter information at the factory, such as RF frequency bias calibration, ADC calibration, etc. Please erase it carefully.

3) U_Cfg_Info

Information about user configuration parameters. For example, the install code and factory reset flags are stored in this area.

4) NV(NV_1, NV_2)

Network information is stored in the NV area after the node is on the network. The 512KB Flash network information is stored in two parts, 48KB in NV_1 and 24KB in NV_2. The 1MB Flash network information is stored in 88KB in NV. So, if you only update the firmware or power off and reboot, the network information will not be lost.

5) Firmware and OTA_Image

TLSR8 and TLSR9 support Flash multi-address boot, i.e., you can boot from address 0x0 or 0x40000. The Telink Zigbee SDK uses this feature with ping-pong mode to implement OTA functionality, using the Firmware and OTA_Image to switch between each other.

6) BootLoader

Bootstrapping program, used in BootLoader operation mode.

2.6 Firmware burning

1) Hardware connection

Connect the Telink burning EVK to PC via mini USB cable, the EVK board indicator will blink once to indicate that the EVK is properly connected to the PC; then use three DuPont cables to connect the VCC, GND and SWM of the EVK to the VCC, GND and SWS of the target board to be burned, respectively.



Figure 2.28: Burning connection

2) After the hardware is connected, open the Telink BDT.exe burning tool and prepare to burn the firmware.

- a) Select "8258" chip (this document takes 8258 for example).
- b) Select "File" -> "Open" and choose the firmware to be burned.
- c) Click "Erase" to erase 512KB Flash.
- d) Click "Download" to download the firmware.
- e) If you use the bootloader mode, please select "setting" and modify "Download Addr", burn the bootloader and App firmware to the corresponding address (refer to [section 2.4.3.](#))

(For detailed usage of the burning tool, please refer to the documentation on using the burning tool.)

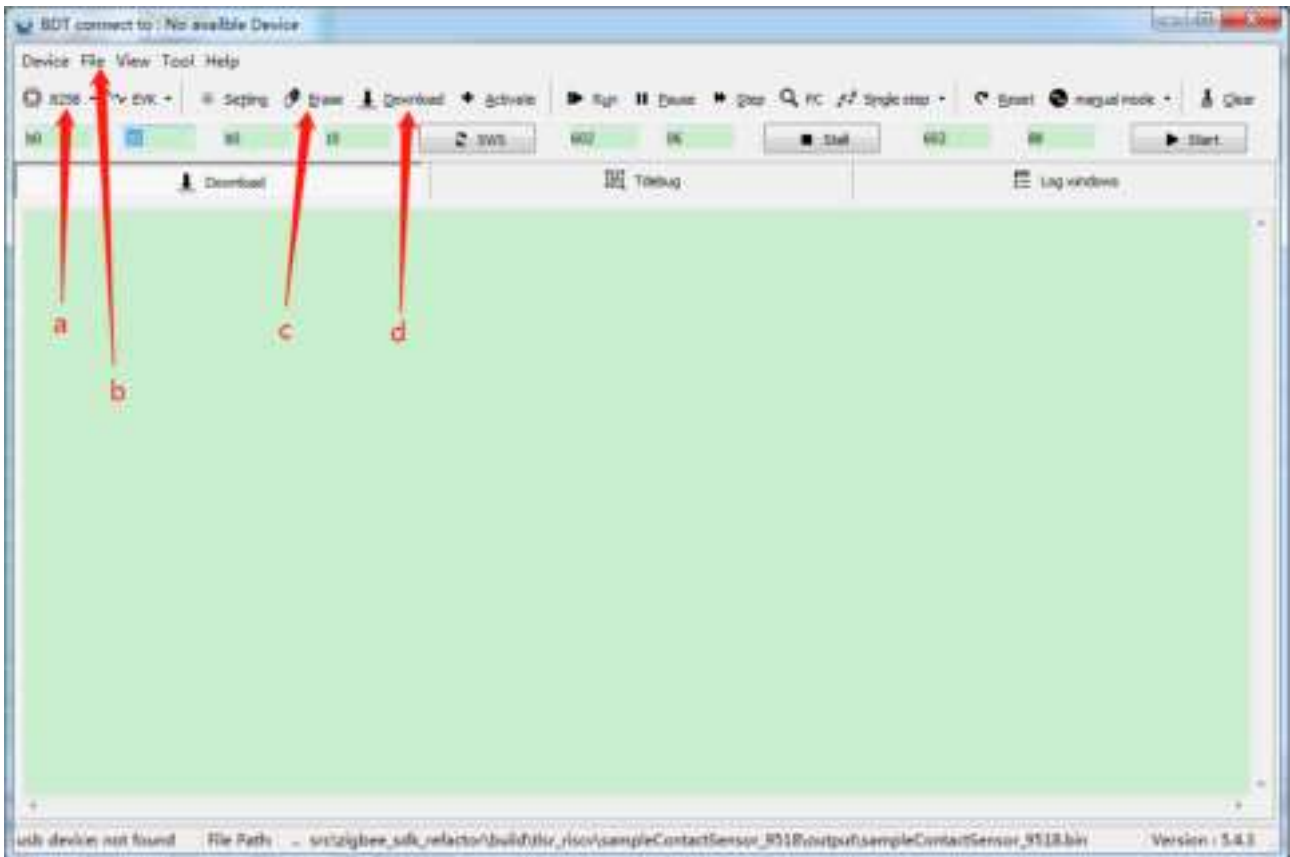


Figure 2.29: Burning tool

3 Software Architecture

The directory structure of Telink Zigbee SDK is listed in the [section 2.1.2](#) and [2.2.2](#), and the files in each directory are described in this chapter.

3.1 Directory description

3.1.1 Hardware platform directory

The current SDK supports b85m (826x, 8258, 8278) and b91m (B91 series) for multiple platforms (\tl_zigbee_sdk\platform). Other hardware platforms will be added in the coming versions.

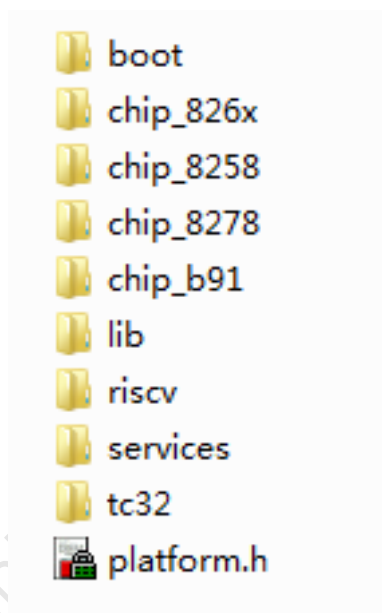


Figure 3.1: Hardware platform directory

- \boot: .S boot code and .link link files for different platforms
- \chip_826x: 826x platform hardware module driver header file
- \chip_8258: 8258 platform hardware module driver header file
- \chip_8278: 8278 platform hardware module driver header file
- \chip_b91: b91 platform hardware module driver header file
- \lib: driver library files for different platforms
- \services: interrupt handling files for different platforms
- \riscv: supporting risc-v platform related files
- \tc32: supporting tc32 platform related files

3.1.2 Common function directory

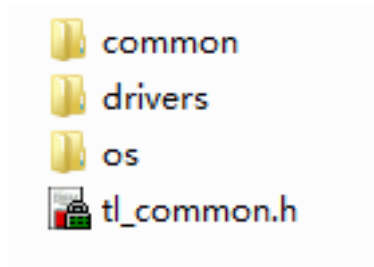


Figure 3.2: Common functions directory

- \common: some common functional functions, such as string, chain table, printing and other processing function files
- \drivers: abstraction layer driver files
- \os: task events, buffer management function files

3.1.3 Zigbee protocol stack directory

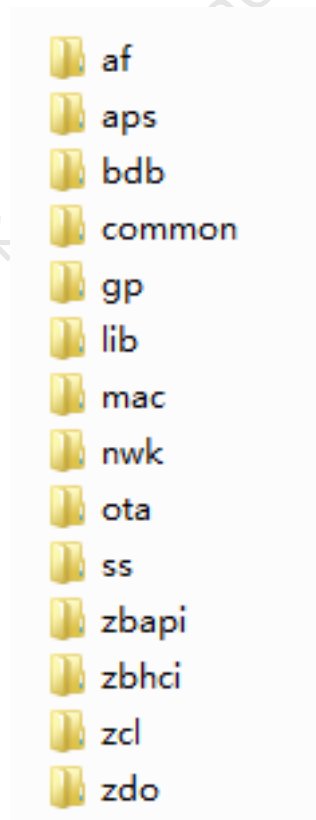


Figure 3.3: Zigbee protocol stack directory

In this directory, most are given in the form of .lib file, while ZCL and zbhci, which are closely related to the PHY layer and the application layer, are given as open source code.

- \af: Application Framework layer related documents
- \aps: Application Support Sub-Layer related files
- \bdb: Base Device Behavior specification-related documents
- \common: Zigbee protocol stack configuration-related files
- \gp: Green Power specification-related documents
- \lib: Zigbee protocol stack library file
- \mac: Media Access Control layer related files
- \nwk: Network layer related files
- \ota: Over The Air upgrade related documents
- \ss: Security Services related documents
- \zbapi: Zigbee common interface function file
- \zcl: Zigbee Cluster Library related files
- \zdo: Zigbee Device Objects related files

3.1.4 Application layer directory

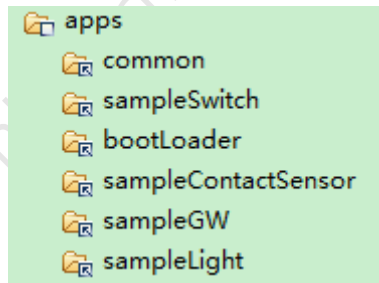


Figure 3.4: Application layer directory

- \common: application layer common file directory
 - factory_reset.c/h: device reset by power-off operation
 - firmwareEncryptChk.c/h: software encryption for anti-copy, using the uniqueness of the UID to prevent the firmware from being copied
 - module_test.c: module test file, only for test verification during development
 - main.c: main function entry
 - common_cfg.h: version and operating mode configuration file
- \sampleGW: Gateway application samples
- \sampleLight: Light application samples

- \sampleSwitch : Switch application samples
- \sampleContactSensor: Magnetic door application samples

3.1.5 Project compiling directory

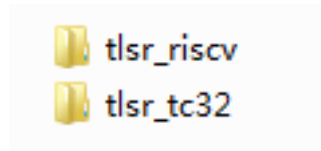


Figure 3.5: Project compiling directory

- \build\tlsr_riscv: the project directory for risc-v platform import and compiling. Please choose to use this directory for TLSR9 series chips.
- \build\tlsr_tc32: the project directory for tc32 platform import and compiling. Please choose to use this directory for TLSR8 series chips.

3.2 Abstract layer driver

Telink Zigbee sdk is compatible with many Telink chips, in order to facilitate the unification of the driver interface, a driver abstraction layer has been added, located under /proj/drvier/, the specific use of each driver can be found in apps/module_test.c file.

3.2.1 Platform initialization

- Initialization

```
drv_platform_init(void)
```

Complete the initialization of chip platform, system clock, gpio, radio frequency (RF), timer and other modules required in Zigbee application development.

3.2.2 Radio Frequency (RF)

- Initialization

```
ZB_RADIO_INIT()
```

- Mode switching

ZB_RADIO_TRX_SWITCH(mode, chn)

mode:

RF_MODE_TX = 0, transmit mode

RF_MODE_RX = 1, receive mode

RF_MODE_AUTO = 2, Zigbee not used

RF_MODE_OFF, turn off the RF module

Chn: 11-26 (16 channel values corresponding to 2.4G in 802.15.4)

- Transmit power

ZB_RADIO_TX_POWER_SET(level)

Set transmit power for RF module, different chips correspond to slightly different power values, please refer to the definition of RF_PowerIndexTypeDef (platform/chip/rf_drv.h)

- Data sending

ZB_RADIO_TX_START(txBuf)

txBuf: the memory address of the data to be sent.

Data format: dma length (4Bytes: payload length+ 1)+ len(1Byte: payload length+ 2)+ payload

- Set the RF receiving buffer

ZB_RADIO_RX_BUF_SET(addr)

When setting the RF to Rx mode, you should ensure Rx buffer set to a valid and safe memory address.

- RSSI obtain

ZB_RADIO_RSSI_GET()

When invoking this function, make sure the RF is in Rx mode at this time.

- Rssi to Lqi

ZB_RADIO_RSSI_TO_LQI(mode, rssi, lqi)

mode: valid only for 8269

```
typedef enum{  
    RF_GAIN_MODE_AUTO,  
    RF_GAIN_MODE_MANU_MAX,  
}rf_rxGainMode_t;
```

- Receive interrupt

```
rf_rx_irq_handler(void)
```

- Send interrupt

```
rf_tx_irq_handler(void)
```

Note:

- RF interrupt functions rf_rx_irq_handler/rf_tx_irq_handler are for Zigbee stack use only. If users call ZB_RADIO_TX_START on their own, the generated interrupt needs to be handled by the users themselves (registering a new interrupt callback function) to avoid affecting the operation status of the Zigbee stack.

3.2.3 GPIO

- Initialization and configuration

```
gpio_init()
```

This function is used to initialize gpio by writing the default configuration under platform/chip_xxxx/gpio_default.h (which can be modified by board_xxxx.h in the application layer) to the gpio registers.

- IO function setting

```
void drv_gpio_func_set(u32 pin)
```

Set IO specific functions

Parameter pin: See the definition of GPIO_PinTypeDef

- GPIO read

```
void drv_gpio_read(u32 pin);
```

Read the high and low levels of gpio

- GPIO write

```
void drv_gpio_write(u32 pin, bool value);
```

Set the high and low levels of gpio

- Output enable

```
void drv_gpio_output_en(u32 pin, bool enable);
```

- Input enable

```
void drv_gpio_input_en(u32 pin, bool enable)
```

- GPIO interrupt operation

The chip can support up to 3 external GPIO interrupts at the same time, the interrupt modes are: GPIO_IRQ_MODE, GPIO_IRQ_RISC0_MODE and GPIO_IRQ_RISC1_MODE.

1) Register interrupt service functions

```
int drv_gpio_irq_conf(drv_gpio_irq_mode_t mode, u32 pin,  
                     drv_gpioPoll_e polarity,  
                     irq_callback gpio_irq_callback);
```

2) Enable interrupt pin

```
int drv_gpio_irq_en(u32 pin);  
int drv_gpio_irq_risc0_en(u32 pin);  
int drv_gpio_irq_risc1_en(u32 pin);
```

3.2.4 UART

- Pin setting

```
void drv_uart_pin_set(u32 txPin, u32 rxPin)
```

Before using uart, you should select the corresponding IO as sending and receiving pin of uart.

- Initialization

```
void drv_uart_init(u32 baudrate, u8 *rxBuf, u16 rxBufLen,  
                  uart_irq_callback uart_recvCb)
```

baudrate: Baud rate

rxBuf: Receiving buffer

rxBufLen: maximum length of received data

uart_recvCb: Callback function for the application layer when receiving interrupts

- Sending

```
u8 drv_uart_tx_start(u8 *data, u32 len)
```

- Receive interrupt function

```
void drv_uart_rx_irq_handler(void)
```

- Send interrupt function

```
void drv_uart_tx_irq_handler(void)
```

- Exception processing function

```
void drv_uart_exceptionProcess(void)
```

Note:

- When using uart, main_loop should be polled for this exception processing in order to avoid communication exceptions.

3.2.5 ADC

- Initialization

```
bool drv_adc_init(void);
```

- Configuration

```
void drv_adc_mode_pin_set(drv_adc_mode_t mode, GPIO_PinTypeDef pin)
```

mode: see drv_adc_mode_t definition

pin: for the pin number used, see GPIO_PinTypeDef definition

- Get the sample value

```
u16 drv_get_adc_data(void)
```

3.2.6 PWM

- Initialization

```
void drv_pwm_init(void)
```

- Configuration

```
void drv_pwm_cfg(u8 pwmId, u16 cmp_tick, u16 cycle_tick)
```

pwmId: pwm channel

cmp_tick: the number of ticks that are high in one cycle of PWM

cycle_tick: the number of ticks contained in one PWM cycle

3.2.7 Timer

- Initialization

```
void drv_hwTmr_init(u8 tmrIdx, u8 mode)
```

tmrIdx: TIMER_IDX_0, TIMER_IDX_1, TIMER_IDX_2, TIMER_IDX_3

mode: TIMER_MODE_SYSCLOCK, TIMER_MODE_GPIO_TRIGGER, TIMER_MODE_GPIO_WIDTH, TIMER_MODE_TICK

- Setting

```
void drv_hwTmr_set(u8 tmrIdx, u32 t_us, timerCb_t func, void *arg)
```

tmrIdx: The timer to be used

t_us: Timing interval, unit: us

func: The timed interrupt application layer callback function

arg: Parameters required by the application layer callback function

- Cancel

```
void drv_hwTmr_cancel(u8 tmrIdx)
```

- Interrupt function

```
void drv_timer_irq0_handler(void)
```

```
void drv_timer_irq1_handler(void)
```

```
void drv_timer_irq3_handler(void)
```

Where TIMER_IDX_2 is used as watchdog by default, users are advised not to use it.

TIMER_IDX_3 is used as MAC-CSMA and users are advised not to use it.

3.2.8 Watchdog

- Initialization

```
void drv_wd_setInterval(u32 ms)
```

ms: Timeout time

- Start

```
void drv_wd_start(void)
```

- Feed the watchdog

```
void drv_wd_clear(void)
```

3.2.9 System Tick

The System Tick counter is a 32-bit length readable counter that automatically adds one every clock cycle.

Since the clock source of System Timer of 826x and other series ICs are different, the System Timer of 8269 is 32M and the System Timer of 8258, 8278 and B91 series is 16M, so there is a difference in the maximum counting time.

- Maximum Timing Time of 8269: $(1/32)\mu s \times (2^{32})$ is approximately equal to 134 seconds, System Timer Tick runs one cycle every 134 seconds.
- Maximum Timing Time of 8258, 8278, B91 series: $(1/16)\mu s \times (2^{32})$ is approximately equal to 268 seconds, System Timer Tick runs one cycle every 268 seconds.

The user can use the `clock_time()` interface to get the current tick.

3.2.10 Voltage detection

In order to prevent abnormal operation of low-voltage systems, the SDK provides a voltage detection function based on the ADC driver implementation, which requires below attention.

- 1) It needs to use the I/O port that supports ADC function, and the I/O port used for ADC detection cannot be used for other functions.
- 2) When using `DRV_ADC_VBAT_MODE` mode, the I/O port needs to be floating.
- 3) When using `DRV_ADC_BASE_MODE` mode, the I/O port needs to be connected to a voltage test point.
- 4) B91 can only use `DRV_ADC_BASE_MODE` mode, and I/O ports need to do external partial voltage processing

- Initialization

```
void voltage_detect_init(void)
```

- Voltage detection

```
voltage_detect(void)
```

3.2.11 Sleep and wake-up

Telink Zigbee SDK provides related low-power management functions. The sampleSwitch_826x uses suspend mode; sampleSwitch_8258, sampleSwitch_8278 and sampleSwitch_B91 use deep with retention mode, i.e. RAM data can be retained while sleeping (8258 and 8278 support 32KB RAM retention, B91 supports 64KB RAM retention, please refer to the datasheet for details).

- Wake-up source type

Supports button wake-up and timer wake-up

- Configure wake-up pins

If you need the button wake-up function, you need to configure the corresponding wake-up pins and wake-up voltage level first.

```
/**
 * @brief Definition for wakeup source and level for PM
 */
drv_pm_pinCfg_t g_switchPmCfg[] =
{
    {BUTTON1, PM_WAKEUP_LEVEL},
    {BUTTON2, PM_WAKEUP_LEVEL},
};
drv_pm_wakeupPinConfig(g_switchPmCfg,
    sizeof(g_switchPmCfg)/sizeof(drv_pm_pinCfg_t));
```

- Deepsleep functions

```
drv_pm_lowPowerEnter(void);
```

If button wake-up is enabled, when invoking this function, if the current level of the corresponding pin is the same as the wake-up level, the system cannot effectively enter the low-power state.

- Condition detection for entering sleep mode

- 1) Call `ingtl_stackBusy()` and `zb_isTaskDone()` to check whether the sleep conditions are met.
- 2) Iterate through the software timed task list, check if a timed task exists, if so execute step 3), if not execute step 4).
- 3) Retrieve the time of the approaching task and use that time as the sleep time to enter Suspend or DeepRetention mode, which can be timer wake-up and button wake-up.
- 4) Enter deep sleep mode, you can press the button to wake up.

3.3 Memory management

3.3.1 Dynamic memory management

Telink Zigbee SDK provides an interface for dynamic memory allocation and release, it can be invoked directly by users in development. The detailed description of the interface is as below.

1) Memory allocate

```
u8 *ev_buf_allocate(u16 size);
```

2) Memory free

```
buf_sts_t ev_buf_free(u8 *pBuf);
```

3) Resource allocation

By default it consists of four different sets of memory pools of different numbers and sizes, which can be modified by users according to product requirements and current memory usage.

```
MEMPOOL_DECLARE(size_x_pool, size_x_mem, BUFFER_GROUP_x, BUFFER_NUM_IN_GROUPx);
```

3.3.2 NV Management

Because Zigbee needs to store the network information parameters of each layer to be able to recover the network after an abnormal power failure, the SDK divides an area from flash specifically for storing such information, which we invoke the NV area, i.e., NV(NV_1, NV_2) area in [section 2.5](#).

Since the chip flash supports only sector (1 sector= 4k) erasure, the minimum unit of information storage module in NV area is 1 sector.

The following information modules are used in the SDK, among them NV_MODULE_APP is for users, if you need to add new information modules, please add them at the end, you cannot change the existing module serial number.

```
typedef enum {  
    NV_MODULE_ZB_INFO           = 0,  
    NV_MODULE_ADDRESS_TABLE     = 1,  
    NV_MODULE_APS               = 2,  
    NV_MODULE_ZCL               = 3,  
    NV_MODULE_NWK_FRAME_COUNT   = 4,  
    NV_MODULE_OTA               = 5,  
    NV_MODULE_APP               = 6,  
    NV_MODULE_KEYPAIR           = 7,  
  
    //NV_MODULE_ADD,
```

```
NV_MAX_MODULS
}nv_module_t;
```

The information module consists of multiple entry messages, for example NV_MODULE_ZCL consists of a series of NV_ITEM_ZCL_XX. The entries are defined as follows. If you need to add new entry information, please add them at the end, you cannot change the existing entry serial number.

```
typedef enum {
    NV_ITEM_ID_INVALID          = 0, /* Item id 0 should not be used. */

    NV_ITEM_ZB_INFO             = 1,
    NV_ITEM_ADDRESS_FOR_NEIGHBOR,
    NV_ITEM_ADDRESS_FOR_BIND,
    NV_ITEM_APS_SSIB,
    NV_ITEM_APS_GROUP_TABLE,
    NV_ITEM_APS_BINDING_TABLE,
    NV_ITEM_NWK_FRAME_COUNT,
    NV_ITEM_SS_KEY_PAIR,
    NV_ITEM_OTA_HDR_SERVERINFO,
    NV_ITEM_OTA_CODE,
    NV_ITEM_ZCL_REPORT          = 0x20,
    NV_ITEM_ZCL_ON_OFF,
    NV_ITEM_ZCL_LEVEL,
    NV_ITEM_ZCL_COLOR_CTRL,
    NV_ITEM_ZCL_SCENE_TABLE,
    NV_ITEM_ZCL_GP_PROXY_TABLE,
    NV_ITEM_ZCL_GP_SINK_TABLE,
    NV_ITEM_APP_SIMPLE_DESC,
    NV_ITEM_APP_POWER_CNT,

    NV_ITEM_ID_MAX              = 0xFF, /* Item id 0xFF should not be used. */
}nv_item_t;
```

- Each module occupies 2 or (2*n) sectors, in the format of

sector info + item index + item content

where the sector info identifies whether this module is valid, the length is sizeof(nv_sect_info_t), followed by the item index to be written, the item content starts at a 512Byte or 1024Byte offset from the module's first address.

- To avoid frequent erasure operations, NV uses a single-module append-write method until one sector is full, moves the valid information to another sector, and then erases the contents of the previous sector.
- The interface for NV area read and write operations is as follows.

```
nv_sts_t nv_flashWriteNew(u8 single, u16 id, u8 itemId, u16 len, u8 *buf);  
nv_sts_t nv_flashReadNew(u8 single, u8 id, u8 itemId, u16 len, u8 *buf);
```

Note:

- When writing an item, if there is only one valid value, single should be set to 1; when this item is written again, previous valid item will be cleared; otherwise, you need to retrieve the same item according to its content, set it to invalid, and then write a new item.

3.4 Task management

3.4.1 Single task queue

- Interface function

```
TL_SCHEDULE_TASK(tl_zb_callback_t func, void *arg)
```

func: Task callback function

arg: Required parameters for the task

- Execute only once, process in order without priority
- Recommended use occasions.
 - 1) Needs to avoid stack overflow problems caused by deep nesting of functions.
 - 2) In the interrupt function, avoid consuming too much time in the interrupt function by pressing data and events into the queue.
- Size: 32 (avoid pushing too many tasks at once)

3.4.2 Standing task queue

- **Task registration** (start after task registration by default)

```
ev_on_poll(ev_poll_e e, ev_poll_callback_t cb)
```

- **Task suspension**

```
ev_disable_poll(ev_poll_e e, ev_poll_callback_t cb)
```

- **Task restart**

```
ev_enable_poll(ev_poll_e e, ev_poll_callback_t cb)
```

After this task is registered, it will be executed in the main loop all the time.

3.4.3 Software timer task

In order to facilitate users to implement timing tasks that do not require high time accuracy, Telink Zigbee SDK provides a software timing task management mechanism.

It should be noted that from SDK V3.6.2 version, the time unit of software timing task management changed from original ticker to millisecond, which solves the problem of long time timing task demand.

3.4.3.1 Interface functions

- **Task registration**

```
TL_ZB_TIMER_SCHEDULE(cb, arg, timeout)
```

- **Task cancellation:**

```
TL_ZB_TIMER_CANCEL(evt)
```

3.4.3.2 Attentions

Three uses of the return value of the timed task callback function.

- If the return value is less than 0, the task is automatically deleted after execution and the task ceases to exist.
- If the return value is equal to 0, the callback function always triggered periodically using the timeout in previous start-up.
- If the return value is greater than 0, the return value is used as the new period to trigger the callback function at regular intervals, in ms.

Note:

- Avoid using TL_ZB_TIMER_CANCEL() in the interrupt function.

3.4.3.3 Examples

When VK_SW1 button is pressed, a 10-second timed task started or cancelled, command to broadcast On/Off Toggle executed once when the time is reached, and exits when the execution is finished. The code is as follows.

Note that brc_toggleEvt must be cleared to zero before return -1. Otherwise, after the callback function is executed, brc_toggleEvt still points to the memory of the timed task, even though the memory is freed. When VK_SW1 button is pressed again, it will be executed incorrectly.

```
ev_timer_event_t *brc_toggleEvt = NULL;

s32 brc_toggleCb(void *arg)
{
    epInfo_t dstEpInfo;
    TL_SETSTRUCTCONTENT(dstEpInfo, 0);

    dstEpInfo.dstAddrMode = APS_SHORT_DSTADDR_WITHEP;
    dstEpInfo.dstEp = SAMPLE_GW_ENDPOINT;
    dstEpInfo.dstAddr.shortAddr = 0xffff;
    dstEpInfo.profileId = HA_PROFILE_ID;
    dstEpInfo.txOptions = 0;
    dstEpInfo.radius = 0;

    zcl_onOff_toggleCmd(SAMPLE_GW_ENDPOINT, &dstEpInfo, FALSE);

    brc_toggleEvt = NULL; //must clear
    return -1;
}

void buttonShortPressed(u8 btNum)
{
    if(btNum == VK_SW1){
        if(!brc_toggleEvt){
            brc_toggleEvt = TL_ZB_TIMER_SCHEDULE(brc_toggleCb,
                                                    NULL,
                                                    10 * 1000);
        }else{
            TL_ZB_TIMER_CANCEL(&brc_toggleEvt);
        }
    }
}
```

4 Zigbee SDK Application Development

4.1 Operation mode selection

The user can change the operation mode by modifying the macro definition `BOOT_LOADER_MODE` in the `comm_cfg.h` file, see [section 2.4](#) for details on the differences between the two modes.

```
#define BOOT_LOADER_MODE 0//1
```

- SDK defaults to multi-address boot mode (`BOOT_LOADER_MODE` is 0)
- If bootloader method is used, please set `BOOT_LOADER_MODE` to 1.

1) `BOOT_LOADER_IMAGE_ADDR`: the location where the bootloader image is placed, usually 0.

2) `APP_IMAGE_ADDR`: the location of the user image, can be modified according to the actual application, the default is 0x8000.

4.2 Hardware selection

Telink Zigbee SDK demo can run on different series of chips and different hardware boards, users need to do the corresponding configuration for different hardware conditions.

4.2.1 Chip model confirmation

Telink Zigbee SDK lists the following chip types in `comm_cfg.h` file. Before compiling project examples, users should check whether the chip type selected in `version_cfg.h` under the corresponding project is the same as the actual chip type used.

4.2.1.1 comm_cfg.h chip type definition

```
/* Chip IDs */
#define TLSR_8267          0x00
#define TLSR_8269          0x01
#define TLSR_8258_512K    0x02
#define TLSR_8258_1M      0x03
#define TLSR_8278          0x04
#define TLSR_B91           0x05
```

4.2.1.2 version_cfg.h chip type selection

```
#if defined(MCU_CORE_826x)
    #if (CHIP_8269)
        #define CHIP_TYPE        TLSR_8269
    #else
        #define CHIP_TYPE        TLSR_8267
    #endif
#elif defined(MCU_CORE_8258)
    #define CHIP_TYPE        TLSR_8258_512K//TLSR_8258_1M
#elif defined(MCU_CORE_8278)
    #define CHIP_TYPE        TLSR_8278
#elif defined(MCU_CORE_B91)
    #define CHIP_TYPE        TLSR_B91
#endif
```

4.2.2 Target board selection

The Telink Zigbee SDK provides demos that run on EVK boards or USB Dongle, and the user can change the target board by selecting the app_cfg.h file under the corresponding demo.

```
/* Board ID */
#define BOARD_826x_EVK            0
#define BOARD_826x_DONGLE        1
#define BOARD_826x_DONGLE_PA     2
#define BOARD_8258_EVK          3
#define BOARD_8258_EVK_V1P2      4//C1T139A30_V1.2
#define BOARD_8258_DONGLE        5
#define BOARD_8278_EVK          6
#define BOARD_8278_DONGLE        7
#define BOARD_B91_EVK           8
#define BOARD_B91_DONGLE         9

/* Board define */
#if defined(MCU_CORE_8258)
    #if (CHIP_TYPE == TLSR_8258_1M)
        #define FLASH_CAP_SIZE_1M        1
    #endif
    #define BOARD                        BOARD_8258_DONGLE
    #define CLOCK_SYS_CLOCK_HZ          48000000
#elif defined(MCU_CORE_8278)
    #define FLASH_CAP_SIZE_1M            1
    #define BOARD                        BOARD_8278_DONGLE
    #define CLOCK_SYS_CLOCK_HZ          48000000
#elif defined(MCU_CORE_B91)
    #define FLASH_CAP_SIZE_1M            1
    #define BOARD                        BOARD_B91_DONGLE
```

```
#define CLOCK_SYS_CLOCK_HZ      48000000
#else
#error "MCU is undefined!"
#endif
```

During the pre-compiling stage, we need to load corresponding board-level configuration depending on the previous chip type selection. The configuration file is board_xx.h in each project directory, and you need to pay attention to whether the flash capacity of the target board is consistent with the board configuration to prevent errors in loading data.

```
#define FLASH_CAP_SIZE_1M      1
```

4.3 Print debug configuration

4.3.1 UART print

To avoid the waste of hardware UART resources, we implement the UART print function (printf() function) by simulating UART TX through GPIO, and the user can change the appropriate GPIO at will. The configuration method is as follows.

1) Print enable configuration

```
#define UART_PRINTF_MODE      1
```

2) Print port configuration

```
#define DEBUG_INFO_TX_PIN      GPIO_PC4//print
```

3) Baud rate configuration

```
#define BAUDRATE              1000000//1M
```

Note:

- 826x supports a maximum of 2M baud rate, while 8258, 8278 and B91 supports a maximum of 1M baud rate.
- Do not use the print in the interrupt processing function to prevent printing more data or function nesting too deep resulting in interrupt stack overflow.

4.3.2 USB print

USB print function can be used with Telink BDT tool. The configuration is as follows.

1) Print enable configuration

```
#define USB_PRINTF_MODE    1
```

2) Enables the USB port

```
#define HW_USB_CFG()      do{ |\n                          usb_set_pin_en();|\n                        }while(0)
```

Note:

- The USB print function is disabled when the ZBHCI_USB_CDC or ZBHCI_USB_HID function is used.

4.4 Zigbee network development process

4.4.1 Application layer initialization (user_init)

The application layer initialization mainly consists of Zigbee stack and Zigbee application layer initialization.

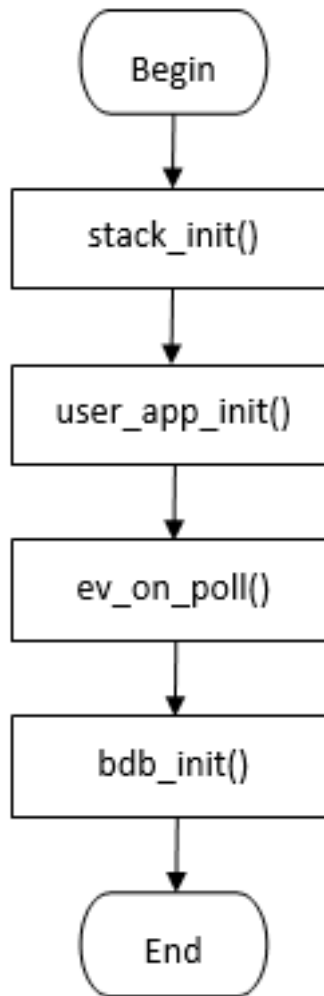


Figure 4.1: Application layer initialization flow

1) **stack_init()**

Complete the basic protocol stack initialization and the registration of the application layer callback functions given by the protocol stack.

- **zb_init():** Initialize the Zigbee protocol stack, including mac, nwk, aps, zdo and other layers, and for not factory-new devices, read incoming information from NV and restore network information and properties of each layer.

Note:

For devices that are already on the network, you cannot change the attributes by modifying the default attribute table configuration in `zb_config.c` file. If you want to modify the attributes, you should operate directly on the corresponding attribute using `g_zbMacPib/g_zbNIB/g_touchlinkAttr/g_bdbAttrs`.

- **zb_zdoCbRegister():** registers some callback functions given by the Zigbee stack, which can be used to obtain relevant information during the network operation. For example, the result of network creation/entry, device leaving the network, device joining the network, network state change, parent node receiving synchronization information, etc.

2) **user_app_init()**

The user registers the desired ports as well as clusters and attributes according to the specific product features.

- Port registration:

```
af_endpointRegister()
```

Register a port to specify the detailed function of the port (profileId, deviceId, port number, and supported input clusters (server), output clusters (client)).

- Cluster registration:

```
zcl_register()
```

Register the Clusters, Attributes and command callback processing functions for a port.

- Cluster initialization:

```
zcl_init()
```

Used to register callback functions for basic command processing supported by the ZCL layer, such as ZCL_CMD_WRITE/ZCL_CMD_WRITE_RSP/ZCL_CMD_READ/ZCL_CMD_READ_RSP etc.

3) **ev_on_poll()**

Register user polling events.

4.4.2 BDB process

Complete a series of operations of node network initialization, network building and network entry.

4.4.2.1 BDB initialization (bdb_init)

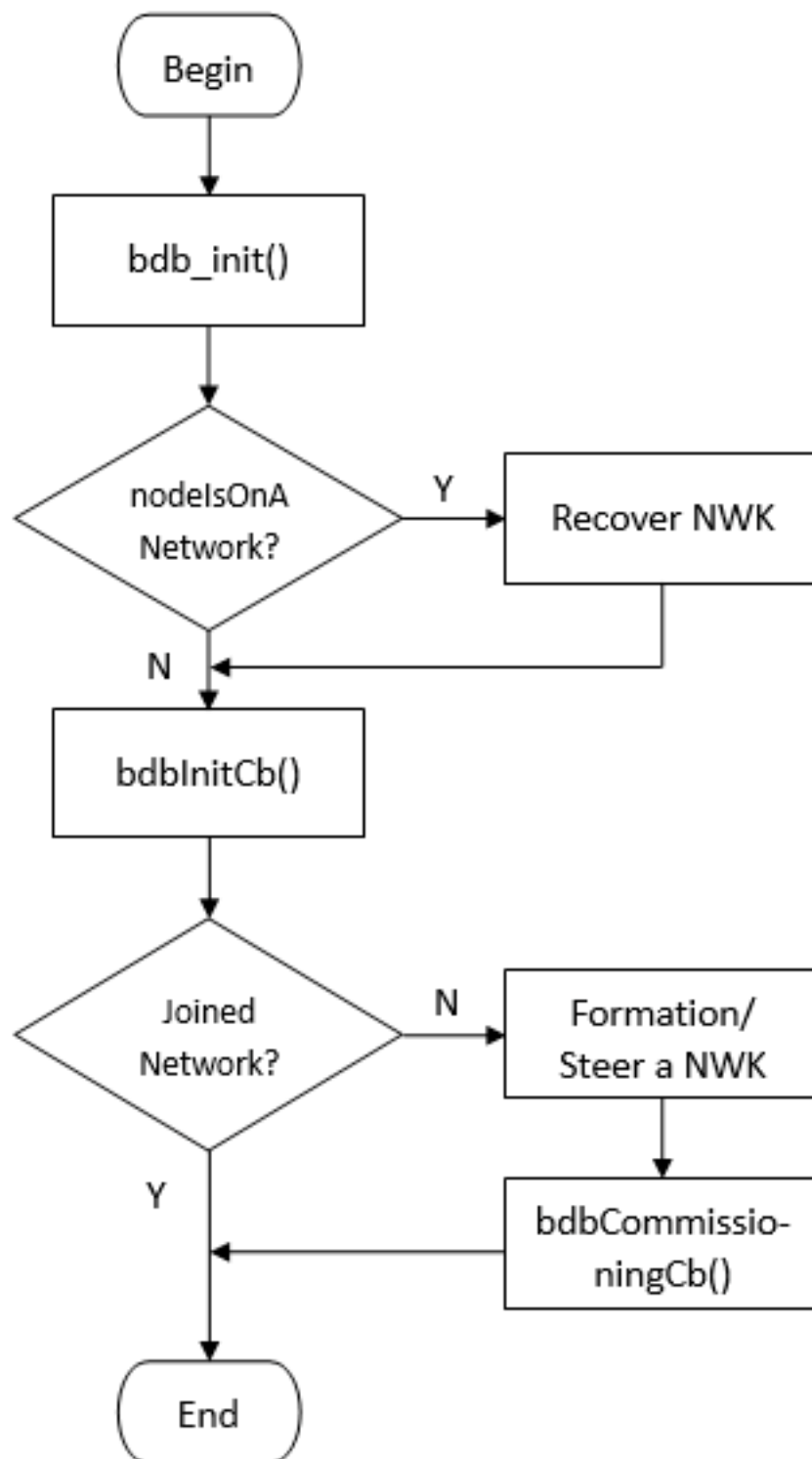


Figure 4.2: BDB initialization flow

The function of bdb_init(): complete related protocol functions in Chapter 7 of ZigBee Base Device Behavior Specification, and complete the simple initial configuration work for the unnetworked devices; for the

networked devices, reconnect to the network.

After BDB initialization, the callback function `bdbInitCb` registered by the application layer will be invoked, and then the user will invoke the relevant function to determine the subsequent BDB Commissioning behavior.

4.4.2.2 BDB Commissioning

Completing the protocol functions associated with Chapter 8 of ZigBee Base Device Behavior Specification, BDB Commissioning includes the following types of behavior.

- Network steering: Search for discoveries and join in a network.
- Network formation: New device, Router or Coordinator, forms a Distribute network or a Central network respectively.
- Finding & Binding: Search for matching functions after joining a network and bind the method of sending data.
- Touch Link: Access to the network through touch link.

After executing BDB Commissioning, the system will call back the `bdbCommissioningCb` function to return the result of commissioning to the user, who will determine the subsequent action.

For example, if an End Device returns a status of `BDB_COMMISSION_STA_NO_NETWORK` after performing BDB Commissioning, then the user can determine whether to enter hibernation or continue attempting to search to join a network.

4.4.3 Network security

When a new device is first connected to a network, it requests a network key from the Trust Center (generally the coordinator gateway), and later encrypts over-the-air data for messaging with that key.

How to secure the network key? The following methods are commonly used.

- 1) Use the Default TCLK to encrypt the network key.
- 2) Use the Link key derived from Install code to encrypt the network key.
- 3) Pre-configure the NWK key.

4.4.3.1 Network access process

The new device can either access the Trust Center directly (Figure 4.3 Direct access to the network) or through other routing nodes (Figure 4.4 Indirect access to the network). During the process of accessing the network, the Trust Center will send the encrypted network key to the new device using the APS Transport Key command, and when the new device resolves successfully, it will broadcast a Device Announce command to notify its address information to the surrounding devices. If the new device is a prior-zigbee 3.0 protocol device, then the network entry is successful. On the contrary, Zigbee 3.0 compliant device, it will also send a Node Descriptor Request command to the Trust Center to check if the Trust Center supports Zigbee 3.0, and if so, it will continue to request a new Trust Center Link Key from the Trust Center.

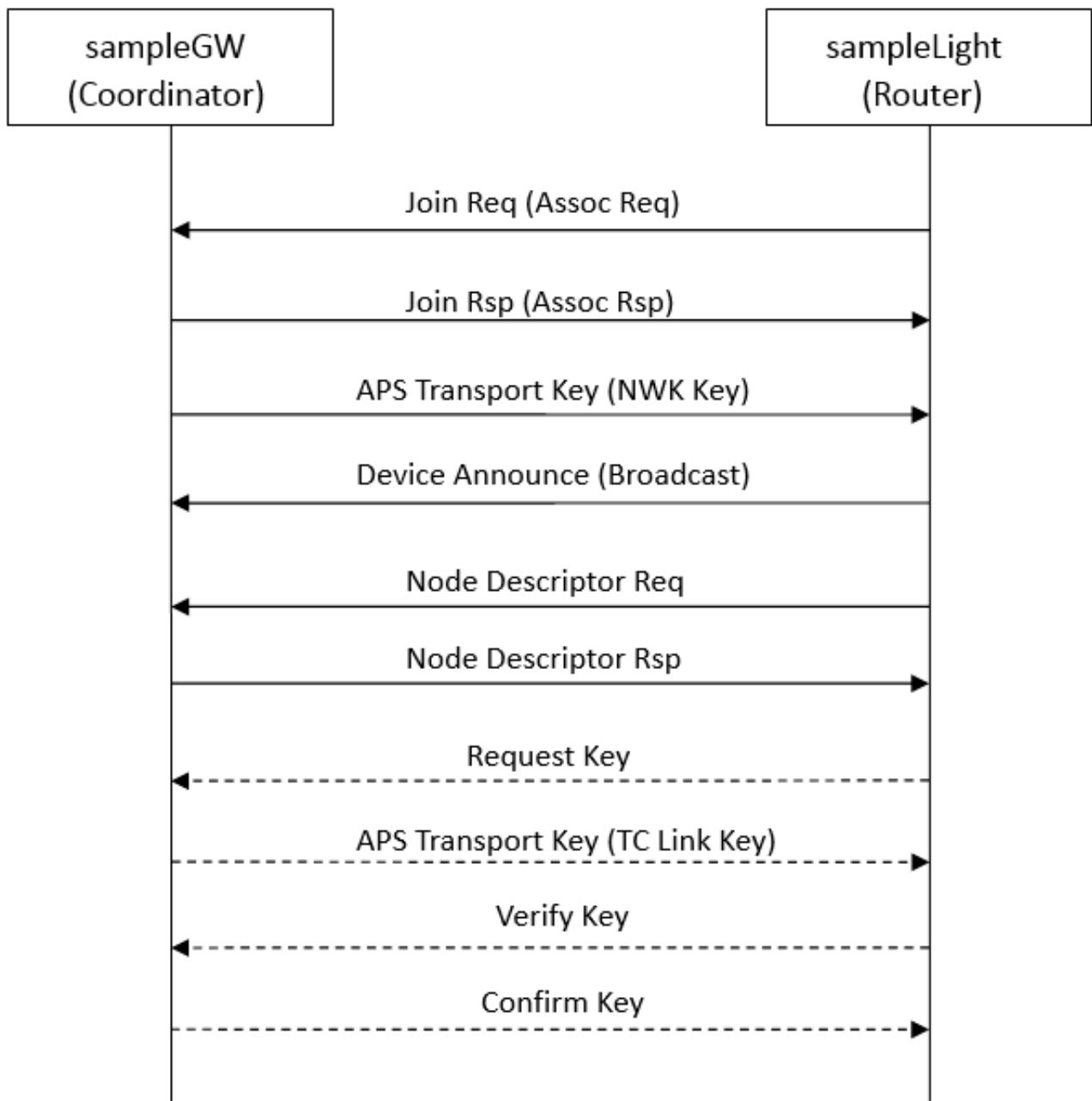


Figure 4.3: Direct access to the network

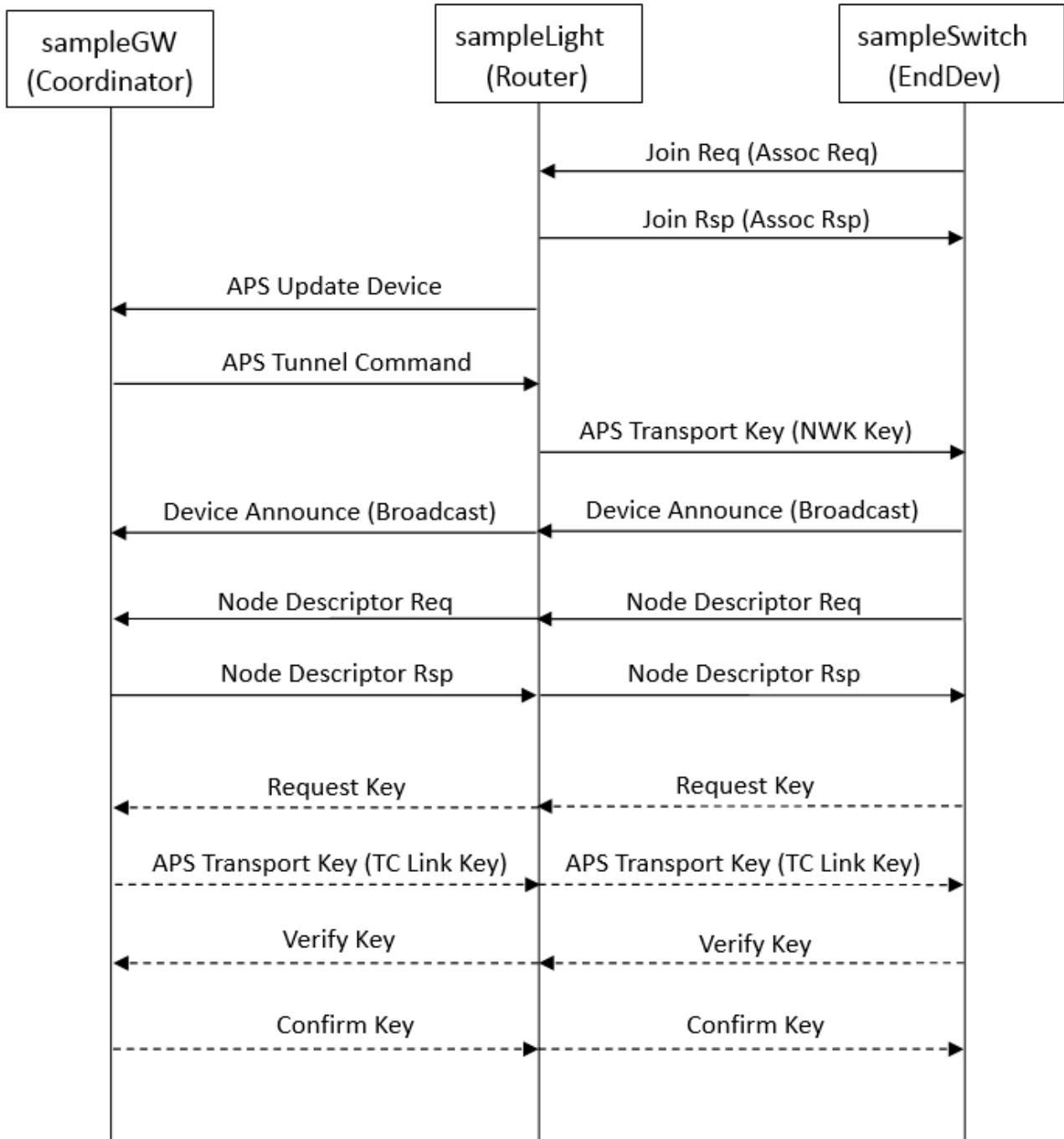


Figure 4.4: Indirect access to the network

4.4.3.2 Default TCLK

Default TCLK, a set of shared Trust Center Link Key specified by the Alliance, ensures both the security and interoperability of the Zigbee network. Telink Zigbee SDK uses Default TCLK to encrypt the network key by default, but since this is a shared link key, there are security risks for private networks.

```
const u8 tcLinkKeyCentralDefault[] = {  
    0x5a, 0x69, 0x67, 0x42, 0x65, 0x65, 0x41, 0x6c,  
    0x6c, 0x69, 0x61, 0x6e, 0x63, 0x65, 0x30, 0x39  
};
```

4.4.3.3 Install Code

Install Code is an encryption method defined by the Alliance to derive TCLK through the MMO hash algorithm (please refer to Zigbee base device behavior specification), and only when the trust center and the device to be enrolled have the same Install code, can the network be successfully connected.

If an install code is used, normally each node has a unique install code stored in a fixed location in Flash (see [section 2.5](#), pre-written during the production phase). When the device needs to access the gateway, the install code and IEEE address of this device are imported into the gateway.

The interface for the gateway to import the target device install code is as follows.

```
void zb_pre_install_code_store(addrExt_t ieeeAddr, u8 *pInstallCode)
```

The interface for the device to be accessed to load from Flash using the install code is as follows.

```
void zb_pre_install_code_load(app_linkkey_info_t *appLinkKey)
```

4.4.3.4 Pre-configured NWK Key

Pre-configured NWK Key is to configure the network key to the gateway and the device to be connected in advance, and the gateway sends an empty invalid network key to the connected device during the access process, and the accessed node ignores the nwk key field in the transport key command.

The interface is as follows, this function needs to be invoked after bdb_init().

```
void zb_preConfigNwkKey(u8 *nwkKey, bool enTransKey)
```

4.4.4 Data interaction

Once the network is successfully established, the nodes of the same network can transmit and receive data with each other.

Users can use the standard ZCL for data interaction or directly adopt the associated API to customize the processing of transmitted and received data.

4.4.4.1 Data interaction using ZCL layer

The Zigbee Alliance standardizes the basic commands such as Read/Write/Report/Configure Report of the ZCL standard, as well as processing of Cluster, Attribute and Command for different applications.

The Telink Zigbee SDK gives most of the ZCL implementation methods in source code for users to invoke directly or add their own extensions. Several demos provided by the SDK adopt the ZCL methods for data interaction, please refer to ZigBee Cluster Library Specification for details.

Take sampleLight as an example to illustrate the implementation method.

1) Register the AF layer port description and ZCL receive processing callback function `zcl_rx_handler` and send confirmation callback function `af_dataSentConfirm`.

```
/* Register endPoint */
af_endpointRegister(SAMPLE_LIGHT_ENDPOINT,
                   (af_simple_descriptor_t*)&sampleLight_simpleDesc,
                   zcl_rx_handler, af_dataSentConfirm );
```

2) Register the ZCL basic command processing callback function `zclProcessIncomingMsg`.

```
/* Register Incoming ZCL Foundation command/response messages */
zcl_init(sampleLight_zclProcessIncomingMsg);
```

3) Register the callback function table for command processing corresponding to Clusters that user needs to support, i.e. the registration table mentioned in [section 4.4.1](#).

```
/* Register ZCL specific cluster information */
zcl_register(SAMPLE_LIGHT_ENDPOINT,
            SAMPLELIGHT_CB_CLUSTER_NUM,
            g_sampleLightClusterList);
```

4.4.4.2 Data interaction using AF layer

In addition to sending and receiving data directly using the ZCL specification, users can directly register custom receive processing functions or send data to other devices directly using the `af_dataSend` function provided by the AF layer. In fact, the ZCL specification is a secondary encapsulation of the AF layer data processing.

If not using `zcl` in SDK, there is no need to invoke `zcl_register()` and `zcl_init()`.

Implementation method is as below.

1) Register the data reception processing callback function `user_rx_handler` and the send confirmation callback function `af_dataSentConfirm`.

```
af_endpointRegister(SAMPLE_LIGHT_ENDPOINT,
                   (af_simple_descriptor_t*)&sampleLight_simpleDesc,
                   user_rx_handler, NULL);
```

2) Invoke `af_dataSend` to send data.

```
u8 af_dataSend(u8 srcEp, epInfo_t *pDstEpInfo, u16 clusterId,  
              u8 cmdPldLen, u8 *cmdPld, u8 *seqNo);
```

4.4.5 Network parameters configuration

Users can modify the network parameters configuration in /zigbee/common/zb_config.c according to their requirements.

- TL_ZB_NWK_ADDR_MAP_NUM

The maximum number of address mapping tables supported by the node is related to the maximum network capacity. According to the default configuration of the SDK, 512KB flash supports maximum of 127 and 1MB flash supports a maximum of 300.

- TL_ZB_NEIGHBOR_TABLE_NUM

The number of neighbor tables supported by the node. The default for routing devices is 26.

- ROUTING_TABLE_NUM

The number of AODV routing tables supported by the node.

- NWK_ROUTE_RECORD_TABLE_NUM

Number of MTO routing record tables supported by the node.

- APS_BINDING_TABLE_NUM

The number of binding tables supported by the node.

- APS_GROUP_TABLE_NUM

The number of groups supported by the node.

- nwkKeyDefault

Network key default value. When the default value is 0, the coordinator will randomly generate the network key for distribution to the incoming devices; when the default value is non-zero, the coordinator will use nwkKeyDefault as the network key.

- macPibDefault

The basic parameter information of the MAC layer.

- nwkNibDefault

The basic parameter information of the NWK layer.

4.4.6 System exception processing

Invoke sys_exceptHandlerRegister() to register the exception callback function, triggered when buffer leak or status exception occurs, user can add the corresponding exception processing in this function.

Suggestion: In the development stage, the callback function directly use while(1) in order to locate the problem, and the variable T_evtExcept[1] can locate what kind of exception has occurred; in the product stage, it is better to do the reset process.

For example:

```
volatile u16 T_debug_except_code = 0;

static void sampleLightSysException(void){
    T_debug_except_code = T_evtExcept[1];
    while(1);//or SYSTEM_RESET();
}

/* Register except handler for test */
sys_exceptHandlerRegister(sampleLightSysException);
```

Telink Semiconductor

5 Commonly Used APIs

5.1 Application Framework (AF)

5.1.1 af_endpointRegister()

Register the endpoint descriptor and the received and sent result callback processing functions for that endpoint at the application framework layer.

Prototype

```
bool af_endpointRegister(u8 ep, af_simple_descriptor_t *simple_desc,
                        af_endpoint_cb_t rx_cb, af_dataCnf_cb_t cnfCb)
```

Return value

TRUE or FALSE.

Name	Type	Description
ep	u8	Endpoint.
simple_desc	af_simple_descriptor_t	Pointer to simple descriptor.
rx_cb	af_endpoint_cb_t	Callback function to handle the APS data sent to this endpoint.
cnfCb	af_dataCnf_cb_t	Confirm callback function for APS data transmission.

5.1.2 af_dataSend()

Use the AF layer to send data.

Prototype

```
u8 af_dataSend(u8 srcEp, epInfo_t *pDstEpInfo, u16 clusterId,
               u16 cmdPldLen, u8 *cmdPld, u8 *apsCnt)
```

Return value

RET_OK or Others.

Name	Type	Description
srcEp	u8	Source endpoint.
pDstEpInfo	epInfo_t*	Pointer to destination information.

Name	Type	Description
clusterId	u16	Cluster identifier.
cmdPldLen	u16	Payload length.
cmdPld	u8*	Pointer to the payload.
apsCnt	u8*	Pointer to the APS Counter.

5.2 Base Device Behaviour (BDB)

5.2.1 bdb_init()

Initialize the BDB and register the callback processing function.

Prototype

```
u8 bdb_init(af_simple_descriptor_t *simple_desc,
            bdb_commissionSetting_t *setting,
            bdb_appCb_t *cb, u8 repower)
```

Return value

TRUE

Name	Type	Description
simple_desc	af_simple_descriptor_t*	Pointer to the simple descriptor.
setting	bdb_commissionSetting_t*	Pointer to the BDB commissioning setting.
cb	bdb_appCb_t*	Callback function for BDB commissioning.
repower	u8	This function is called after power-on or sleep wakeup.

5.2.2 bdb_networkFormationStart()

Initiating the creation of a network will create a centralized network if invoked by the coordinator, and a distributed network if invoked by the router device.

Prototype

```
u8 bdb_networkFormationStart(void)
```

Return value

BDB_STATE

5.2.3 bdb_networkSteerStart()

Start searching a network.

Prototype

```
u8 bdb_networkSteerStart(void)
```

Return value

BDB_STATE

5.2.4 bdb_networkTouchLinkStart()

Start Touch Link.

Prototype

```
u8 bdb_networkTouchLinkStart(u8 role)
```

Return value

BDB_STATE

Name	Type	Description
role	u8	BDB_COMMISSIONING_ROLE_INITIATOR or BDB_COMMISSIONING_ROLE_TARGET.

5.2.5 bdb_findAndBindStart()

Start finding and binding.

Prototype

```
u8 bdb_findAndBindStart(u8 role)
```

Return value

BDB_STATE

Name	Type	Description
role	u8	BDB_COMMISSIONING_ROLE_INITIATOR or BDB_COMMISSIONING_ROLE_TARGET.

5.2.6 bdb_defaultReportingCfg()

Configure the default reporting message, which takes effect after binding.

Prototype

```
status_t bdb_defaultReportingCfg(u8 endpoint, u16 profileID,
                                u16 clusterID, u16 attrID,
                                u16 minReportInt, u16 maxReportInt,
                                u8 *reportableChange)
```

Return value

ZCL_STA

Name	Type	Description
endpoint	u8	Endpoint.
profileID	u16	Profile Identifier.
clusterID	u16	Cluster Identifier.
attrID	u16	Attribute Identifier.
minReportInt	u16	Minimum reporting interval, in seconds.
maxReportInt	u16	Maximum reporting interval, in seconds.
reportableChange	u8*	Reportable change (only applicable to analog data type).

5.3 Network management

5.3.1 zb_init()

Initialize the layers of the Zigbee protocol stack.

Prototype

```
void zb_init(void)
```

Return value

None

5.3.2 zb_zdoCbRegister()

Register the protocol stack callback function.

Prototype

```
void zb_zdoCbRegister(zdo_appIndCb_t *cb)
```

Return value

None

Name	Type	Description
cb	zdo_appIndCb_t*	Callback function.

5.3.3 zb_setPollRate()

Set the sending rate of data request, in milliseconds. It is valid only for the end device after entering the network.

Prototype

```
u8 zb_setPollRate(u32 newRate)
```

Return value

zdo_status_t

Name	Type	Description
newRate	u32	New poll rate, in millisecond.

5.3.4 zb_rejoinReq()

A network rejoin request, usually invoke after the end device has lost its parent node.

Prototype

```
zdo_status_t zb_rejoinReq(u32 scanChannels, u8 scanDuration)
```

Return value

zdo_status_t

Name	Type	Description
method	u32	Channel mask.
chm	u38	Scan duration.

5.3.5 zb_factoryReset()

Reset to factory settings. It is used for routers and end devices. It will broadcast leave-network notifications before restoring factory settings.

Prototype

```
void zb_factoryReset(void)
```

Return value

None

5.3.6 zb_mgmtPermitJoinReq()

Control command of permitting to join network. The coordinator or router device can use this command to control the time allowed for other devices to join the network.

Prototype

```
zdo_status_t zb_mgmtPermitJoinReq(u16 dstNwkAddr, u8 permitJoinDuration,
                                   u8 tcSignificance, u8 *seqNo,
                                   zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
permitJoinDuration	u8	Time in seconds during which the device allows to join.
tcSignificance	u8	TC significance.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	NULL.

5.3.7 zb_mgmtLeaveReq()

Control command of leaving network.

Prototype

```
zdo_status_t zb_mgmtLeaveReq(u16 dstNwkAddr,  
                             zdo_mgmt_leave_req_t *pReq,  
                             u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_leave_req_t*	Address information of the device that needs to be leaved.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Mgmt_Leave_rsp.

5.3.8 zb_mgmtLqiReq()

Get the associated neighbor information of the target device.

Prototype

```
zdo_status_t zb_mgmtLqiReq(u16 dstNwkAddr,  
                            zdo_mgmt_lqi_req_t *pReq,  
                            u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_lqi_req_t*	Address information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Mgmt_Lqi_rsp.

5.3.9 zb_mgmtBindReq()

Get the binding table information of the target device.

Prototype

```
zdo_status_t zb_mgmtBindReq(u16 dstNwkAddr,
                           zdo_mgmt_bind_req_t *pReq,
                           u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_bind_req_t*	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Mgmt_Bind_rsp.

5.3.10 zb_mgmtNwkUpdateReq()

Update network parameters or request network environment information.

Prototype

```
zdo_status_t zb_mgmtNwkUpdateReq(u16 dstNwkAddr,
                                  zdo_mgmt_nwk_update_req_t *pReq,
                                  u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_mgmt_nwk_update_req_t*	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.

Name	Type	Description
indCb	zdo_callback	Callback function for Mgmt_Nwk_Update_notify.

5.3.11 zb_zdoBindUnbindReq()

Send the bind or unbind command.

Prototype

```
zdo_status_t zb_zdoBindUnbindReq(bool isBinding, zdo_bind_req_t *pReq,
                                   u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
isBinding	bool	TRUE - bind, FALSE - unbind.
pReq	zdo_bind_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Bind_rsp/Unbind_rsp.

5.3.12 zb_zdoNwkAddrReq()

Request target short address command.

Prototype

```
zdo_status_t zb_zdoNwkAddrReq(u16 dstNwkAddr, zdo_nwk_addr_req_t *pReq,
                               u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_nwk_addr_req_t *	Information of the device that needs to be requested.

Name	Type	Description
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for NWK_addr_rsp.

5.3.13 zb_zdoIeeeAddrReq()

Request target long address command.

Prototype

```
zdo_status_t zb_zdoIeeeAddrReq(u16 dstNwkAddr, zdo_ieee_addr_req_t *pReq,
                                u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_ieee_addr_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for IEEE_addr_rsp.

5.3.14 zb_zdoSimpleDescReq()

Request information of the target simple descriptor.

Prototype

```
zdo_status_t zb_zdoSimpleDescReq(u16 dstNwkAddr,
                                   zdo_simple_descriptor_req_t *pReq,
                                   u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.

Name	Type	Description
pReq	zdo_simple_descriptor_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Simple_Desc_rsp.

5.3.15 zb_zdoNodeDescReq()

Request information about the target node descriptor.

Prototype

```
zdo_status_t zb_zdoNodeDescReq(u16 dstNwkAddr,
                                zdo_node_descriptor_req_t *pReq,
                                u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_node_descriptor_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Node_Desc_rsp.

5.3.16 zb_zdoPowerDescReq()

Requests information about the target power descriptor.

Prototype

```
zdo_status_t zb_zdoPowerDescReq(u16 dstNwkAddr,
                                  zdo_power_descriptor_req_t *pReq,
                                  u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_power_descriptor_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Power_Desc_rsp.

5.3.17 zb_zdoActiveEpReq()

Request information about the target's active endpoints.

Prototype

```
zdo_status_t zb_zdoActiveEpReq(u16 dstNwkAddr, zdo_active_ep_req_t *pReq,
                               u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_active_ep_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Active_Ep_rsp.

5.3.18 zb_zdoMatchDescReq()

Request information of target match descriptor.

Prototype

```
zdo_status_t zb_zdoMatchDescReq(u16 dstNwkAddr,
                                 zdo_match_descriptor_req_t *pReq,
                                 u8 *seqNo, zdo_callback indCb)
```

Return value

zdo_status_t

Name	Type	Description
dstNwkAddr	u16	Short address of the target device.
pReq	zdo_match_descriptor_req_t *	Information of the device that needs to be requested.
seqNo	u8*	The sequence number used by this command.
indCb	zdo_callback	Callback function for Match_Desc_rsp.

5.3.19 zb_isDeviceFactoryNew()

Check if the node is a new device.

Prototype

```
bool zb_isDeviceFactoryNew(void)
```

Return value

TRUE or FALSE

5.3.20 zb_isDeviceJoinedNwk()

Check if the node is already in the network.

Prototype

```
bool zb_isDeviceJoinedNwk(void)
```

Return value

TRUE or FALSE

5.3.21 zb_getMacAssocPermit()

Get the network access permission status of this node.

Prototype

```
bool zb_getMacAssocPermit(void)
```

Return value

TRUE or FALSE

5.3.22 zb_apsExtPanidSet()

Set the Extend PAN ID.

Prototype

```
void zb_apsExtPanidSet(extPANId_t panId)
```

Return value

None

Name	Type	Description
panId	extPANId_t	Extend PAN identifier.

5.4 ZCL

5.4.1 zcl_init()

Initialize the ZCL layer and register the foundation command processing function.

Prototype

```
void zcl_init(zcl_hookFn_t fn)
```

Return value

None

Name	Type	Description
fn	zcl_hookFn_t	Callback function.

5.4.2 zcl_register()

Register the clusters, attributes and commands supported by the application layer endpoints.

Prototype

```
void zcl_register(u8 endpoint, u8 clusterNum, zcl_specClusterInfo_t *info)
```

Return value

None

Name	Type	Description
endpoint	u8	Endpoint.
clusterNum	u8	The total number of clusters supported by the endpoint.
info	zcl_specClusterInfo_t*	Information of the clusters.

5.5 OTA

5.5.1 ota_init()

Initialize the endpoint information and callback functions needed by OTA.

Prototype

```
void ota_init(ota_type_e type,
             af_simple_descriptor_t *simpleDesc,
             ota_preamble_t *otaPreamble,
             ota_callBack_t *cb)
```

Return value

None

Name	Type	Description
type	ota_type_e	Client or server.
simpleDesc	af_simple_descriptor_t*	Simple descriptor.
otaPreamble	ota_preamble_t*	Information of the OTA preamble.
cb	ota_callBack_t*	Callback function for OTA message.

5.5.2 ota_queryStart()

Start the OTA query function in seconds.

Prototype

```
void ota_queryStart(u16 seconds)
```

Return value

None

Name	Type	Description
seconds	u8	Query cycle, in second

Telink Semiconductor

6 OTA

The TLSR8 and TLSR9 series chips support flash multi-address booting: in addition to Flash address 0x00000, they also support reading firmware from 0x40000. This features is used by Telink Zigbee SDK to implement OTA function.

As you can see from [section 2.5](#), we have allocated two firmware areas Firmware and OTA-Image, and the firmware size should not exceed 208KB (512KB flash).

Assuming that Firmware is currently running, when the device performs an OTA upgrade, the new firmware data will be stored in the OTA-Image. After the OTA is completed and verified, it will reboot and run the firmware in the OTA-Image. Subsequent OTAs will be executed alternately.

6.1 OTA initialization

```
typedef enum{
    OTA_TYPE_CLIENT,
    OTA_TYPE_SERVER
}ota_type_e;

void ota_init(ota_type_e type, af_simple_descriptor_t *simpleDesc,
             ota_preamble_t *otaPreamble, ota_callBack_t *cb);
```

OTA devices include service devices (Server) and end devices (Client). Therefore, it is necessary to pay attention to the type of service to be initialized by OTA during OTA initialization.

Generally, the device being upgraded is the end device (Client) and the device providing the new firmware to the upgraded device is the service device (Server).

6.2 OTA Server

The OTA server needs to write the new firmware required by the upgraded device to the OTA-Image area for OTA use.

For example, if the firmware run by the server itself is in the Firmware area, then the OTA image of the target device (new firmware with OTA Header, refer to Zigbee Cluster Library Specification) can be temporarily stored in the OTA-Image area.

After the OTA server invokes the ota_init function, it will automatically load the information of the OTA image.

6.3 OTA Client

Set the OTA query period for OTA Client in seconds.

```
void ota_queryStart(u16 seconds);
```

After invoking the `ota_init` function, the OTA Client will automatically reload the last OTA progress information. If joining in a network successfully, it will start the OTA request after the set seconds.

6.4 OTA Image

The OTA Image is a Zigbee file containing OTA Header for over-the-air upgrade on the OTA client device. The user can use the `zigbee_ota_tool` tool provided by Telink to convert the firmware on the client device into an upgrade file. The SDK requires that the file version of the firmware to be upgraded (please refer to [section 2.3.3](#)) should be greater than the current file version, otherwise, when client device initiates OTA query, the server device will reply `NO_IMAGE_AVAILABLE`.

For the OTA Image file generation, take `sampleLight_8258` for example.

- (1) Copy `sampleLight_8258.bin` to the folder where "`zigbee_ota_tool_v2.2.exe`" is located.
- (2) Double-click "`zigbee_ota_tool_v2.2.exe`" and follow the prompts.
- (3) Or use the command line "`./zigbee_ota_tool_v2.2.exe [arg1] [arg2]`".

[arg1] file name, the name of the file to be translated, input parameters such as: `sampleLight_8258.bin`.

[arg2] If this parameter is not present, it means no encryption. If it is present, it means AES-128 encryption is used. Enter parameters such as: `00 11 22 33 44 55 66 77 88 99 AA BB CC DD EE FF`.

Output file:

`1141-0201-10013001-sampleLight_8258.zigbee`

File Import:

- (1) Change the file extension of the output file from `".zigbee"` to `".bin"`, and use the Telink BDT.exe programming tool to directly download the file to the OTA-Image area in the Flash of the OTA server (gateway).
- (2) Change the file extension of the output file from `".zigbee"` to `".ota"`. Use the ZGC tool and select "HCI OTA" to import the file into the OTA-Image area in the Flash of the OTA server (gateway). This method requires enabling `ZBHCI_EN` on the OTA server (gateway) for it to work.



Figure 6.1: File import

7 Bootloader

Telink Zigbee SDK also provides a bootloader project, allowing users to implement product functionality using the bootloader + app mode.

The bootloader is a simple boot program that implements functions such as application upgrade (requires UART_ENABLE), checking, copying, and jumping. The app is a Zigbee application program, which can be sampleGW, sampleLight, sampleSwitch, or the user's own application program.

When compiling the bootloader project, it is necessary to define BOOT_LOADER_MODE as 1; otherwise, compilation will result in an error. When compiling the app project, BOOT_LOADER_MODE also needs to be defined as 1, otherwise it will not be able to run properly due to mismatched linker files.

After compiling the bootloader and app projects to generate the bin files, users can refer to [section 2.4.3](#) and [section 2.6](#), and directly use the BDT burning tool to burn the two bin files to the corresponding addresses. You can burn the bootloader only, then use the serial upgrade function of the bootloader (requires UART_ENABLE) to upgrade the app to the firmware area through the serial port.

7.1 Application Upgrade

1) Serial port enable:

```
#define UART_ENABLE      1
```

2) Serial port pin configuration:

```
#define UART_TX_PIN      UART_TX_PD7
#define UART_RX_PIN      UART_RX_PA0
```

The pin configuration may vary on different chips and development platforms. The specific configurations can be viewed in the \bootLoader\board_xx.h file.

3) Serial port baud rate:

It is 115200 bps by default.

For upgrade method, take sampleLight_8258 as an example:

- (1) Pre-burn the bootloader onto the development board and connect the serial port.
- (2) Prepare the sampleLight_8258.bin in advance. Unlike the previous [section 6](#), it doesn't need to be converted into an OTA image upgrade file.
- (3) Power on the development board and quickly press the VK_SW1 button to trigger the upgrade mode.
- (4) Open the ZGC tool, select the "HCI OTA" tab, choose "sampleLight_8258.bin", and click "Start OTA" to start the upgrade.
- (5) After the progress bar reaches full, the bootloader will check and transfer the new application program before jumping to run the app program.

Note:

If the serial port upgrade function is enabled, the upgrade function will only be triggered within 2 seconds after powering on (when the LED_POWER light is on). The trigger conditions are either detecting that the VK_SW1 button is pressed within the 2 seconds or receiving the serial command MSG_CMD_OTA_START_REQUEST. If the window period is missed, a re-power-up is required.



Figure 7.1: Bootloader upgrade

8 Extended Function HCI Interface

Telink Zigbee SDK provides HCI interface for user to do extended application development, please refer to ../zigbee/zbhci for details.

The current HCI interface implementation provides 3 methods: UART, USB CDC and USB Print. The corresponding macros are as follows.

```
UART      -> #define ZBHCI_UART 1
USB CDC   -> #define ZBHCI_USB_CDC 1
USB Print -> #define ZBHCI_USB_PRINT 1
```

To use a certain method, set the corresponding macro to 1.

For example, if the demo sampleGW needs to use the HCI function of the UART mode, please set the macro ZBHCI_UART to 1.

8.1 Control flow chart

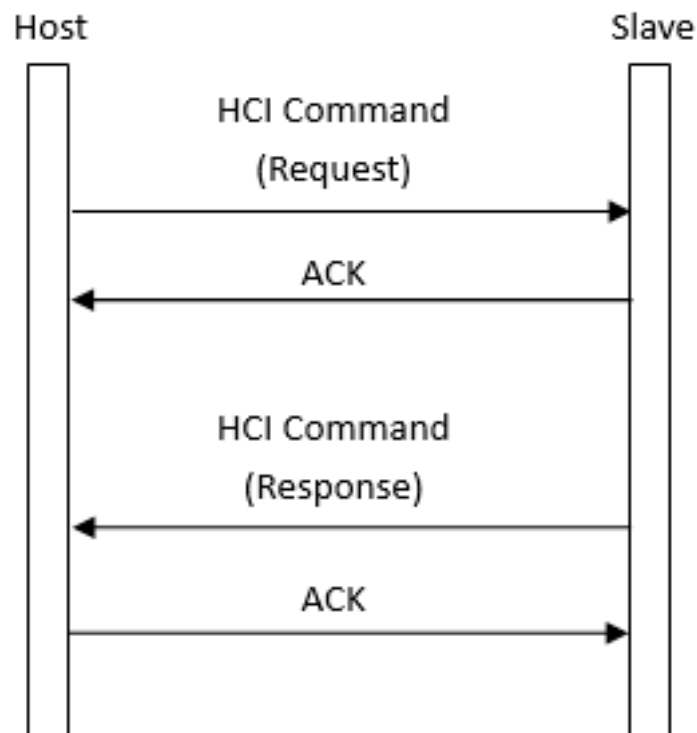


Figure 8.1: Control flow chart

8.2 Command frame format

Header	Message Type	Message Length	Checksum	Payload	Tail
1 Byte	2 Bytes	2 Bytes	1 Byte	Variable	1 Byte

Field	Description
Header	Header flag, shall be 0x55.
Message Type	Message Type, big-endian.
Message Length	Payload length, big-endian.
Checksum	The checksum.
Payload	Payload.
Tail	Tail flag, shall be 0xAA.

Note:

- All fields of the HCI command are in big-endian mode.

8.3 Acknowledge format

8.3.1 Message Type

Message Type	Value
ZBHCI_CMD_ACKNOWLEDGE	0x8000

8.3.2 Payload

Message Type	Status	Reserved
2 Bytes	1 Byte	1 Byte

Name	Description
Message Type	HCI command message type.

Name	Description
Status	0 = Success; 1 = Wrong parameter; 2 = Unsupported command; 3 = Busy; 4 = No memory.
Reserved	0

Example:

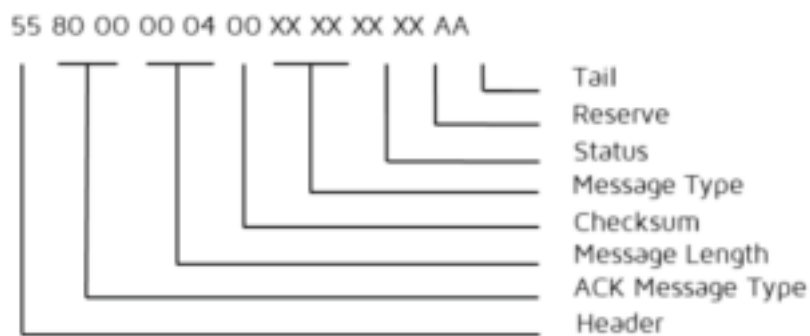


Figure 8.2: Command frame format

8.4 BDB commands

8.4.1 Message Type

Message Type	Value
ZBHCI_CMD_BDB_COMMISSION_FORMATION	0x0001
ZBHCI_CMD_BDB_COMMISSION_STEER	0x0002
ZBHCI_CMD_BDB_COMMISSION_TOUCHLINK	0x0003
ZBHCI_CMD_BDB_COMMISSION_FINDBIND	0x0004
ZBHCI_CMD_BDB_FACTORY_RESET	0x0005
ZBHCI_CMD_BDB_PRE_INSTALL_CODE	0x0006
ZBHCI_CMD_BDB_CHANNEL_SET	0x0007

8.4.2 Payload

1) ZBHCI_CMD_BDB_COMMISSION_FORMATION

No payload.

Example: 55 00 01 00 00 00 AA

2) ZBHCI_CMD_BDB_COMMISSION_STEER

No payload.

Example: 55 00 02 00 00 00 AA

3) ZBHCI_CMD_BDB_COMMISSION_TOUCHLINK

Role

1 Byte

Name	Description
Role	1 = Touch link initiator; 2 = Touch link target.

Example: 55 00 03 00 01 00 XX AA

4) ZBHCI_CMD_BDB_COMMISSION_FINDBIND

Role

1 Byte

Name	Description
Role	1 = Find & Bind initiator; 2 = Find & Bind target.

Example: 55 00 04 00 01 00 XX AA

5) ZBHCI_CMD_BDB_FACTORY_RESET

No payload.

Example: 55 00 05 00 00 00 AA

6) ZBHCI_CMD_BDB_PRE_INSTALL_CODE

devAddr	pre-install code
---------	------------------

8 Bytes	16 Bytes
---------	----------

Name	Description
devAddr	The IEEE address of the pre-configured device.
pre-install code	The pre-install code to generate link key for the device.

Example: 55 00 06 00 18 00 XX[8 Bytes] XX[16 Bytes] AA

7) ZBHCI_CMD_BDB_CHANNEL_SET

channelIdx

1 Byte

Name	Description
channelIdx	11~26

Example: 55 00 07 00 01 00 XX AA

8.5 Network management command

8.5.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_DISCOVERY_NWK_ADDR_REQ	0x0010
ZBHCI_CMD_DISCOVERY_IEEE_ADDR_REQ	0x0011
ZBHCI_CMD_DISCOVERY_NODE_DESC_REQ	0x0012
ZBHCI_CMD_DISCOVERY_SIMPLE_DESC_REQ	0x0013
ZBHCI_CMD_DISCOVERY_MATCH_DESC_REQ	0x0014
ZBHCI_CMD_DISCOVERY_ACTIVE_EP_REQ	0x0015
ZBHCI_CMD_BIND_REQ	0x0020

Message Type	Value
ZBHCI_CMD_UNBIND_REQ	0x0021
ZBHCI_CMD_MGMT_LQI_REQ	0x0030
ZBHCI_CMD_MGMT_BIND_REQ	0x0031
ZBHCI_CMD_MGMT_LEAVE_REQ	0x0032
ZBHCI_CMD_MGMT_DIRECT_JOIN_REQ	0x0033
ZBHCI_CMD_MGMT_PERMIT_JOIN_REQ	0x0034
ZBHCI_CMD_MGMT_NWK_UPDATE_REQ	0x0035
ZBHCI_CMD_NODES_JOINED_GET_REQ	0x0040
ZBHCI_CMD_NODES_TOGGLE_TEST_REQ	0x0041
ZBHCI_CMD_GET_LOCAL_NWK_INFO_REQ	0x0045

8.5.2 Payload (Host)

1) ZBHCI_CMD_DISCOVERY_NWK_ADDR_REQ

dstAddr	ieeeAddr	reqType	startIdx
2 Bytes	8 Bytes	1 Byte	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.
ieeeAddr	The IEEE address to be matched by the remote device.
reqType	Request type for this command: 0x00 – Single device response; 0x01 – Extended response; 0x02 ~ 0xFF – Reserved.
startIdx	If the request type for this command is extended response, the startIdx provides the starting index for the requested elements of the associated device list.

Example: 55 00 10 00 0C 00 XX[2 Bytes] XX[8 Bytes] XX XX AA

2) ZBHCI_CMD_DISCOVERY_IEEE_ADDR_REQ

dstAddr	nwkAddrOfInterest	reqType	startIdx
2 Bytes	2 Bytes	1 Byte	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast or broadcast to all devices for which macRxOnWhenIdle = TRUE.
nwkAddrOfInterest	The NWK address that is used for IEEE address mapping.
reqType	Request type for this command: 0x00 – Single device response; 0x01 – Extended response; 0x02 ~ 0xFF – Reserved.
startIdx	If the request type for this command is extended response, the startIdx provides the starting index for the requested elements of the associated device list.

Example: 55 00 11 00 06 00 XX[2 Bytes] XX[2 Bytes] XX XX AA

3) ZBHCL_CMD_DISCOVERY_NODE_DESC_REQ

dstAddr	nwkAddrOfInterest
2 Bytes	2 Bytes

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.
nwkAddrOfInterest	NWK address of the target.

Example: 55 00 12 00 04 00 XX[2 Bytes] XX[2 Bytes] AA

4) ZBHCL_CMD_DISCOVERY_SIMPLE_DESC_REQ

dstAddr	nwkAddrOfInterest	endpoint
2 Bytes	2 Bytes	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.
nwkAddrOfInterest	NWK address of the target.
endpoint	The endpoint on the destination.

Example: 55 00 13 00 05 00 XX[2 Bytes] XX[2 Bytes] XX AA

5) ZBHCI_CMD_DISCOVERY_MATCH_DESC_REQ

dstAddr	nwkAddrOfInterest	profileID	numInClusters	numOutClusters	clusterList
2 Bytes	2 Bytes	2 Bytes	1 Byte	1 Byte	n Bytes

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.
nwkAddrOfInterest	NWK address of the target.
profileID	Profile ID to be matched at the destination.
numInClusters	The number of input clusters provided for matching within the inClusterList.
numOutClusters	The number of output clusters provided for matching within the outClusterList.
clusterList	inClusterList + outClusterList. List of input cluster IDs to be used for matching, the inClusterList is the desired list to be matched by the Remote Device (the elements of the inClusterList are the supported output clusters of the Local Device). List of output cluster IDs to be used for matching, the outClusterList is the desired list to be matched by the Remote Device (the elements of the outClusterList are the supported input clusters of the Local Device).

Example: 55 00 14 msgLenH msgLenL 00 XX[2 Bytes] XX[2 Bytes] XX[2 Bytes] XX XX XX[n Bytes] AA

6) ZBHCI_CMD_DISCOVERY_ACTIVE_EP_REQ

dstAddr	nwkAddrOfInterest
2 Bytes	2 Bytes

Name	Description
dstAddr	The destination address to which this command will be sent, shall be unicast either to the remote device itself or to an alternative device that contains the discovery information of the remote device.
nwkAddrOfInterest	NWK address of the target.

Example: 55 00 15 00 04 00 XX[2 Bytes] XX[2 Bytes] AA

7) ZBHCL_CMD_BIND_REQ / ZBHCL_CMD_UNBIND_REQ

srcIEEEAddr	srcEndpoint	clusterID	dstAddrMode	dstAddr	dstEndpoint
8 Bytes	1 Byte	2 Bytes	1 Byte	2 or 8 Bytes	0 or 1 Byte

Name	Description
srcIEEEAddr	The IEEE address for the source.
srcEndpoint	The source endpoint for the binding entry.
clusterID	The identifier of the cluster on the source device that is bound to the destination.
dstAddrMode	The addressing mode for the destination address used in this command. 0x00 – Reserved; 0x01 – 16-bit group address for dstAddr and dstEndpoint not present; 0x02 – Reserved; 0x03 – 64-bit extended address for dstAddr and dstEndpoint present; 0x04 ~ 0xFF – Reserved.
dstAddr	The destination address for the binding entry.
dstEndpoint	Shall be present only if the dstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

Example: ZBHCL_CMD_BIND_REQ:

55 00 20 msgLenH msgLenL 00 XX[8 Bytes] XX XX[2 Bytes] XX XX[2 or 8 Bytes] XX[0 or 1 Byte] AA

ZBHCL_CMD_UNBIND_REQ:

55 00 21 msgLenH msgLenL 00 XX[8 Bytes] XX XX[2 Bytes] XX XX[2 or 8 Bytes] XX[0 or 1 Byte] AA

8) ZBHCL_CMD_MGMT_LQI_REQ

dstAddr	startIdx
---------	----------

2 Bytes	1 Byte
---------	--------

Name	Description
------	-------------

dstAddr	The destination address to which this command will be sent, shall be unicast.
---------	---

startIdx	Starting index for the requested elements of the neighbor table.
----------	--

Example: 55 00 30 00 03 00 XX[2 Bytes] XX AA

9) ZBHCL_CMD_MGMT_BIND_REQ

dstAddr	startIdx
---------	----------

2 Bytes	1 Byte
---------	--------

Name	Description
------	-------------

dstAddr	The destination address to which this command will be sent, shall be unicast.
---------	---

startIdx	Starting index for the requested elements of the binding table.
----------	---

Example: 55 00 31 00 03 00 XX[2 Bytes] XX AA

10) ZBHCL_CMD_MGMT_LEAVE_REQ

dstAddr	devAddr	rejoin	removeChildren
---------	---------	--------	----------------

2 Bytes	8 Bytes	1 Byte	1 Byte
---------	---------	--------	--------

Name	Description
------	-------------

dstAddr	The destination address to which this command will be sent, shall be unicast.
---------	---

devAddr	The IEEE address of the device to be removed or NULL if the device removes itself.
---------	--

Name	Description
rejoin	TRUE if the device is being asked to leave from the current parent and requested to rejoin the network. Otherwise, the parameter has a value of FALSE.
removeChildren	TRUE if the device being asked to leave the network is also being asked to remove its child device, if any. Otherwise, it has a value of FALSE.

Example: 55 00 32 00 0C 00 XX[2 Bytes] XX[8 Bytes] XX XX AA

11) ZBHCI_CMD_MGMT_PERMIT_JOIN_REQ

dstAddr	permitDuration	TC_significance
2 Bytes	1 Byte	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent.
permitDuration	The length of time in seconds during which the coordinator or router will allow associations. The value 0x00 and 0xff indicate that permission is disabled or enabled, respectively, without a specified limit.
TC_significance	This field shall always have a value of 1, indicating a request to change the Trust Center policy. If a frame is received with a value of 0, it shall be treated as having a value of 1.

Example: 55 00 34 00 04 00 XX[2 Bytes] XX XX AA

12) ZBHCI_CMD_MGMT_NWK_UPDATE_REQ

dstAddr	nwkManagerAddr	scanChannels	scanDuration	scanCount/ nwkUpdateId
2 Bytes	2 Bytes	4 Bytes	1 Byte	1 Byte

Name	Description
dstAddr	The destination address to which this command will be sent.
nwkManagerAddr	This field shall be present only if the scanDuration is set to 0xff, and, when present, it indicates the NWK address for the device with the Network Manager bit set in its Node Descriptor.

Name	Description
scanChannels	Channel bit mask, 32-bit field structure.
scanDuration	A value used to calculate the length of time to spend on scanning each channel. 0x00 ~ 0x05 or 0xfe or 0xff.
scanCount	This field represents the number of energy scans to be conducted and reported. This field shall be present only if the scanDuration is within the range of 0x00 to 0x05.
nwkUpdateId	The value of the nwkUpdateId contained in this request. This value is set by the Network Channel Manager prior to sending the message. This field shall only be present if the scanDuration is 0xfe or 0xff. If the scanDuration is 0xff, then the value in the nwkUpdateId shall be ignored.

Example: 55 00 35 00 0A 00 XX[2 Bytes] XX[2 Bytes] XX[4 Bytes] XX XX AA

13) ZBHCI_CMD_NODES_JOINED_GET_REQ

startIndex

2 Bytes

Name	Description
startIndex	Starting index for the requested elements of the joined node list.

Example: 55 00 40 00 02 00 XX[2 Bytes] AA

14) ZBHCI_CMD_NODES_TOGGLE_TEST_REQ

unicast/broadcast timerInterval

1 Byte

1 Byte

Name	Description
unicast/broadcast	0x00 – broadcast mode; 0x01 – unicast mode, send toggle command to each node in turn.
timerInterval	The timer interval of the data transmission. Unit: 10millisecond.

Example: 55 00 41 00 02 00 XX XX AA

15) ZBHCI_CMD_GET_LOCAL_NWK_INFO_REQ

No payload.

Example: 55 00 45 00 00 00 AA

8.5.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_DISCOVERY_NWK_ADDR_RSP	0x8010
ZBHCI_CMD_DISCOVERY_IEEE_ADDR_RSP	0x8011
ZBHCI_CMD_DISCOVERY_NODE_DESC_RSP	0x8012
ZBHCI_CMD_DISCOVERY_SIMPLE_DESC_RSP	0x8013
ZBHCI_CMD_DISCOVERY_MATCH_DESC_RSP	0x8014
ZBHCI_CMD_DISCOVERY_ACTIVE_EP_RSP	0x8015
ZBHCI_CMD_BIND_RSP	0x8020
ZBHCI_CMD_UNBIND_RSP	0x8021
ZBHCI_CMD_MGMT_LQI_RSP	0x8030
ZBHCI_CMD_MGMT_BIND_RSP	0x8031
ZBHCI_CMD_MGMT_LEAVE_RSP	0x8032
ZBHCI_CMD_MGMT_DIRECT_JOIN_RSP	0x8033
ZBHCI_CMD_MGMT_PERMIT_JOIN_RSP	0x8034
ZBHCI_CMD_MGMT_NWK_UPDATE_RSP	0x8035
ZBHCI_CMD_NODES_JOINED_GET_RSP	0x8040
ZBHCI_CMD_NODES_TOGGLE_TEST_RSP	0x8041
ZBHCI_CMD_NODES_DEV_ANNCIE_IND	0x8043
ZBHCI_CMD_GET_LOCAL_NWK_INFO_RSP	0x8045
ZBHCI_CMD_DATA_CONFIRM	0x8200
ZBHCI_CMD_NODE_LEAVE_IND	0x8202

8.5.4 Payload (Slave)

1) ZBHCI_CMD_DISCOVERY_NWK_ADDR_RSP

srcAddr	seqNum	status	ieeeAddr	nwkAddr	numAssocDev	startIdx	assocDevList
2 Bytes	1 Byte	1 Byte	8 Bytes	2 Bytes	0 or 1 Byte	0 or 1 Byte	0 or n Bytes

Name	Description
-------------	--------------------

srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
ieeeAddr	64-bit address for the Remote Device.
nwkAddr	16-bit address for the Remote Device.
numAssocDev	Count of the number of 16-bit short address to follow.
startIdx	Starting index into the list of associated devices for this report.
assocDevList	The list of associated devices.

Example: 55 80 10 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX[8 Bytes] XX[2 Bytes] {XX XX XX[n Bytes]} AA

2) ZBHCI_CMD_DISCOVERY_IEEE_ADDR_RSP

srcAddr	seqNum	status	ieeeAddr	nwkAddr	numAssocDev	startIdx	assocDevList
2 Bytes	1 Byte	1 Byte	8 Bytes	2 Bytes	0 or 1 Byte	0 or 1 Byte	0 or n Bytes

Name	Description
-------------	--------------------

srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
ieeeAddr	64-bit address for the Remote Device.
nwkAddr	16-bit address for the Remote Device.
numAssocDev	Count of the number of 16-bit short address to follow.
startIdx	Starting index into the list of associated devices for this report.
assocDevList	The list of associated devices.

Example: 55 80 11 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX[8 Bytes] XX[2 Bytes] {XX XX XX[n Bytes]} AA

3) ZBHCL_CMD_DISCOVERY_NODE_DESC_RSP

srcAddr	seqNum	status	nwkAddrOfInterest	nodeDesc
2 Bytes	1 Byte	1 Byte	2 Bytes	0 or n Bytes

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.
nodeDesc	This field shall only be included in the frame if the status field is SUCCESS.

Example: 55 80 12 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX[2 Bytes] {XX[n Bytes]} AA

4) ZBHCL_CMD_DISCOVERY_SIMPLE_DESC_RSP

srcAddr	seqNum	status	nwkAddrOfInterest	length	simpleDesc
2 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.
length	The length of simple description.
simpleDesc	This field shall only be included in the frame if the status field is SUCCESS.

Example: 55 80 13 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX[2 Bytes] XX {XX[n Bytes]} AA

5) ZBHCL_CMD_DISCOVERY_MATCH_DESC_RSP

srcAddr	seqNum	status	nwkAddrOfInterest	matchLen	matchList
2 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.
matchLen	The count of endpoints on the Remote Device that match the request criteria.
matchList	List of bytes each of which represents an 8-bit endpoint.

Example: 55 80 14 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX[2 Bytes] XX {XX[n Bytes]} AA

6) ZBHCI_CMD_DISCOVERY_ACTIVE_EP_RSP

srcAddr	seqNum	status	nwkAddrOfInterest	activeEpCount	epList
2 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
nwkAddrOfInterest	NWK address for the request.
activeEpCount	The count of active endpoints.
epList	List of active endpoints.

Example: 55 80 15 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX[2 Bytes] XX {XX[n Bytes]} AA

7) ZBHCI_CMD_BIND_RSP / ZBHCI_CMD_UNBIND_RSP

srcAddr	seqNum	status
2 Bytes	1 Byte	1 Byte

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.

Example: ZBHCL_CMD_BIND_RSP:

55 80 20 00 04 00 XX[2 Bytes] XX XX AA

ZBHCL_CMD_UNBIND_RSP:

55 80 21 00 04 00 XX[2 Bytes] XX XX AA

8) ZBHCL_CMD_MGMT_LQI_RSP

srcAddr	seqNum	status	neighborTab Entries	startIdx	neighborTabList Count	neighborTabList
2 Bytes	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	n Bytes

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
neighborTabEntries	Total number of Neighbor Table entries within the Remote Device.
startIdx	Starting index within the Neighbor Table to begin reporting for the neighborTabList.
neighborTabListCount	Number of Neighbor Table entries included within neighborTabList.
neighborTabList	A list of descriptors, beginning with the startIdx element and continuing for neighborTabListCount.

Example: 55 80 30 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX XX XX {XX[n Bytes]} AA

9) ZBHCL_CMD_MGMT_BIND_RSP

srcAddr	seqNum	status	bindingTabEntries	startIdx	bindingTabListCount	bindingTabList
2 Bytes	1 Byte	1 Byte	1 Byte	1 Byte	1 Byte	n Bytes

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
bindingTabEntries	Total number of Binding Table entries within the Remote Device.
startIdx	Starting index within the Binding Table to begin reporting for the bindingTabList.
bindingTabListCount	Number of Binding Table entries included within bindingTabList.
bindingTabList	A list of descriptors, beginning with the startIdx element and continuing for bindingTabListCount.

Example: 55 80 31 msgLenH msgLenL 00 XX[2 Bytes] XX XX XX XX XX {XX[n Bytes]} AA

10) ZBHCI_CMD_MGMT_LEAVE_RSP

srcAddr	seqNum	status	ieeeAddr	rejoin
2 Bytes	1 Byte	1 Byte	8 Bytes	1 Byte

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.
ieeeAddr	64-bit address for the leave device.
rejoin	Whether the leave device to rejoin.

Example: 55 80 32 00 0D 00 XX[2 Bytes] XX XX XX[8 Bytes] XX AA

11) ZBHCI_CMD_MGMT_PERMIT_JOIN_RSP

srcAddr	seqNum	status
2 Bytes	1 Byte	1 Byte

Name	Description
srcAddr	The source address of this command has been received.
seqNum	ZDP transaction sequence number.
status	The status of the request command.

Example: 55 80 34 00 02 00 XX[2 Bytes] XX XX AA

12) ZBHCI_CMD_NODES_JOINED_GET_RSP

totalCnt	startIdx	listCount	status	addrList
2 Bytes	2 Byte	1 Byte	1 Byte	n Bytes

addrList:

IEEEAddr	nwkAddr
8 Bytes	2 Bytes

Name	Description
totalCnt	The total count of the joined nodes.
startIdx	Starting index within the mac address list.
listCount	The count of the MAC address list in the current packet.
status	The status of the request command.
addrList	The address list in the current packet.

Example: 55 80 40 msgLenH msgLenL 00 XX[2 Bytes] XX[2 Bytes] XX XX {XX[n Bytes]} AA

13) ZBHCI_CMD_NODES_TOGGLE_TEST_RSP

No payload.

Example: 55 80 41 00 00 00 AA

14) ZBHCL_CMD_NODES_DEV_ANNCIE_IND

nwkAddr	ieeeAddr	capability
2 Bytes	8 Bytes	1 Byte

Name	Description
nwkAddr	NWK address of the joined device.
ieeeAddr	IEEE address of the joined device.
capability	Capability of the joined device.

Example: 55 80 43 00 0B 00 XX[2 Bytes] XX[8 Bytes] XX AA

15) ZBHCL_CMD_GET_LOCAL_NWK_INFO_RSP

devType	capability	onANetwork	panID	extPanID	nwkAddr	ieeeAddr
1 Byte	1 Byte	1 Byte	2 Bytes	8 Bytes	2 Bytes	8 Bytes

Name	Description
devType	0x00 – coordinator; 0x01 – router; 0x02 – endDevice;
capability	Capability of the local device.
onANetwork	Whether the device on a network.
panID	The 16-bit PAN identifier of the network.
extPanID	The 64-bit PAN identifier of the network.
nwkAddr	The network address of the local device.
ieeeAddr	IEEE address of the local device.

Example: 55 80 45 00 17 00 XX XX XX XX[2 Bytes] XX[8 Bytes] XX[2 Bytes] XX[8 Bytes] AA

16) ZBHCL_CMD_DATA_CONFIRM

dstAddrMode	dstAddr	srcEP	dstEP	clusterID	status
1 Byte	0/2/8 Bytes	1 Byte	0/1 Byte	2 Bytes	1 Byte

Name	Description
dstAddrMode	The destination address mode to which this command will be sent.
devAddr	The destination address to which this command will be sent.
srcEP	The source endpoint for this command.
dstEP	The destination endpoint for this command.
clusterID	The identifier of the cluster for this command.
status	The status of sending this command.
apsCnt	The aps level counter corresponding to this command.

Example: 55 82 00 msgLenH msgLenL 00 XX XX[0/2/8 Bytes] XX XX[0/1 Byte] XX[2 Bytes] XX XX AA

17) ZBHCL_CMD_NODE_LEAVE_IND

totalCnt	extAddr
2 Bytes	8 Bytes

Name	Description
totalCnt	The total count of leave indication which the device has received.
extAddr	The extension address of the leave node.

Example: 55 82 02 00 0a 00 XX[2 Bytes] XX[8 Bytes] AA

8.6 ZCL Cluster commands

8.6.1 ZCL command header format

ZCLCmdHdr:

dstAddrMode	dstAddr	srcEp	dstEp
1 Byte	0/2/8 Bytes	1 Byte	1 Byte

Name	Description
dstAddrMode	Destination address mode: 0 – without destination address and endpoint, for binding; 1 – with group address; 2 – with destination short address and endpoint; 3 – with destination IEEE address and endpoint.
dstAddr	Destination address for this command.
srcEp	Source endpoint.
dstEp	Destination endpoint if dstAddrMode is 2 or 3.

ZCLCmdRspHdr:

srcAddr	srcEp	dstEp	seqNum
2 Bytes	1 Byte	1 Byte	1 Byte

Name	Description
srcAddr	Source address for this command.
srcEp	Source endpoint.
dstEp	Destination endpoint.
seqNum	The sequence number in the zcl header.

8.6.2 General cluster command

8.6.2.1 Message Type (Host)

Message Type	Value
ZBHCL_CMD_ZCL_ATTR_READ	0x0100
ZBHCL_CMD_ZCL_ATTR_WRITE	0x0101
ZBHCL_CMD_ZCL_CONFIG_REPORT	0x0102
ZBHCL_CMD_ZCL_READ_REPORT_CFG	0x0103

8.6.2.2 Payload (Host)

1) ZBHCL_CMD_ZCL_ATTR_READ

ZCLCmdHdr	profileID	direction	clusterID	attrNum	attrList
n Bytes	2 Bytes	1 Byte	2 Bytes	1 Byte	n Bytes

attrList:

attrID[0]	attrID[1]	...	attrID[n]
2 Bytes	2 Bytes	...	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
profileID	Profile identifier.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.
attrNum	The number of attributes to be read.
attrList	The list of the attribute IDs to be read.

Example: 55 01 00 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX XX[2 Bytes] XX {XX[n Bytes]} AA

2) ZBHCL_CMD_ZCL_ATTR_WRITE

ZCLCmdHdr	profileID	direction	clusterID	attrNum	attrList[0]	...	attrList[n]
n Bytes	2 Bytes	1 Byte	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

attrID	dataType	attrData
2 Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
profileID	Profile identifier.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.
attrNum	The number of attributes to be written.
attrList	The list of the attributes to be written.

Example: 55 01 01 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX XX[2 Bytes] XX {XX[n Bytes] ...} AA

3) ZBHCL_CMD_ZCL_CONFIG_REPORT

ZCLCmdHdr	profileID	direction	clusterID	attrNum	attrList[0]	...	attrList[n]
n Bytes	2 Bytes	1 Byte	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

Report Direction	attrID	dataType	minRep Interval	maxRep Interval	reportable Change	timeout Period
1 Byte	2 Bytes	1 Byte	2 Bytes	2 Bytes	n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
profileID	Profile identifier.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.
attrNum	The number of attributes to be configured.
attrList	The list of the attributes to be configured.

Example: 55 01 02 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX XX[2 Bytes] XX {XX[n Bytes] ...} AA

4) ZBHCL_CMD_ZCL_READ_REPORT_CFG

ZCLCmdHdr	profileID	direction	clusterID	attrNum	attrList[0]	...	attrList[n]
n Bytes	2 Bytes	1 Byte	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

reportDirection	attrID
1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
profileID	Profile identifier.
direction	0 – Client to server; 1 – Server to client.
clusterID	Cluster identifier.
attrNum	The number of attributes' configuration to be read.
attrList	The list of the attributes to be read.

Example: 55 01 03 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX XX[2 Bytes] XX {XX[n Bytes] ...} AA

8.6.2.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_ZCL_ATTR_READ_RSP	0x8100
ZBHCI_CMD_ZCL_ATTR_WRITE_RSP	0x8101
ZBHCI_CMD_ZCL_CONFIG_REPORT_RSP	0x8102
ZBHCI_CMD_ZCL_READ_REPORT_CFG_RSP	0x8103
ZBHCI_CMD_ZCL_REPORT_MSG_RCV	0x8104
ZBHCI_CMD_ZCL_DEFAULT_RSP	0x8105

8.6.2.4 Payload (Slave)

1) ZBHCI_CMD_ZCL_ATTR_READ_RSP

ZCLCmdRspHdr	clusterID	attrNum	attrList[0]	...	attrList[n]
5 Bytes	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

attrID	status	dataType	data
2 Bytes	1 Byte	1 Byte	n Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
clusterID	Cluster identifier.
attrNum	The number of attributes to be read.
attrList	The list of the attributes to be read.

Example: 55 81 00 msgLenH msgLenL 00 XX[5 Bytes] XX[2 Bytes] XX {XX[n Bytes] ...} AA

2) ZBHCL_CMD_ZCL_ATTR_WRITE_RSP

ZCLCmdRspHdr	clusterID	attrNum	attrList[0]	...	attrList[n]
5 Bytes	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

status	attrID
1 Byte	2 Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
clusterID	Cluster identifier.
attrNum	The number of attributes to be written.
attrList	The list of the attributes to be written.

Example: 55 81 01 msgLenH msgLenL 00 XX[5 Bytes] XX[2 Bytes] XX {XX[n Bytes] ...} AA

3) ZBHCL_CMD_ZCL_CONFIG_REPORT_RSP

ZCLCmdRspHdr	clusterID	attrNum	attrList[0]	...	attrList[n]
5 Bytes	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

status	reportDirection	attrID
1 Byte	1 Byte	2 Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
clusterID	Cluster identifier.
attrNum	The number of attributes' reporting to be configured.
attrList	The list of the attributes to be configured.

Example: 55 81 02 msgLenH msgLenL 00 XX[5 Bytes] XX[2 Bytes] XX {XX[n Bytes] ...} AA

4) ZBHCL_CMD_ZCL_READ_REPORT_CFG_RSP

ZCLCmdRspHdr	clusterID	attrNum	attrList[0]	...	attrList[n]
5 Bytes	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

status	Report Direction	attrID	dataType	minRep Interval	maxRep Interval	Reportable Change	timeout Period
1 Byte	1 Byte	2 Bytes	1 Byte	2 Bytes	2 Bytes	n Bytes	2 Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
clusterID	Cluster identifier.

Name	Description
attrNum	The number of attributes' reporting to be read.
attrList	The list of the attributes to be read.

Example: 55 81 03 msgLenH msgLenL 00 XX[5 Bytes] XX[2 Bytes] XX {XX[n Bytes] ...} AA

5) ZBHCL_CMD_ZCL_REPORT_MSG_RCV

ZCLCmdRspHdr	clusterID	attrNum	attrList[0]	...	attrList[n]
5 Bytes	2 Bytes	1 Byte	n Bytes	...	n Bytes

attrList:

attrID	dataType	data
2 Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
clusterID	Cluster identifier.
attrNum	The number of attributes' reporting message to be received.
attrList	The list of the attributes' reporting message to be received.

Example: 55 81 04 msgLenH msgLenL 00 XX[5 Bytes] XX[2 Bytes] XX {XX[n Bytes] ...} AA

6) ZBHCL_CMD_ZCL_DEFAULT_RSP

ZCLCmdRspHdr	clusterID	commandID	status
5 Bytes	2 Bytes	1 Byte	1 Byte

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
clusterID	Cluster identifier.

Name	Description
commandID	The commandID of the send message.
status	The status of the message.

Example: 55 81 05 00 09 00 XX[5 Bytes] XX[2 Bytes] XX XX AA

8.6.3 Basic cluster command

8.6.3.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_BASIC_RESET	0x0110

8.6.3.2 Payload (Host)

ZCLCmdHdr
n Bytes

Example: 55 01 10 msgLenH msgLenL 00 XX[n Bytes] AA

8.6.4 Group cluster command

8.6.4.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_GROUP_ADD	0x0120
ZBHCI_CMD_ZCL_GROUP_VIEW	0x0121
ZBHCI_CMD_ZCL_GROUP_GET_MEMBERSHIP	0x0122
ZBHCI_CMD_ZCL_GROUP_REMOVE	0x0123
ZBHCI_CMD_ZCL_GROUP_REMOVE_ALL	0x0124
ZBHCI_CMD_ZCL_GROUP_ADD_IF_IDENTIFYING	0x0125

8.6.4.2 Payload (Host)

1) ZBHCL_CMD_ZCL_GROUP_ADD

ZCLCmdHdr	groupId	groupName
n Bytes	2 Bytes	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	Group identifier.
groupName	Group name, character string.

Example: 55 01 20 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX[n Bytes] AA

2) ZBHCL_CMD_ZCL_GROUP_VIEW

ZCLCmdHdr	groupId
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	Group identifier.

Example: 55 01 21 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

3) ZBHCL_CMD_ZCL_GROUP_GET_MEMBERSHIP

ZCLCmdHdr	groupCount	groupList
n Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupCount	Group count.

Name	Description
------	-------------

groupList	Group list.
-----------	-------------

Example: 55 01 22 msgLenH msgLenL 00 XX[n Bytes] XX XX[n Bytes] AA

4) ZBHCL_CMD_ZCL_GROUP_REMOVE

ZCLCmdHdr	groupId
-----------	---------

n Bytes	2 Bytes
---------	---------

Name	Description
------	-------------

ZCLCmdHdr	ZCL command header.
-----------	---------------------

groupId	Group identifier.
---------	-------------------

Example: 55 01 23 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

5) ZBHCL_CMD_ZCL_GROUP_REMOVE_ALL

ZCLCmdHdr

n Bytes

Example: 55 01 24 msgLenH msgLenL 00 XX[n Bytes] AA

6) ZBHCL_CMD_ZCL_GROUP_ADD_IF_IDENTIFYING

ZCLCmdHdr	groupId	groupName
-----------	---------	-----------

n Bytes	2 Bytes	n Bytes
---------	---------	---------

Name	Description
------	-------------

ZCLCmdHdr	ZCL command header.
-----------	---------------------

groupId	Group identifier.
---------	-------------------

groupName	Group name, character string.
-----------	-------------------------------

Example: 55 01 25 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX[n Bytes] AA

8.6.4.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_ZCL_GROUP_ADD_RSP	0x8120
ZBHCI_CMD_ZCL_GROUP_VIEW_RSP	0x8121
ZBHCI_CMD_ZCL_GROUP_GET_MEMBERSHIP_RSP	0x8122
ZBHCI_CMD_ZCL_GROUP_REMOVE_RSP	0x8123

8.6.4.4 Payload (Slave)

1) ZBHCI_CMD_ZCL_GROUP_ADD_RSP

ZCLCmdRspHdr	status	groupId
5 Bytes	1 Byte	2 Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
status	The status field is set to SUCCESS, DUPLICATE_EXISTS, or INSUFFICIENT_SPACE as appropriate.
groupId	Group identifier.

Example: 55 81 20 00 08 00 XX[5 Bytes] XX XX[2 Bytes] AA

2) ZBHCI_CMD_ZCL_GROUP_VIEW_RSP

ZCLCmdRspHdr	status	groupId	groupName
5 Bytes	1 Byte	2 Bytes	n Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.

Name	Description
status	The status field is set to SUCCESS, or NOT_FOUND as appropriate.
groupId	Group identifier.
groupName	Group name, character string.

Example: 55 81 21 msgLenH msgLenL 00 XX[5 Bytes] XX XX[2 Bytes] XX[n Bytes] AA

3) ZBHCL_CMD_ZCL_GROUP_GET_MEMBERSHIP_RSP

ZCLCmdRspHdr	capability	groupCount	groupList
5 Bytes	1 Byte	1 Byte	n Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
capability	The remaining capability of the group table of the device.
groupCount	The number of groups contained in the group list field.
groupName	The list of groupId in the group list field.

Example: 55 81 22 msgLenH msgLenL 00 XX[5 Bytes] XX XX XX[n Bytes] AA

4) ZBHCL_CMD_ZCL_GROUP_REMOVE_RSP

ZCLCmdRspHdr	status	groupId
5 Bytes	1 Byte	2 Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
status	The status field is set to SUCCESS, DUPLICATE_EXISTS, or INSUFFICIENT_SPACE as appropriate.
groupId	Group identifier.

Example: 55 81 23 00 08 00 XX[5 Bytes] XX XX[2 Bytes] AA

8.6.5 Identify cluster command

8.6.5.1 Message Type (Host)

Message Type	Value
ZBHCL_CMD_ZCL_IDENTIFY	0x0130
ZBHCL_CMD_ZCL_IDENTIFY_QUERY	0x0131

8.6.5.2 Payload (Host)

1) ZBHCL_CMD_ZCL_IDENTIFY

ZCLCmdHdr	identifyTime
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
identifyTime	Unsigned 16-bit integer.

Example: 55 01 30 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

2) ZBHCL_CMD_ZCL_IDENTIFY_QUERY

ZCLCmdHdr
n Bytes

Example: 55 01 31 msgLenH msgLenL 00 XX[n Bytes] AA

8.6.5.3 Message Type (Slave)

Message Type	Value
ZBHCL_CMD_ZCL_IDENTIFY_QUERY_RSP	0x8131

8.6.5.4 Payload (Slave)

1) ZBHCI_CMD_ZCL_IDENTIFY_QUERY_RSP

ZCLCmdRspHdr	timeout
5 Bytes	2 Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
timeout	The remaining time.

Example: 55 81 31 00 07 00 XX[5 Bytes] XX[2 Bytes] AA

8.6.6 On/Off cluster command

8.6.6.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_ONOFF_ON	0x0140
ZBHCI_CMD_ZCL_ONOFF_OFF	0x0141
ZBHCI_CMD_ZCL_ONOFF_TOGGLE	0x0142

8.6.6.2 Payload (Host)

ZCLCmdHdr
n Bytes

Example: ZBHCI_CMD_ZCL_ONOFF_ON:

55 01 40 msgLenH msgLenL 00 XX[n Bytes] AA

ZBHCI_CMD_ZCL_ONOFF_OFF:

55 01 41 msgLenH msgLenL 00 XX[n Bytes] AA

ZBHCI_CMD_ZCL_ONOFF_TOGGLE:

55 01 42 msgLenH msgLenL 00 XX[n Bytes] AA

8.6.7 Level cluster command

8.6.7.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_LEVEL_MOVE2LEVEL	0x0150
ZBHCI_CMD_ZCL_LEVEL_MOVE	0x0151
ZBHCI_CMD_ZCL_LEVEL_STEP	0x0152
ZBHCI_CMD_ZCL_LEVEL_STOP	0x0153
ZBHCI_CMD_ZCL_LEVEL_MOVE2LEVEL_WITHONOFF	0x0154
ZBHCI_CMD_ZCL_LEVEL_MOVE_WITHONOFF	0x0155
ZBHCI_CMD_ZCL_LEVEL_STEP_WITHONOFF	0x0156
ZBHCI_CMD_ZCL_LEVEL_STOP_WITHONOFF	0x0157

8.6.7.2 Payload (Host)

1) ZBHCI_CMD_ZCL_LEVEL_MOVE2LEVEL

ZCLCmdHdr	level	transTime
n Bytes	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
level	Level.
transTime	Transition time, 1/10ths of a second.

Example: 55 01 50 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] AA

2) ZBHCI_CMD_ZCL_LEVEL_MOVE

ZCLCmdHdr	mode	rate
n Bytes	1 Byte	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Move mode. 0x00 – Up; 0x01 – Down.
rate	The rate field specifies the rate of movement in units per second.

Example: 55 01 51 msgLenH msgLenL 00 XX[n Bytes] XX XX AA

3) ZBHCL_CMD_ZCL_LEVEL_STEP

ZCLCmdHdr	mode	stepSize	transTime
n Bytes	1 Byte	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Step mode. 0x00 – Up; 0x01 – Down.
stepSize	A step is a change in the current level of “step size” units.
transTime	The transition time field specifies the time that shall be taken to perform the step, in 1/10ths of a second.

Example: 55 01 52 msgLenH msgLenL 00 XX[n Bytes] XX XX XX[2 Bytes] AA

4) ZBHCL_CMD_ZCL_LEVEL_STOP

ZCLCmdHdr
n Bytes

Example: 55 01 53 msgLenH msgLenL 00 XX[n Bytes] AA

5) ZBHCL_CMD_ZCL_LEVEL_MOVE2LEVEL_WITHONOFF

ZCLCmdHdr	level	transTime
n Bytes	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
level	Level.
transTime	Transition time, 1/10ths of a second.

Example: 55 01 54 msgLenH msgLenL 00 XX[n Bytes] XX XX[2 Bytes] AA

6) ZBHCL_CMD_ZCL_LEVEL_MOVE_WITHONOFF

ZCLCmdHdr	mode	rate
n Bytes	1 Byte	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Move mode. 0x00 – Up; 0x01 – Down.
rate	The rate field specifies the rate of movement in units per second.

Example: 55 01 55 msgLenH msgLenL 00 XX[n Bytes] XX XX AA

7) ZBHCL_CMD_ZCL_LEVEL_STEP_WITHONOFF

ZCLCmdHdr	mode	stepSize	transTime
n Bytes	1 Byte	1 Byte	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
mode	Step mode. 0x00 – Up; 0x01 – Down.
stepSize	A step is a change in the current level of “step size” units.
transTime	The transition time field specifies the time that shall be taken to perform the step, in 1/10ths of a second.

Example: 55 01 56 msgLenH msgLenL 00 XX[n Bytes] XX XX XX[2 Bytes] AA

8) ZBHCI_CMD_ZCL_LEVEL_STOP_WITHONOFF

ZCLCmdHdr

n Bytes

Example: 55 01 57 msgLenH msgLenL 00 XX[n Bytes] AA

8.6.8 Scene cluster command

8.6.8.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_ZCL_SCENE_ADD	0x0160
ZBHCI_CMD_ZCL_SCENE_VIEW	0x0161
ZBHCI_CMD_ZCL_SCENE_REMOVE	0x0162
ZBHCI_CMD_ZCL_SCENE_REMOVE_ALL	0x0163
ZBHCI_CMD_ZCL_SCENE_STORE	0x0164
ZBHCI_CMD_ZCL_SCENE_RECALL	0x0165
ZBHCI_CMD_ZCL_SCENE_GET_MEMBERSHIP	0x0166

8.6.8.2 Payload (Host)

1) ZBHCI_CMD_ZCL_SCENE_ADD

ZCLCmdHdr	groupId	sceneId	transTime	scene NameLen	scene Name	extField Len	extField Sets
n Bytes	2 Bytes	1 Byte	2 Byte	1 Byte	n Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.
transTime	The amount of time, in seconds, it will take for the device to change from its current state to the requested scene.

Name	Description
sceneNameLen	Length of scene name.
sceneName	Scene name, char string.
extFieldLen	Length of extFieldSets field.
extFieldSets	The sum of all such defines a scene.

Example: 55 01 60 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX XX[2 Bytes] XX {XX[n Bytes]} XX {XX[n Bytes]} AA

2) ZBHCL_CMD_ZCL_SCENE_VIEW

ZCLCmdHdr	groupId	sceneId
n Bytes	2 Bytes	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

Example: 55 01 61 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

3) ZBHCL_CMD_ZCL_SCENE_REMOVE

ZCLCmdHdr	groupId	sceneId
n Bytes	2 Bytes	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

Example: 55 01 62 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

4) ZBHCL_CMD_ZCL_SCENE_REMOVE_ALL

ZCLCmdHdr	groupId
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.

Example: 55 01 63 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

5) ZBHCL_CMD_ZCL_SCENE_STORE

ZCLCmdHdr	groupId	sceneId
n Bytes	2 Bytes	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

Example: 55 01 64 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

6) ZBHCL_CMD_ZCL_SCENE_RECALL

ZCLCmdHdr	groupId	sceneId
n Bytes	2 Bytes	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

Example: 55 01 65 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] XX AA

7) ZBHCI_CMD_ZCL_SCENE_GET_MEMBERSHIP

ZCLCmdHdr	groupId
n Bytes	2 Bytes

Name	Description
ZCLCmdHdr	ZCL command header.
groupId	The group ID for which this scene applies.

Example: 55 01 66 msgLenH msgLenL 00 XX[n Bytes] XX[2 Bytes] AA

8.6.8.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_ZCL_SCENE_ADD_RSP	0x8160
ZBHCI_CMD_ZCL_SCENE_VIEW_RSP	0x8161
ZBHCI_CMD_ZCL_SCENE_REMOVE_RSP	0x8162
ZBHCI_CMD_ZCL_SCENE_REMOVE_ALL_RSP	0x8163
ZBHCI_CMD_ZCL_SCENE_STORE_RSP	0x8164
ZBHCI_CMD_ZCL_SCENE_GET_MEMBERSHIP_RSP	0x8166

8.6.8.4 Payload (Slave)

1) ZBHCI_CMD_ZCL_SCENE_ADD_RSP

ZCLCmdRspHdr	status	groupId	sceneId
5 Bytes	1 Byte	2 Bytes	1 Byte

Name	Description
ZCLCmdRspHdr	The zcl header of response command.

Name	Description
status	SUCCESS, INSUFFICIENT_SPACE or INVALID_FIELD (the group is not present in the group table).
groupid	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

Example: 55 81 60 00 09 00 XX[5 Bytes] XX XX[2 Bytes] XX AA

2) ZBHCL_CMD_ZCL_SCENE_VIEW_RSP

ZCLCmdRspHdr	status	groupid	sceneId	transTime	sceneName	extFieldSets
5 Bytes	1 Byte	2 Bytes	1 Byte	2 Bytes	n Bytes	n Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
status	SUCCESS, NOT_FOUND (the scene is not present in the scene table) or INVALID_FIELD (the group is not present in the group table).
groupid	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.
transTime	Transition time copied from scene table entry.
sceneName	Scene name copied from scene table entry. First byte is the length of the scene name.
extFieldSets	Extension field sets copied from scene table entry. First byte is the length of the extension field sets.

Example: 55 81 61 msgLenH msgLenL 00 XX[5 Bytes] XX XX[2 Bytes] XX XX[2 Bytes] XX[n Bytes]

3) ZBHCL_CMD_ZCL_SCENE_REMOVE_RSP

ZCLCmdRspHdr	status	groupid	sceneId
5 Bytes	1 Byte	2 Bytes	1 Byte

Name	Description
ZCLCmdRspHdr	The zcl header of response command.

Name	Description
status	SUCCESS, NOT_FOUND (the scene is not present in the scene table) or INVALID_FIELD (the group is not present in the group table).
groupid	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

Example: 55 81 62 00 09 00 XX[5 Bytes] XX XX[2 Bytes] XX AA

4) ZBHCL_CMD_ZCL_SCENE_REMOVE_ALL_RSP

ZCLCmdRspHdr	status	groupid
5 Bytes	1 Byte	2 Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
status	SUCCESS or INVALID_FIELD (the group is not present in the group table).
groupid	The group ID for which this scene applies.

Example: 55 81 63 00 08 00 XX[5 Bytes] XX XX[2 Bytes] AA

5) ZBHCL_CMD_ZCL_SCENE_STORE_RSP

ZCLCmdRspHdr	status	groupid	sceneId
5 Bytes	1 Byte	2 Bytes	1 Byte

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
status	SUCCESS, INSUFFICIENT_SPACE or INVALID_FIELD (the group is not present in the group table).
groupid	The group ID for which this scene applies.
sceneId	The identifier, unique within this group, which is used to identify this scene.

Example: 55 81 64 00 09 00 XX[5 Bytes] XX XX[2 Bytes] XX AA

6) ZBHCL_CMD_ZCL_SCENE_GET_MEMBERSHIP_RSP

ZCLCmdRspHdr	status	capability	groupId	sceneCnt	sceneList
5 Bytes	1 Byte	1 Byte	2 Bytes	1 Byte	n Bytes

Name	Description
ZCLCmdRspHdr	The zcl header of response command.
status	SUCCESS or INVALID_FIELD (the group is not present in the group table).
capability	Contain the remaining capacity of the scene table of the device.
groupId	The group ID for which this scene applies.
sceneCnt	The number of scenes contained in the scene list field.
sceneList	Contain the identifiers of all the scenes in the scene table with the corresponding Group ID.

Example: 55 81 66 msgLenH msgLenL 00 XX[5 Bytes] XX XX XX[2 Bytes] XX XX[n Bytes] AA

8.6.9 OTA cluster command

8.6.9.1 Message Type (Host)

Message Type	Value
ZBHCL_CMD_ZCL_OTA_IMAGE_NOTIFY	0x0190

8.6.9.2 Payload (Host)

1) ZBHCL_CMD_ZCL_OTA_IMAGE_NOTIFY

ZCLCmdHdr	payloadType	queryJitter
n Bytes	1 Byte	1 Byte

Name	Description
ZCLCmdHdr	ZCL command header.

Name	Description
payloadType	0x00 – Query jitter; 0x01 – Query jitter and manufacturer code; 0x02 – Query jitter, manufacturer code, and image type; 0x03 – Query jitter, manufacturer code, image type, and new file version.
queryJitter	By using the parameter, it prevents a single notification of a new OTA upgrade image from flooding the upgrade server with requests from clients.

Example: 55 01 90 msgLenH msgLenL 00 XX[n Bytes] XX XX AA

8.7 HCI serial port upgrade command

8.7.1 Message Type (Host)

Message Type	Value
ZBHCI_CMD_OTA_START_REQUEST	0x0210
ZBHCI_CMD_OTA_BLOCK_RESPONSE	0x0211

8.7.2 Payload (Host)

1) ZBHCI_CMD_OTA_START_REQUEST

OTAFileSize

4 Bytes

Name	Description
OTAFileSize	The total size of the OTA file.

Example: 55 02 10 00 04 00 XX[4 Bytes] AA

2) ZBHCI_CMD_OTA_BLOCK_RESPONSE

status	offset	blockLen	blockContent
1 Byte	4 Bytes	1 Byte	n Bytes

Name	Description
status	The status of the request which the slave send.
offset	The offset of the block.
blockLen	The length of the block.
blockContent	The content of the block.

Example: 55 02 11 msgLenH msgLenL 00 XX XX[4 Bytes] XX XX[n Bytes] AA

8.7.3 Message Type (Slave)

Message Type	Value
ZBHCI_CMD_OTA_START_RESPONSE	0x8210
ZBHCI_CMD_OTA_BLOCK_REQUEST	0x8211
ZBHCI_CMD_OTA_END_STATUS	0x8212

8.7.4 Payload (Slave)

1) ZBHCI_CMD_OTA_START_RESPONSE

flashAddrStart	totalSize	offset	status
4 Bytes	4 Bytes	4 Bytes	1 Byte

Name	Description
flashAddrStart	The start address of the flash which to save the OTA file.
totalSize	The total size of the upgrade file.
offset	The offset of the block.
status	The status of the request.

Example: 55 82 10 00 0d 00 XX[4 Bytes] XX[4 Bytes] XX[4 Bytes] XX AA

2) ZBHCI_CMD_OTA_BLOCK_REQUEST

offset	blockLen
4 Bytes	1 Bytes

Name	Description
offset	The offset of the block.
blockLen	The length of the block.

Example: 55 82 11 00 05 00 XX[4 Bytes] XX AA

3) ZBHCI_CMD_OTA_END_STATUS

totalSize	offset	status
4 Bytes	4 Bytes	1 Byte

Name	Description
totalSize	The total size of the upgrade file.
offset	The offset of the block.
status	The status of the request.

Example: 55 82 12 00 00 09 00 XX[4 Bytes] XX[4 Bytes] XX AA

9 Appendix: Zigbee Alliance Pro R21 Certification



Figure 9.1: Zigbee Alliance