BroadBee Application Software Reference Manual

Broadcom Corporation
5300 California Avenue
Irvine, California, USA 92677
Phone: 949-926-5000
Fax: 949-926-5203
www.broadcom.com

Revision History

Revision	Date	Change Description
1.0	Feb. 02, 2015	Initial release
1.1	Apr. 14, 2015	Added API's for the IEEE addresses
1.2	Mar. 4, 2016	Added use cases for ZHA, ZRC2.0, ZRC1.1
1.3	Mar. 18, 2016	Added CIE device, updated API

Table of Contents

List of Tables

List of Figures

1. References

[1] 053474r20: ZigBee® PRO Specification r20
[2] 053520r29: ZigBee® Home Automation Public Application Profile
[3] IEEE 802.15.4-2006: MAC/PHY Specification for WPANs
[4] 075123r04ZB: ZigBee® Cluster Library Specification
[5] 094945r00ZB: ZigBee® RF4CE Specification v1.01
[6] Comcast, Comcast-SP-RF4CE-MSO-Profile-Pre-I01-120130, RF4CE MSO Profile Specification
[7] 11187r23ZB: ZigBee® RF4CE Generic Device Profile Specification v2.0 (draft)
[8] 12-0368-20: ZigBee® RF4CE ZRC Profile Specification v2.0.0 (draft)
[9] OC-SP-RF4CE-I01-120924: Cable Profile for the ZigBee® RF4CE Remote Control Specification

2. Introduction

BroadBee is Broadcom's ZigBee software stack to implement both ZigBee-RF4CE and ZigBee-PRO stacks defined by ZigBee Alliance, WPAN MAC and PHY layers defined by IEEE-802.15.4, and MSO's Profiles for the applications of Remote Control, Input Devices, Home Automation, and others. The building blocks of ZigBee-PRO and ZigBee-RF4CE stacks are shown on Figure 1.

BroadBee will implement the Dual Stacks that shares the same MAC/PHY/Radio hardware and software as shown on Figure 2.

As shown on Figure 2 and Figure 3, the ZigBee block within a SoC will contain all necessary hardware as well as BroadBee software to perform the normal ZigBee operations within the AON (Always ON) island. Only application specific software will reside in Host side. After ZigBee hardware has been initialized and BroadBee software has been downloaded into ZigBee block, all communications between Host application software and BroadBee software will be done through the Mailbox hardware.

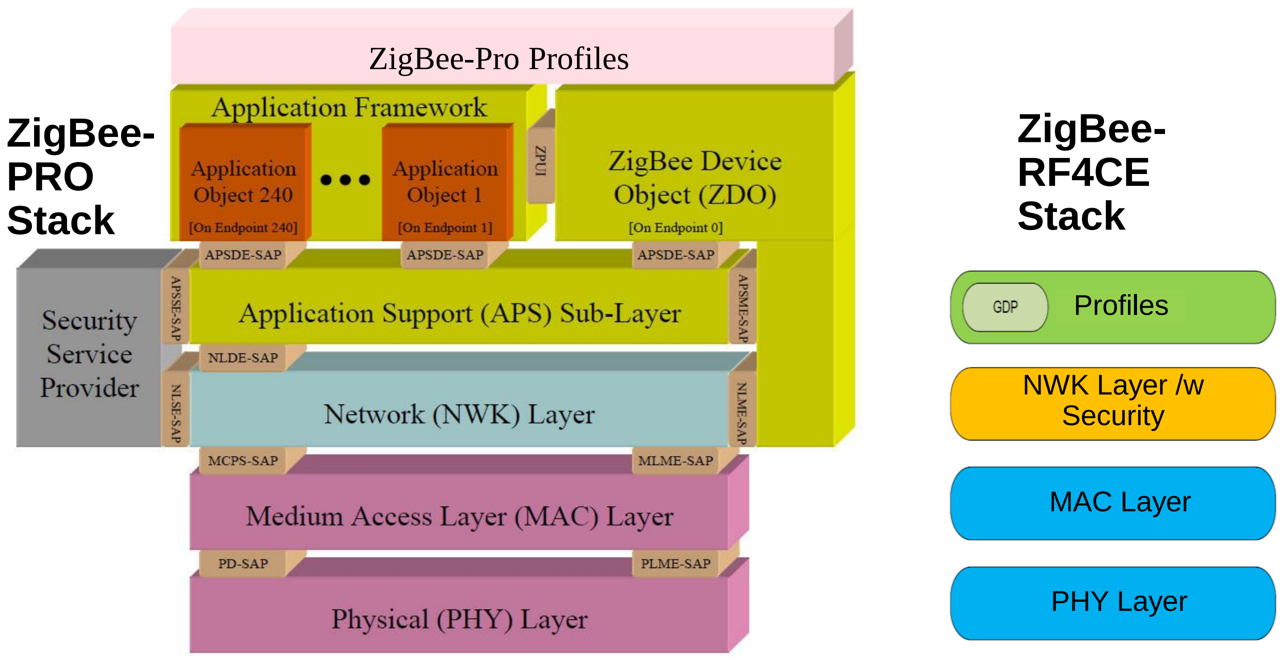


Figure 1: ZigBee-PRO and ZigBee-RF4CE Stacks

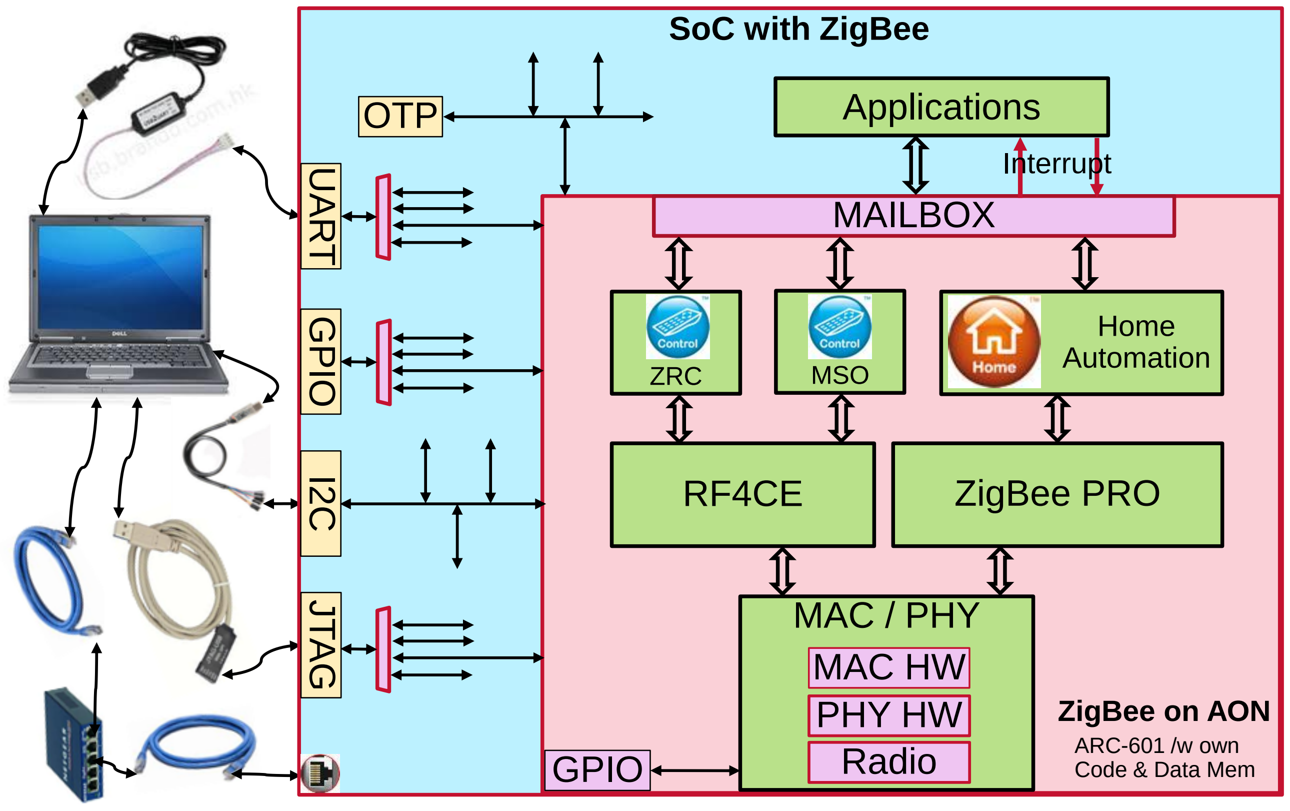


Figure 2: ZigBee Block Diagram in a SoC

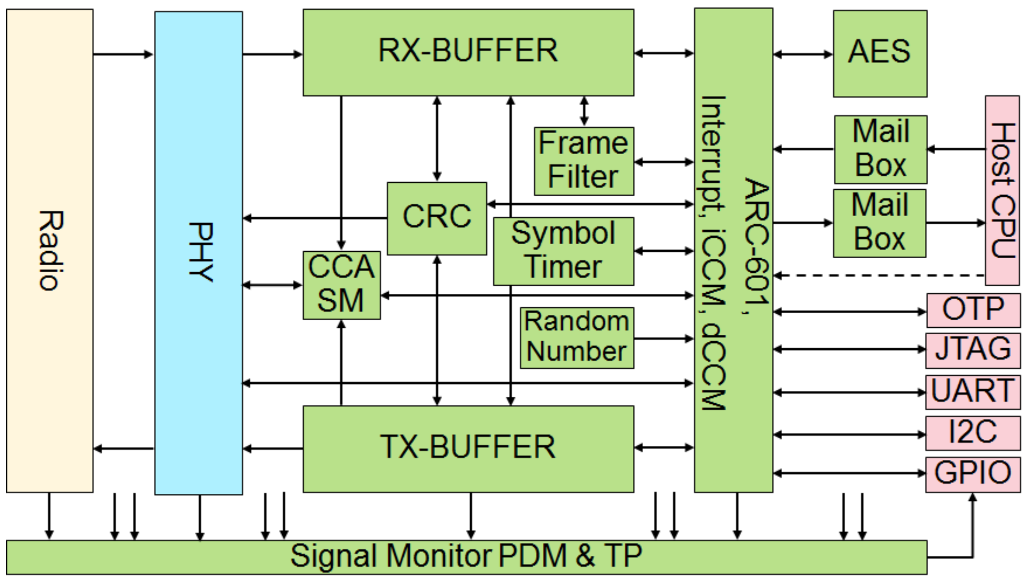


Figure 3: ZigBee Hardware Block Diagram

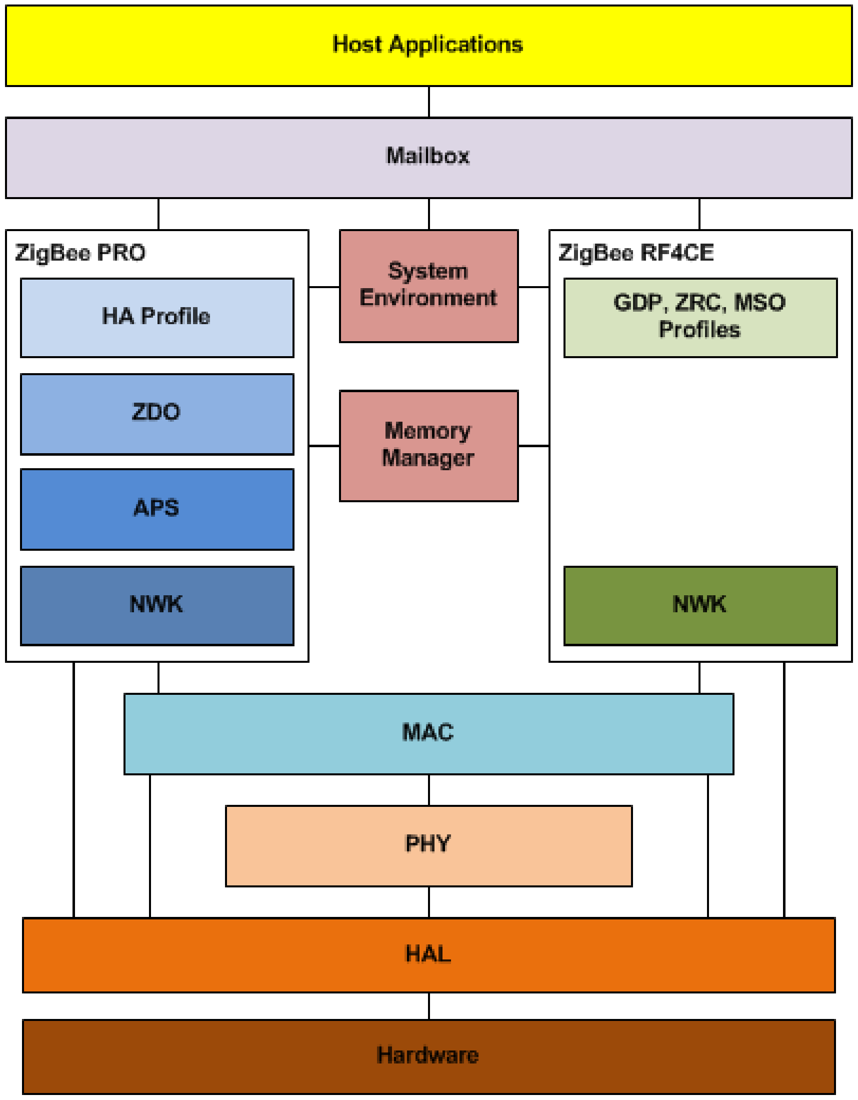


Figure 4: BroadBee Software Block Diagram

Figure 4 shows the software block diagram of BroadBee. As clearly shown on this, BroadBee support two separate software stacks; ZigBee-PRO and ZigBee-RF4CE at the same time. The tasks on each stack should run concurrently with a proper task scheduling, but the activities on one stack should not affect anything on the other stack. Also, depending on the customer's configuration, either of two stacks could be running along, and there shouldn't be any dependency between two stacks.

The same MAC and PHY are used by both stacks, and should be implemented context independently for the same code to be used with different data.

User application software will reside in Host system, and should communicate through the dedicated mailboxes. The mailbox subsystem hardware and software cover the low level communication. APIs use the concept of Remote Procedure Call (RPC), and will work as if there is no Mailbox in between ZigBee and Host sides.

The purpose of this document is to show the Mailbox hardware/software in brief, ZigBee firmware downloading mechanism, and specify the user level APIs to communicate with BroadBee software through the Mailbox hardware.

3. Mailbox

Since ZigBee block will be on AON and will not have a direct access to DDR on Host side, DDR memory cannot be used for the communication between ZigBee and Host. To resolve this, ZigBee block has included two dedicated mailboxes with 128-byte deep FIFO each; one for message from Host to ZigBee, and the other from ZigBee to Host.

After the ZigBee system has been initialized and started, all communication between ZigBee and Host systems shall be through the mailboxes with interrupts. The mailbox access protocol is as shown on the Figure 5. The sender can put the message into the mailbox when the mailbox is empty, and sets the semaphore flag to indicate the message ready to the receiver. The receiver shall get an interrupt from the semaphore flag, pull the message out from the mailbox, and clear the semaphore flag.

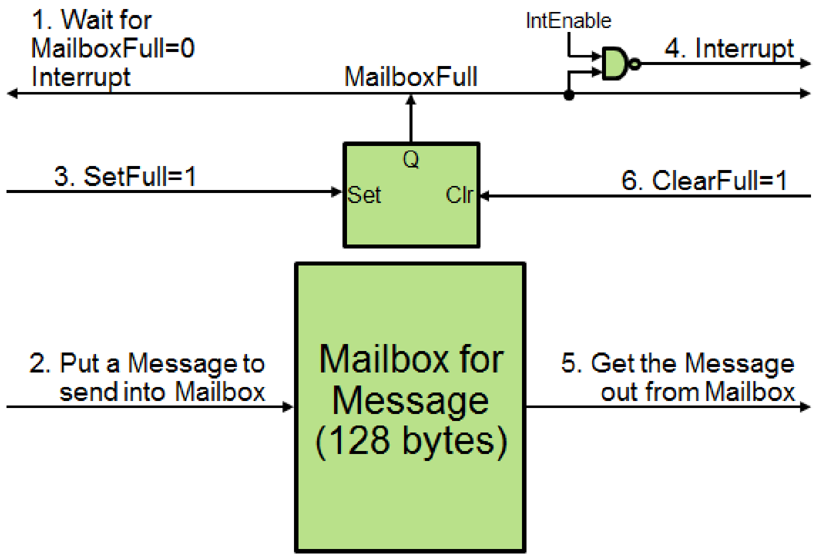

1. Wait for MailboxFull=0 Interrupt
IntEnable
4. Interrupt
MailboxFull
Q
3. SetFull=1
Set
Clr
6. ClearFull=1
2. Put a Message to send into Mailbox
Mailbox for Message (128 bytes)
5. Get the Message out from Mailbox

Figure 5: Mailbox Access Protocol

The API over the mailboxes will be asynchronous way, and will use the service primitive concept with Request, Confirm, Indication, and Response, as shown on Figure 6. Request primitive from Host to ZigBee will be processed and responded by Confirm primitive. Indication primitive from ZigBee to Host will be processed and responded by Response primitive. The Confirm and Response primitives shall be used for most of Request and Indication primitives, but some cases it could be omitted depending on the message.

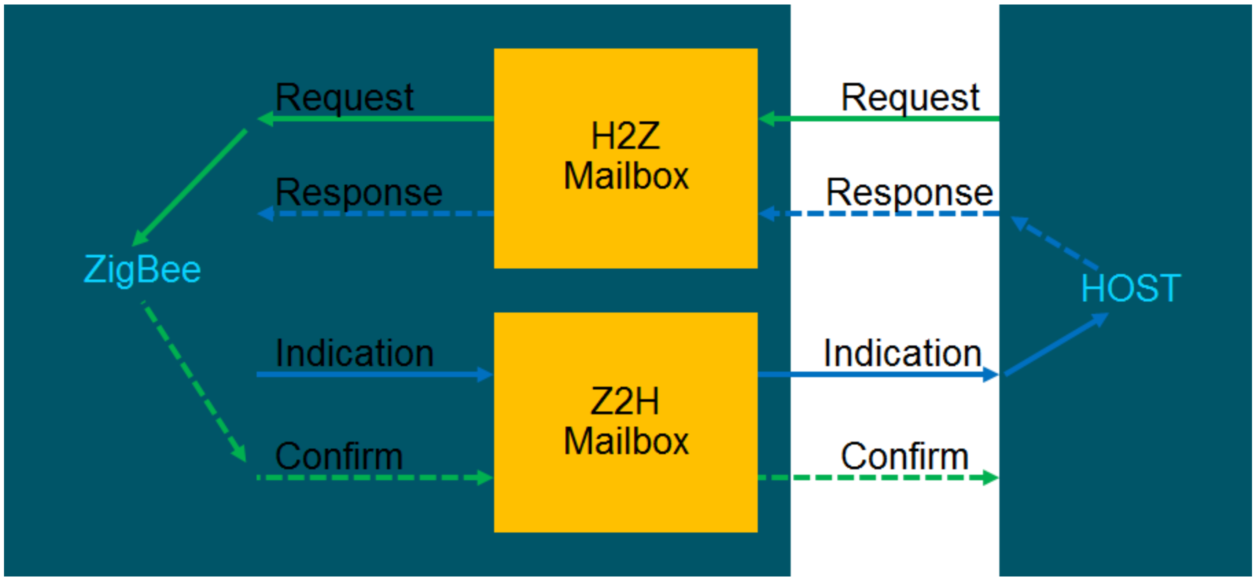


Figure 6: Service Primitives through Mailbox

In our system application, there could be multiple CPUs on the Host side, and each could access the Mailboxes to communicate with ZigBee independently. At this moment there could be up to three sub-modules in Host side to communicate with ZigBee; Set Top Box, Cable Modem, and Route Gateway. For ZigBee to communicate with three sub-modules, it could be easiest to have total six mailboxes, but it is too expensive on hardware. To use only two mailboxes between ZigBee and three different sub-modules, we need some arbitrator for these mailboxes.

In case of message from ZigBee to Host, ZigBee software itself will do the role of arbitrator. Figure 7 shows how this works. There are three separate interrupt signals; one for each Host sub-module. When the Z2H Mailbox is empty, none of three interrupt bits to Host sub-modules shall be set, and the mailbox empty interrupt request bit should be set for the ZigBee CPU. When Z2H Mailbox is empty, ZigBee CPU will load a message to a sub-module, and will set an interrupt to the corresponding sub-module on Host side. A sub-module on the Host side shall get an interrupt, take the message out from the Z2H Mailbox, and clear the interrupt bit. This should trigger a mailbox empty interrupt to ZigBee CPU for the next message.

Figure 8 shows how a sub-module in Host side could send a message to ZigBee. The “MBOX_SEM” bit will be set by ZigBee CPU only, and will be cleared automatically when it is read by any sub-module in Host side. MBOX_SEM bit will generate an interrupt request to all three sub-modules. Any sub-module that wants to use the H2Z Mailbox to send a message to ZigBee should read the MBOX_SEM bit first. Whoever reads it first will get one from this register and gets the grant to use H2Z Mailbox. If others than the first one read this bit, they will get zero, and should not use the mailbox. A sub-module that received the grant (MBOX_SEM=1) shall put the message into H2Z Mailbox and set the MBOX_H2Z_FULL_INTR interrupt request bit. ZigBee CPU will get an interrupt, read the message from the mailbox, clear H2Z_MB_Interrupt, and set the MBOX_SEM bit for the next message. If a sub-module has gotten the grant, but doesn't set the MBOX_H2Z_FULL_ INTR interrupt request bit within the time length defined by COMPARE register, then it is considered as a hang situation by some reason on Host side and the hardware will generate MBOX_SEM_TIMER_INTR interrupt request to ZigBee. ZigBee will clear out this situation, and will set the MBOX_SEM bit again to free up the H2Z Mailbox for the next message.

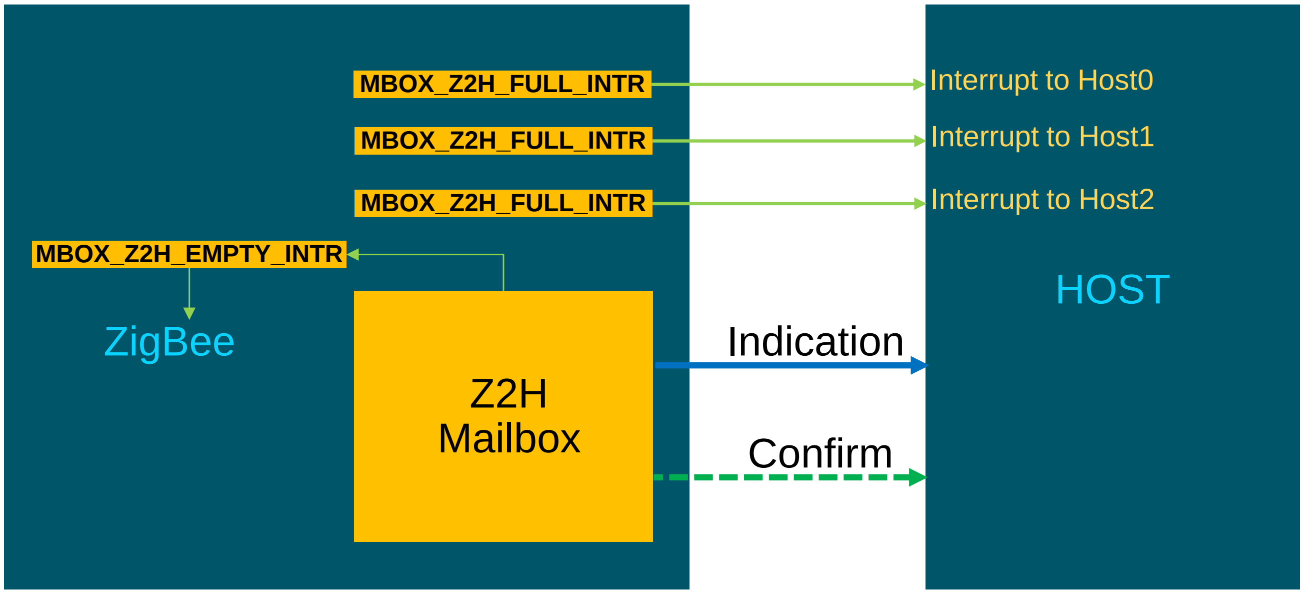


Figure 7: Message from ZigBee to Host

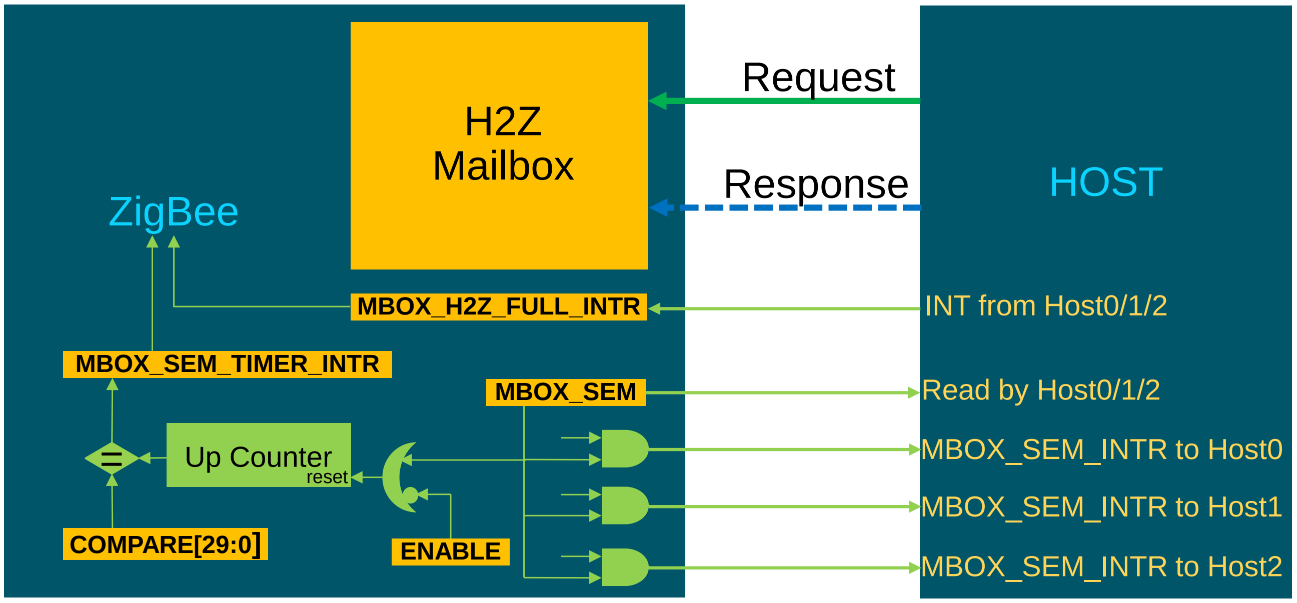


Figure 8: Message from Host to ZigBee

The mail message will be formatted as shown on Figure 9. There are fixed fields to be used for all message frames, even for the multiple frame message cases. The definitions of fields are as below;

- SubSystemID[2:0]: This field will show the source or destination sub-module in Host side. UART is a special case that will be used for the ZigBee software test purpose, and should not be used by Host software.
- Fragment[0]: This flag is to indicate if more message frame(s) will be followed after the current message frame, to handle the multiple frame message that is longer than the size of the Mailbox FIFO. Most of cases, the message will be single frame, and this field will be zero.
- MSG_ID[9:0]: This field is to identify up to 1,024 messages that should be defined later part of this document
- MSG_Type[1:0]: This field is to show the Request, Confirm, Indication, and Response message types.
- MSG Sequence Number[7:0]: This field shows the sequence number of message generated by each sub system. ZigBee, Host0, Host1, Host2, and UART will start with zero for the first message generated, and will increase it by one for each new Request or Indication message. The Confirm and Response message should use the same number from corresponding Request or Indication message.
- Protocol Version[2:0]: This field will show the version of communication protocol, to avoid any communication issue in case there is any change in the future. If the version is beyond what it could be handled, the communication will be terminated with an error message.
- Frame Length[4:0]: The length of current message frame. The total number of bytes of the packet is {(Frame Length + 1) * 4} where 0 <= Frame Length <= 31.
- Payload: This field is for additional parameters for the messages including the callback function pointer and callback data pointer. The number of bytes on the payload is {(Frame Length - 1) * 4}. The structure of this will vary depending on the MSG_ID and MSG_Type.

The payload will be different for each mailbox message, and the definition shall be given through the API header file.

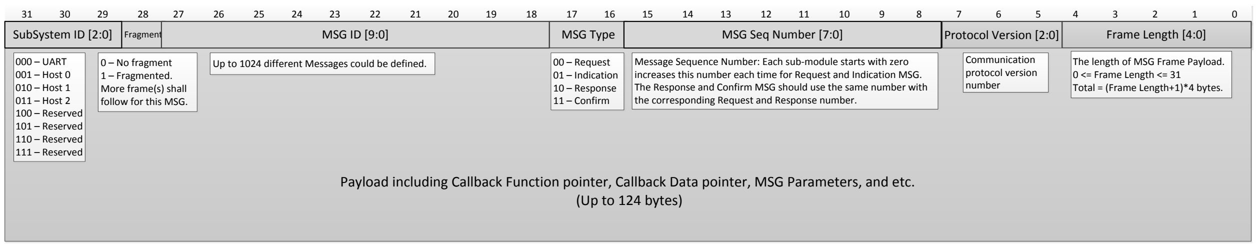

31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
SubSystem ID [2:0]
Fragment
MSG ID [9:0]
MSG Type
MSG Seq Number [7:0]
Protocol Version [2:0]
Frame Length [4:0]
000 – UART
001 – Host 0
010 – Host 1
011 – Host 2
100 – Reserved
101 – Reserved
110 – Reserved
111 – Reserved
0 – No fragment
1 – Fragmented. More frame(s) shall follow for this MSG.
Up to 1024 different Messages could be defined.
00 – Request
01 – Indication
10 – Response
11 – Confirm
Message Sequence Number: Each sub-module starts with zero increases this number each time for Request and Indication MSG. The Response and Confirm MSG should use the same number with the corresponding Request and Response number.
Communication protocol version number
The length of MSG Frame Payload. 0 <= Frame Length <= 31 Total = (Frame Length+1)*4 bytes.
Payload including Callback Function pointer, Callback Data pointer, MSG Parameters, and etc. (Up to 124 bytes)

Figure 9: Message Format

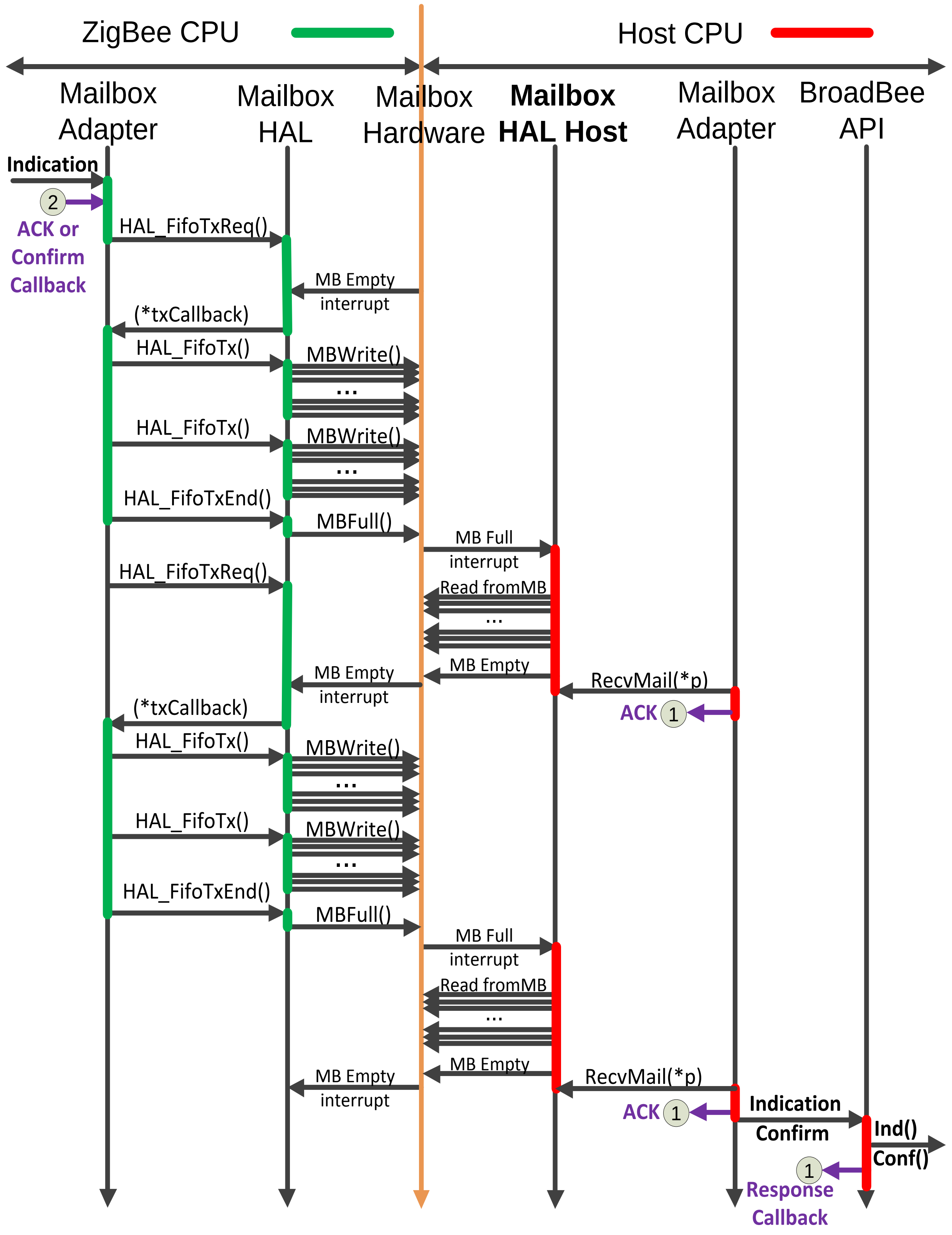


Figure 10: Example of a message from ZigBee to Host

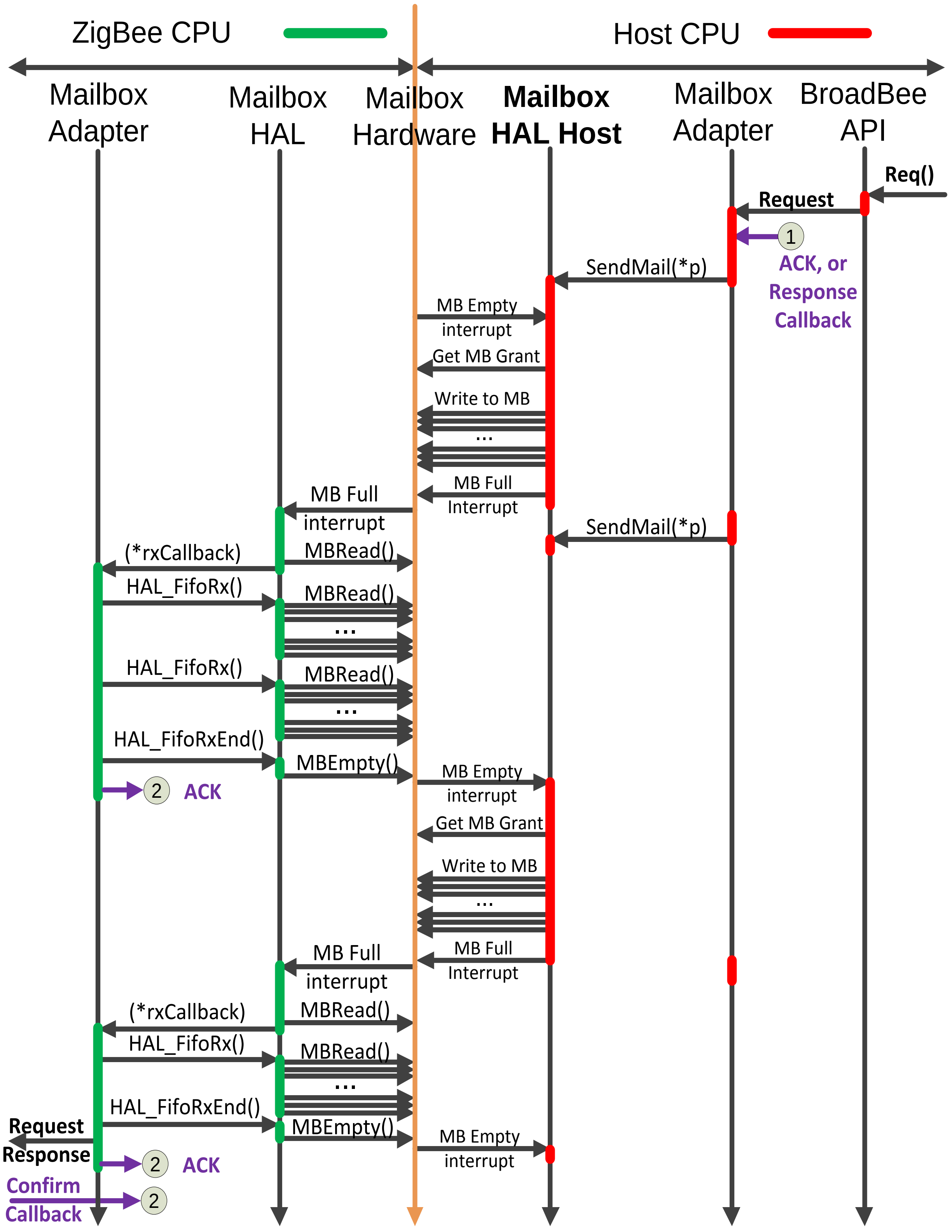


Figure 11: Example of a message from Host to ZigBee

The detail operation of the message over mailbox hardware will be handled by the Mailbox HAL and Mailbox Adapter software on both sides of the mailbox, as shown on the timing chart of Figure 10 and 11. These figures show how the Request and Indication messages are sent across the mailbox, and how the Confirm and Response messages are returned back. Each request and indication packet across the mailbox will be acknolweged by a special ACK packet to confirm the communication between ZigBee and Host CPU’s.

BroadBee API and Mailbox Adapter will be provided by BroadBee software. The **Mailbox HAL Host** should be provided by Host software team to take care of the proper OS interface per each Host CPU.

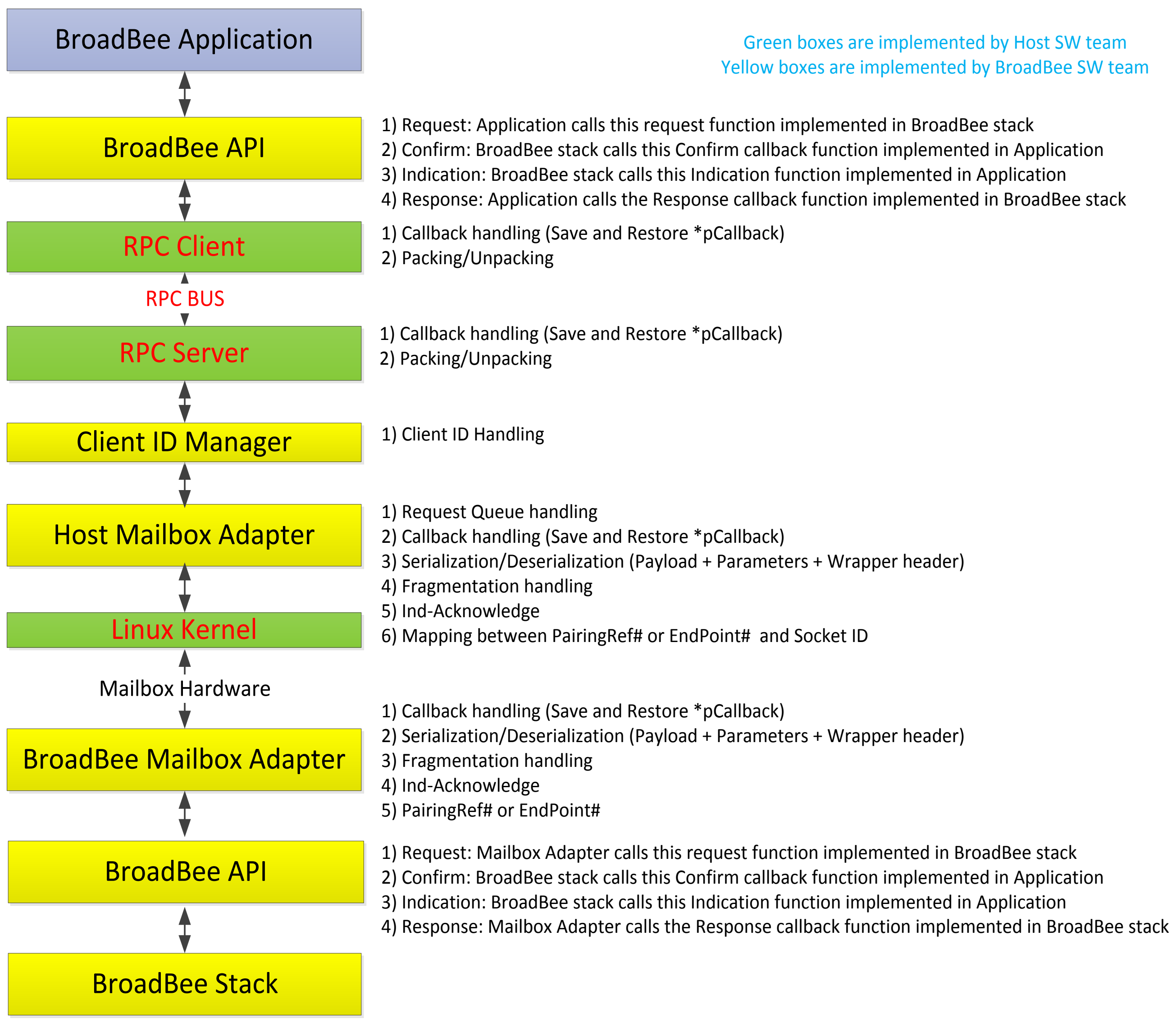

BroadBee Application
Green boxes are implemented by Host SW team
Yellow boxes are implemented by BroadBee SW team
BroadBee API
1) Request: Application calls this request function implemented in BroadBee stack
2) Confirm: BroadBee stack calls this Confirm callback function implemented in Application
3) Indication: BroadBee stack calls this Indication function implemented in Application
4) Response: Application calls the Response callback function implemented in BroadBee stack
RPC Client
1) Callback handling (Save and Restore *pCallback)
2) Packing/Unpacking
RPC BUS
RPC Server
1) Callback handling (Save and Restore *pCallback)
2) Packing/Unpacking
Client ID Manager
1) Client ID Handling
Host Mailbox Adapter
1) Request Queue handling
2) Callback handling (Save and Restore *pCallback)
3) Serialization/Deserialization (Payload + Parameters + Wrapper header)
4) Fragmentation handling
5) Ind-Acknowledge
6) Mapping between PairingRef# or EndPoint# and Socket ID
Linux Kernel
Mailbox Hardware
BroadBee Mailbox Adapter
1) Callback handling (Save and Restore *pCallback)
2) Serialization/Deserialization (Payload + Parameters + Wrapper header)
3) Fragmentation handling
4) Ind-Acknowledge
5) PairingRef# or EndPoint#
BroadBee API
1) Request: Mailbox Adapter calls this request function implemented in BroadBee stack
2) Confirm: BroadBee stack calls this Confirm callback function implemented in Application
3) Indication: BroadBee stack calls this Indication function implemented in Application
4) Response: Mailbox Adapter calls the Response callback function implemented in BroadBee stack
BroadBee Stack

Figure 12: Communication Flow across MailBox

3.1. Mailbox HAL

The Mailbox HAL layer in Host side needs to interface with operating system kernel of Host system to control the mailbox hardware and handle the mailbox interrupts. With this layer, BroadBee software can be independent from the different operating systems used by different Host CPUs.

3.1.1. HAL_MailboxInit()

This function should be called by Mailbox Adapter in Host side to use H2Z and Z2H mailboxes.

Prototype:

```
void HAL_MailboxInit(HOST_HwMailboxDescriptor_t *desc);
```

Brief description:

Request to connect H2Z and Z2H mailboxes.

Parameters:

`Desc`: The software mailbox descriptor.

Return value:

None

3.1.2. HAL_MailboxClose()

This function should be called by Mailbox Adapter in Host to disconnect H2Z and Z2H mailboxes.

Prototype:

```
void HAL_MailboxClose(HOST_HwMailboxDescriptor_t *desc);
```

Brief description:

Request to disconnect H2Z and Z2H mailboxes.

Parameters:

`Desc`: The software mailbox descriptor.

Return value:

None

3.1.3. HAL_MailboxTx()

This function will be called by Mailbox Adapter in Host to send a mail packet to ZigBee through H2Z mailbox. Necessary mailbox hardware control and the mailbox interrupt handling should be taken care of by this mailbox HAL function. The length of packet could be found from “Frame Length[4:0]” of the first 32-bit on the Msg.

It should be better to keep the MBOX_SEM_INTR interrupt masked off normally to ignore the interrupts for H2Z Mailbox empty while there is no message to send through H2Z Mailbox. And, this function should enable the MBOX_SEM_INTR interrupt on own L2 interrupt control register, wait for the interrupt, get the MBOX_SEM=1, then send a message through H2Z Mailbox FIFO.

Prototype:

```
void HAL_MailboxTx(HOST_HwMailboxDescriptor_t *const descr, const
uint8_t *data, uint8_t dataSize);
```

Brief description:

Request to send a mail packet through H2Z mailbox.

Parameters:

`desc`*:* The software mailbox descriptor.

`data`*:* Pointer to mail packet data.

`dataSize`*:* The length of the packet data.

Return value:

None

3.1.4. HAL_MailboxRx()

This function will be called by Mailbox Adapter in Host to receive a mail packet from ZigBee through Z2H mailbox. Necessary mailbox hardware control and the mailbox interrupt handling should be taken care of by this mailbox HAL function. The length of packet could be found from “Frame Length[4:0]” of the first 32-bit on the Msg.

Prototype:

```
Status HAL_MailboxRx (HOST_HwMailboxDescriptor_t *const descr,
uint8_t *buffer, uint8_t length);
```

Brief description:

Request to get a received mail packet through Z2H mailbox

Parameters:

`desc`*:* The software mailbox descriptor.

`buffer`*:* Pointer to buffer receiving data.

`length`*:* The length of the received data.

Return value:

None

3.2. Mailbox Adapter

Mailbox Adapter translates the API function into one or multiple mail packets to deliver the information across the mailbox hardware, and translates the received mail packets into a corresponding API Prototype. After the execution of a Request or an Indication function through the application program, this module will take care of the callback with a Confirm or a Response function as the acknowledgement back to the mail originator side. For some Indication functions without a Response callback function defined, a special acknowledgement callback will be generated by Mailbox Adapter automatically to confirm the mail received by Host.

This Mailbox Adapter module is designed with singleton pattern, and there should be only one instance running in the system simultaneously. It can support multiple threads but only one process. BroadBee software team will provide this software module in source code along with BroadBee API functions.

3.2.1. HOST_HwMailboxDescriptor_t

A structure type used by Mailbox Adapter software.

```
typedef struct _HOST_HwMailboxDescriptor_t
{
    int                                    zigbeeDeviceFd;
    pthread_t                              interruptThread;
    pthread_mutex_t                        rxFifoMutex;
    SYS_FifoDescriptor_t                   txFifo;
    SYS_FifoDescriptor_t                   rxFifo;
    uint8_t
txFifoPage[HAL_MAILBOX_TXRX_FIFO_CAPACITY];
    uint8_t
rxFifoPage[HAL_MAILBOX_TXRX_FIFO_CAPACITY];
    /* Offline callback. */
    HOST_HwMailboxOfflineCallback_t         offlineCallback;
    /* Ready-to-send callback. */
    HOST_HwMailboxReadyToSendCallback_t     rtsCallback;
    /* Data received callback. */
    HOST_HwMailboxDataReceivedCallback_t    rxCallback;
} HOST_HwMailboxDescriptor_t;
```

4. API for RF4CE

4.1. Profiles

4.1.1. API for ZigBee RF4CE Remote Control Profile

The ZigBee RF4CE Remote Control Profile (ZRC) 2.0 is based on the RF4CE GDP profile 2.0 and thus inherits all its functionality. The RF4CE ZRC profile introduces several new profile attributes and one vendor command that performs its functionality. Each of the subsections regarding the RF4CE ZRC profile architecture describes its inheritance dependencies on the correspondent functionality of the RF4CE GDP profile and possible differences.

4.1.1.1. API

4.1.1.1.1. Operations handling

4.1.1.1.1.1. Starting profile

To start the profile user should call RF4CE_StartReq() request. In this case the profile will automatically read attributes from NVM and start MAC, Network and profile. To reset the profile to factory default settings (as well as initializing the new device) the user must call RF4CE_ResetReq() request, then the user should call RF4CE_StartReq() request. After the profile is started the user should tell the profile which device types it supports by calling the RF4CE_SetSupportedDevicesReq() request.

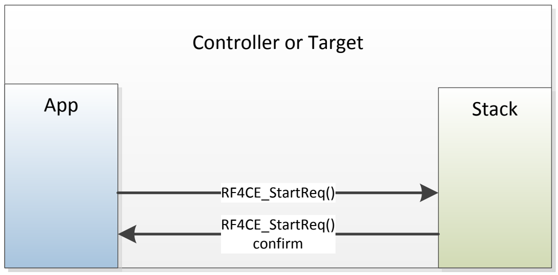


Figure 13: RF4CE ZRC profile start

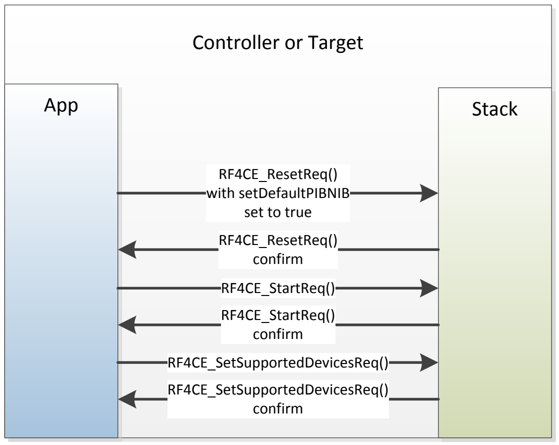

Controller or Target
App
Stack
RF4CE_ResetReq() with setDefaultPIBNIB set to true
RF4CE_ResetReq() confirm
RF4CE_StartReq()
RF4CE_StartReq() confirm
RF4CE_SetSupportedDevicesReq()
RF4CE_SetSupportedDevicesReq() confirm

Figure 14: RF4CE ZRC profile start to factory default settings

4.1.1.1.1.2. Binding handling

The Binding functionality of the RF4CE ZRC profile is inherited from the GDP 2.0 profile. And thus uses the same functions. The configuration step of the button less binding involves the profile attributes exchange between the Controller and the Target: the Controller reads values of the interaction volatile attributes from the Target and then issues the Configuration Complete Request.

The RF4CE ZRC profile actually supports 2 different profiles:

- GDP 1.0 compliant profile with Push Button Based Binding: profile ID 0x01
- GDP 2.0 compliant profile with Validation Based Binding: profile ID 0x03

The Binding response profile ID thus depends on the requested profile ID and if the Push Button was pressed on the Target before the Discovery Request received:

If the Push Button was pressed on the Target before the Discovery request received or the Discovery Request was from the GDP 1.0 compliant Controller then profile ID 0x01 is used. Otherwise profile ID 0x03 is used and subsequent Validation procedure is expected.

4.1.1.1.1.2.1. ZRC 1.1 Binding handling

The RF4CE ZRC 1.1 has its own functionality for binding procedure. In order to bind 2 ZRC 1.1 devices the binding sequence must be executed.
- For Controller. The node must be started. Then the user should issue the RF4CE_ZRC1_ControllerBindReq() request
- For Target. The node must be started. Then the user if the device is a multi-profile device must call the subsequent Bind Disable function (like RF4CE_ZRC_DisableBindingReq() or RF4CE_MSO_TargetDisableBindReq()) to disable buttonless binding. Then the user must call RF4CE_ZRC1_TargetBindReq(). During that request execution the Target should receive the RF4CE_PairInd() indication. Also it is possible to receive RF4CE_UnpairInd() indication if the controller decides to unpair.

4.1.1.1.1.2.2. ZRC 2.0 Binding handling

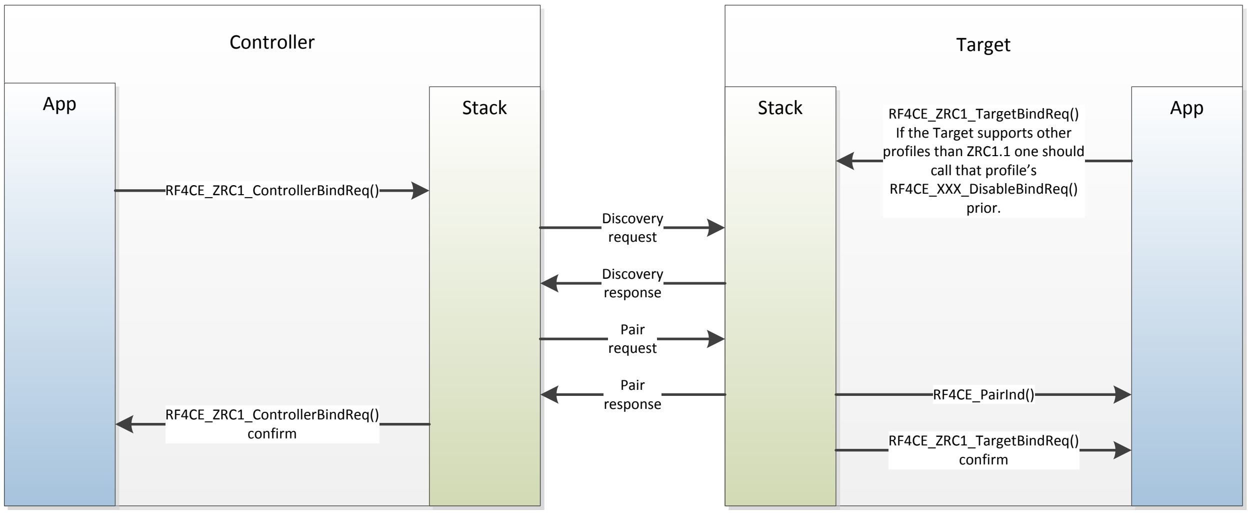

Controller
Target
App
Stack
Stack
App
RF4CE_ZRC1_ControllerBindReq()
RF4CE_ZRC1_TargetBindReq() If the Target supports other profiles than ZRC1.1 one should call that profile's RF4CE_XXX_DisableBindReq() prior.
Discovery request
Discovery response
Pair request
Pair response
RF4CE_PairInd()
RF4CE_ZRC1_ControllerBindReq() confirm
RF4CE_ZRC1_TargetBindReq() confirm

Figure 15: RF4CE ZRC 1.1 Binding Sequence

4.1.1.1.1.3. ZRC 1.1 Command Discovery functionality

The ZRC 1.1 profile has Command Discovery functionality. Each node has the subsequent attribute to be set by the Host via RF4CE_ZRC1_SetAttributesReq() request. Once this is done any bound node can issue the RF4CE_ZRC1_CommandDiscoveryReq() in order to get the bit mask of the supported by the remote node codes.

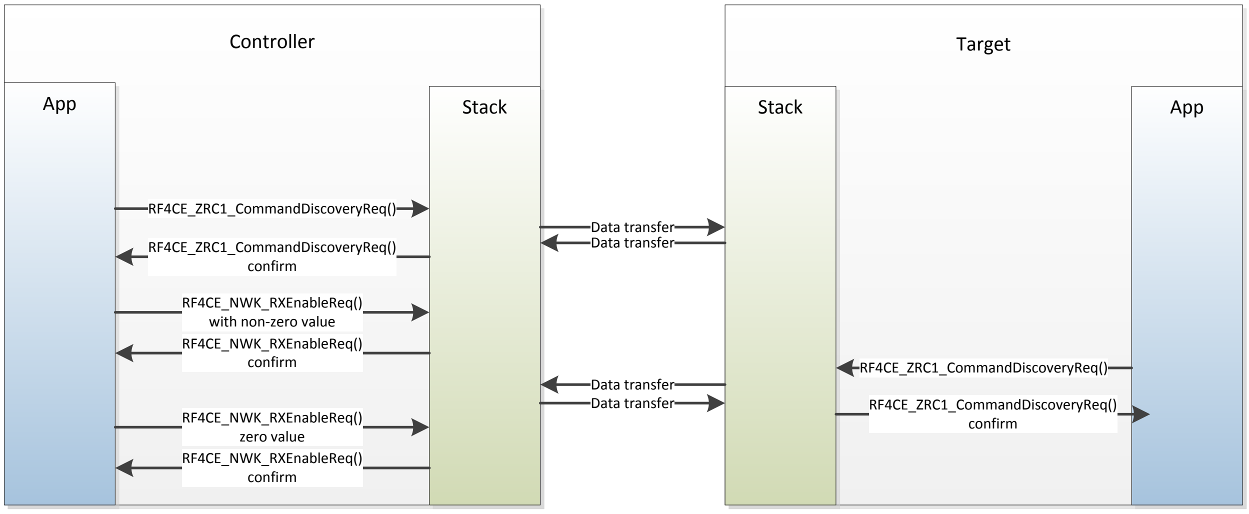

Controller
Target
App
Stack
Stack
App
RF4CE_ZRC1_CommandDiscoveryReq()
Data transfer
Data transfer
RF4CE_ZRC1_CommandDiscoveryReq() confirm
RF4CE_NWK_RXEnableReq() with non-zero value
RF4CE_NWK_RXEnableReq() confirm
RF4CE_ZRC1_CommandDiscoveryReq()
Data transfer
Data transfer
RF4CE_ZRC1_CommandDiscoveryReq() confirm
RF4CE_NWK_RXEnableReq() zero value
RF4CE_NWK_RXEnableReq() confirm

Figure 16: RF4CE ZRC 1.1 Command Discovery function

4.1.1.1.1.4. ZRC 1.1 Control Command handling

The ZRC 1.1 main functionality is sending the key codes of the pressed buttons. There are 2 functions on the Controller side RF4CE_ZRC1_ControlCommandPressedReq() and RF4CE_ZRC1_ControlCommandReleasedReq() to implement that.

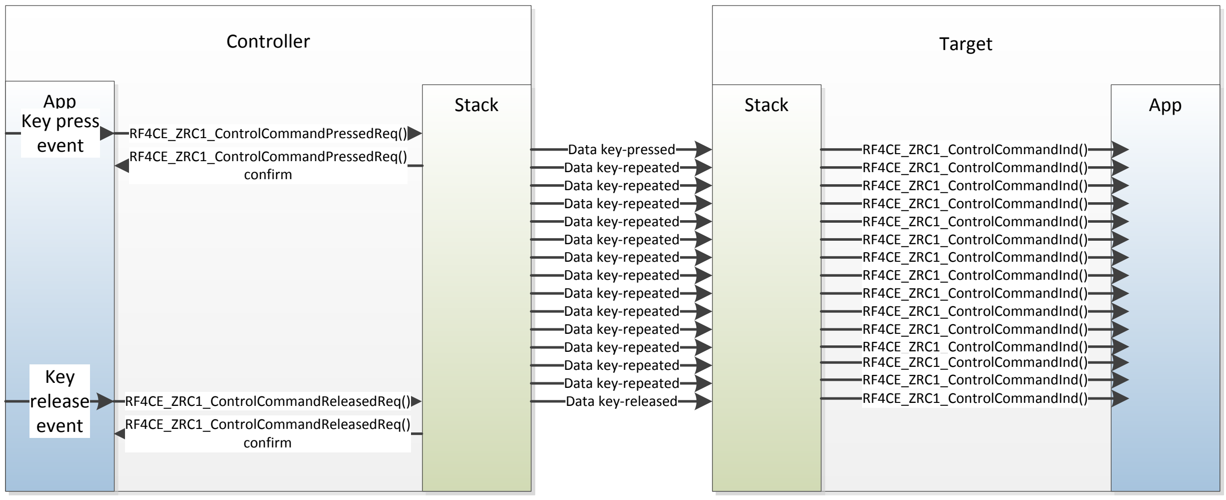

Controller
Target
App
Stack
Stack
App
Key press event
RF4CE_ZRC1_ControlCommandPressedReq()
RF4CE_ZRC1_ControlCommandPressedReq() confirm
Data key-pressed
Data key-repeated
Data key-released
RF4CE_ZRC1_ControlCommandInd()
Key release event
RF4CE_ZRC1_ControlCommandReleasedReq()
RF4CE_ZRC1_ControlCommandReleasedReq() confirm

Figure 17: RF4CE ZRC 1.1 Control Command function

4.1.1.1.1.5. Profile attributes handling

The Profile attributes handling functionality of the RF4CE ZRC profile is inherited from the GDP 2.0 profile. And thus uses the same functions. However the RF4CE ZRC profile introduces new attributes.

4.1.1.1.1.6. Controller notification handling

The Controller Notification functionality of the RF4CE ZRC profile is inherited from the GDP 2.0 profile. And thus uses the same functions.

4.1.1.1.1.7. Key Exchange handling

The Key Exchange functionality of the RF4CE ZRC profile is inherited from the GDP 2.0 profile. And thus uses the same functions.

4.1.1.1.1.8. Heartbeat handling

The Heartbeat functionality of the RF4CE ZRC profile is inherited from the GDP 2.0 profile. And thus uses the same functions.

4.1.1.1.2. Common API

This chapter describes common ZRC 1.1 and ZRC 2.0 functions and their descriptors including data types and required parameters.

A common place for all functions in this chapter would be Request Service type:

```
typedef struct _RF4CE_NWK_RequestService_t
{
    SYS_QueueElement_t serviceData; /*!< Helper field to allow that
structure object to be queued. */
    uint8_t requestID; /*!< Request ID. */
} RF4CE_NWK_RequestService_t;
```

Figure 18: Request Service type

Descriptors sctructure is common and contains service field descriptor, Callback type and Parameters type.

4.1.1.1.2.1. RF4CE_ZRC1_GetAttributesReq()

Brief description:

Starts asynchronous ZRC 1.1 Get Attributes Request.

Prototype

```
void RF4CE_ZRC1_GetAttributesReq(RF4CE_ZRC1_GetAttributeDescr_t
*request);
```

Where `request` is a *pointer to the request descriptor.*

Function descriptor

```
struct _RF4CE_ZRC1_GetAttributeDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;
#endif /* _HOST_ */
    RF4CE_ZRC1_GetAttributeReqParams_t params;
    RF4CE_ZRC1_GetAttributeCallback_t callback;
};
```

Service field type Brief description

See **Figure 18: Request Service type**

Callback type

```
typedef void (*RF4CE_ZRC1_GetAttributeCallback_t)
(RF4CE_ZRC1_GetAttributeDescr_t *req,
RF4CE_ZRC1_GetAttributeConfParams_t *conf);
```

Parameters type

```
typedef struct _RF4CE_ZRC1_GetAttributeReqParams_t
{
    uint8_t attributeId;
} RF4CE_ZRC1_GetAttributeReqParams_t;
```

Callback Parameters type

```
typedef struct _RF4CE_ZRC1_GetAttributeConfParams_t
{
    uint8_t status;
    RF4CE_ZRC1_Attribute_t data;
} RF4CE_ZRC1_GetAttributeConfParams_t;
```

4.1.1.1.2.2. RF4CE_ZRC1_SetAttributesReq()

Brief description:

Starts asynchronous ZRC 1.1 Set Attributes Request .

Prototype

```
void RF4CE_ZRC1_SetAttributesReq(RF4CE_ZRC1_SetAttributeDescr_t
*request);
```

Where `request` is a *pointer to the request descriptor.*

Function descriptor

```
struct _RF4CE_ZRC1_SetAttributeDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;
#endif /* _HOST_ */
    RF4CE_ZRC1_SetAttributeReqParams_t params;
    RF4CE_ZRC1_SetAttributeCallback_t callback;
};
```

Service field type Brief description

See **Figure 18: Request Service type**

Callback type

```
typedef void (*RF4CE_ZRC1_SetAttributeCallback_t)
(RF4CE_ZRC1_SetAttributeDescr_t *req,
RF4CE_ZRC1_SetAttributeConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC1_SetAttributeReqParams_t
{
    uint8_t attributeId;
    RF4CE_ZRC1_Attribute_t data;
} RF4CE_ZRC1_SetAttributeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC1_SetAttributeConfParams_t
{
    uint8_t status;
} RF4CE_ZRC1_SetAttributeConfParams_t;
```

4.1.1.1.2.3. RF4CE_ZRC1_CommandDiscoveryReq()

Starts sending ZRC 1.1 Command Discovery request.

Prototype

```
void
RF4CE_ZRC1_CommandDiscoveryReq(RF4CE_ZRC1_CommandDiscoveryReqDescr_t
*request);
```

Where `request` is a pointer to the ZRC 1.1 Command Discovery request descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC1_CommandDiscoveryReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;
#endif /* _HOST_ */
    RF4CE_ZRC1_CommandDiscoveryReqParams_t params;
    RF4CE_ZRC1_CommandDiscoveryCallback_t callback;
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC1_CommandDiscoveryCallback_t)
(RF4CE_ZRC1_CommandDiscoveryReqDescr_t *req,
RF4CE_ZRC1_CommandDiscoveryConfParams_t *conf);
```

Parameters type:

```
typedef struct PACKED _RF4CE_ZRC1_CommandDiscoveryReqParams_t
{
    uint8_t pairingRef;                  /*!< The pairing reference. */
} RF4CE_ZRC1_CommandDiscoveryReqParams_t;
```

Callback Parameters type:

```
typedef struct PACKED _RF4CE_ZRC1_CommandDiscoveryConfParams_t
{
    RF4CE_ZRC1_CommandDiscoveryStatus_t status; /*!< Status of the
operation. */
    uint8_t bitmap[32];                         /*!< The requested
bitmap. */
} RF4CE_ZRC1_CommandDiscoveryConfParams_t;
```

4.1.1.1.2.4. RF4CE_ZRC1_ControlCommandPressedReq()

Starts sending ZRC 1.1 Control Command.

Prototype:

```
void
RF4CE_ZRC1_ControlCommandPressedReq(RF4CE_ZRC1_ControlCommandReqDescr_t
*request);
```

Where `request` is a `pointer to the ZRC 1.1 Control Command request descriptor structure.`

Descriptor:

```
struct _RF4CE_ZRC1_ControlCommandReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service; /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_ZRC1_ControlCommandReqParams_t params;  /*!< Request
parameters. */
    RF4CE_ZRC1_ControlCommandCallback_t callback; /*!< Request
confirmation callback. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC1_ControlCommandCallback_t)
(RF4CE_ZRC1_ControlCommandReqDescr_t *req,
RF4CE_ZRC_ControlCommandConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC1_ControlCommandReqParams_t
{
    uint8_t pairingRef;                 /*!< Pairing reference. */
    uint8_t commandCode;                /*!< The command code. */
    SYS_DataPointer_t payload;          /*!< Possible additional data.
*/
} RF4CE_ZRC1_ControlCommandReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC_ControlCommandConfParams_t
{
    uint8_t status; /*!< The status of the operation. */
} RF4CE_ZRC_ControlCommandConfParams_t;
```

4.1.1.1.2.5. RF4CE_ZRC1_ControlCommandReleasedReq()

Ends sending ZRC 1.1 Control Command.

Prototype:

```
void
RF4CE_ZRC1_ControlCommandReleasedReq(RF4CE_ZRC1_ControlCommandReqDescr_
t *request);
```

Where `request` is a `pointer to the ZRC 1.1 Control Command request descriptor structure.`

Descriptor:

```
struct _RF4CE_ZRC1_ControlCommandReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;           /*!< Service field.
*/
#endif /* _HOST_ */
    RF4CE_ZRC1_ControlCommandReqParams_t params;  /*!< Request
parameters. */
    RF4CE_ZRC1_ControlCommandCallback_t callback; /*!< Request
confirmation callback. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC1_ControlCommandCallback_t)
(RF4CE_ZRC1_ControlCommandReqDescr_t *req,
RF4CE_ZRC_ControlCommandConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC1_ControlCommandReqParams_t
{
    uint8_t pairingRef;                 /*!< Pairing reference. */
    uint8_t commandCode;                /*!< The command code. */
    SYS_DataPointer_t payload;          /*!< Possible additional data.
*/
} RF4CE_ZRC1_ControlCommandReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC_ControlCommandConfParams_t
{
    uint8_t status; /*!< The status of the operation. */
} RF4CE_ZRC_ControlCommandConfParams_t;
```

4.1.1.1.2.6. RF4CE_ZRC1_VendorSpecificReq()

Starts Vendor Specific sending

Prototype:

```
void RF4CE_ZRC1_VendorSpecificReq(
    RF4CE_ZRC1_VendorSpecificReqDescr_t *request);
```

Where request is a `pointer to the request.`

Descriptor:

```
struct _RF4CE_ZRC1_VendorSpecificReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;            /*!< Service field.
*/
#endif /* _HOST_ */
    RF4CE_ZRC1_VendorSpecificReqParams_t params;  /*!< Request params.
*/
    RF4CE_ZRC1_VendorSpecificCallback_t callback; /*!< Callback on
request completion. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC1_VendorSpecificCallback_t)
(RF4CE_ZRC1_VendorSpecificReqDescr_t *req,
RF4CE_ZRC1_VendorSpecificConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC1_VendorSpecificReqParams_t
{
    uint8_t pairingRef;         /*!< The pairing reference. */
    uint16_t vendorID;          /*!< The Vendor ID. */
    SYS_DataPointer_t payload;  /*!< The payload to be sent. */
} RF4CE_ZRC1_VendorSpecificReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC1_VendorSpecificConfParams_t
{
    RF4CE_ZRC1_VendorSpecificStatus_t status;     /*!< The status of
sending. */
} RF4CE_ZRC1_VendorSpecificConfParams_t;
```

4.1.1.1.2.7. RF4CE_ZRC2_GetAttributesReq()

Starts asynchronous ZRC 2.0 Get Attributes Request.

Prototype:

```
void rf4cezrc2GetAttributesReq(RF4CE_ZRC2_GetAttributesReqDescr_t
*request);
```

Where request is a pointer `to the` get attributes request descriptor `structure.`

Descriptor:

```
struct _RF4CE_ZRC2_GetAttributesReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;          /*!< Service field */
    uint8_t pairingRef;                          /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_GetAttributesReqParams_t params;     /*!< Request
parameters */
    RF4CE_ZRC2_GetAttributesReqCallback_t callback; /*!< Request
callback */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_GetAttributesReqCallback_t)
(RF4CE_ZRC2_GetAttributesReqDescr_t *req,
RF4CE_ZRC2_GetAttributesConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC2_GetAttributesReqParams_t
{
    uint8_t pairingRef;        /*!< Pairing reference. If it is equal
to RF4CE_NWK_INVALID_PAIRING_REF then local attributes are retrieved */
    SYS_DataPointer_t payload; /*!< The payload must contain an array
of RF4CE_ZRC2_AttributeId_t records */
} RF4CE_ZRC2_GetAttributesReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_GetAttributesConfParams_t
{
    uint8_t status;            /*!< Overall status of the operation.
One of the RF4CE_ZRC2GDP2_GetAttributeStatus_t values */
    SYS_DataPointer_t payload; /*!< The payload must contain an array
of aligned RF4CE_ZRC2_GetAttributesConfId_t records */
} RF4CE_ZRC2_GetAttributesConfParams_t;
```

4.1.1.1.2.8. RF4CE_ZRC2_SetAttributesReq()

Starts asynchronous ZRC 2.0 Set Attributes Request.

Prototype:

```
void RF4CE_ZRC2_SetAttributesReq(RF4CE_ZRC2_SetAttributesReqDescr_t
*request);
```

Where request is a pointer to the set attributes request descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_SetAttributesReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;        /*!< Service field */
    uint8_t pairingRef;                        /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_SetAttributesReqParams_t params;     /*!< Request
parameters */
    RF4CE_ZRC2_SetAttributesReqCallback_t callback; /*!< Request
callback */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_SetAttributesReqCallback_t)
(RF4CE_ZRC2_SetAttributesReqDescr_t *req,
RF4CE_ZRC2_SetAttributesConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC2_SetAttributesReqParams_t
{
    uint8_t pairingRef;        /*!< Pairing reference. If it is equal
to RF4CE_NWK_INVALID_PAIRING_REF then local attributes are retrieved */
    SYS_DataPointer_t payload; /*!< The payload must contain an array
of aligned RF4CE_ZRC2_SetAttributeId_t records */
} RF4CE_ZRC2_SetAttributesReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_SetAttributesConfParams_t
{
    uint8_t status;     /*!< The status of the operation: one of the
RF4CE_ZRC2GDP2_Status_t values */
} RF4CE_ZRC2_SetAttributesConfParams_t;
```

4.1.1.1.2.9. RF4CE_ZRC2_KeyExchangeReq()

Starts RF4CE ZRC 2.0 Key Exchange procedure.

Prototype:

```
void RF4CE_ZRC2_KeyExchangeReq(RF4CE_ZRC2_KeyExchangeReqDescr_t
*request);
```

Where `request` is a pointer `to the` Key Exchange request descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_KeyExchangeReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;       /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_ZRC2_KeyExchangeReqParams_t params;   /*!< Filled by
application request structure */
    RF4CE_ZRC2_KeyExchangeCallback_t callback;  /*!< The request
confirmation callback */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_KeyExchangeCallback_t)
(RF4CE_ZRC2_KeyExchangeReqDescr_t *req,
RF4CE_ZRC2_KeyExchangeConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC2_KeyExchangeReqParams_t
{
    uint8_t pairingRef;        /*!< Pairing reference */
    uint8_t flags;             /*!< First byte of Key Exchange flags.
See GDP 2.0 Spec r29 Figure 23. */
    uint8_t vendorSpecific;    /*!< Second byte of Key Exchange flags.
See GDP 2.0 Spec r29 Figure 23. */
} RF4CE_ZRC2_KeyExchangeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_KeyExchangeConfParams_t
{
    uint8_t status;             /*!< Status of the operation: one of
the RF4CE_ZRC2GDP2_Status_t values */
} RF4CE_ZRC2_KeyExchangeConfParams_t;
```

4.1.1.1.2.10. RF4CE_ZRC2_CheckValidationResp()

Starts ZRC 2.0 Check Validation Response.

Funtion call:

```
void
RF4CE_ZRC2_CheckValidationResp(RF4CE_ZRC2_CheckValidationRespDescr_t
*response);
```

Where response is a pointer to the response structure.

Descriptor:

```
typedef struct _RF4CE_ZRC2_CheckValidationRespDescr_t
{
    RF4CE_ZRC2_CheckValidationRespParams_t params; /*!< Response
parameters */
} RF4CE_ZRC2_CheckValidationRespDescr_t;
```

Parameters Brief description:

```
typedef struct _RF4CE_ZRC2_CheckValidationRespParams_t
{
    uint8_t pairingRef; /*!< Pairing reference of the Check Validation
response */
    uint8_t status;     /*!< Validation status. One of the
RF4CE_ZRC2_ValidationStatus_t values */
} RF4CE_ZRC2_CheckValidationRespParams_t;
```

4.1.1.1.2.11. RF4CE_ZRC2_ControlCommandPressedReq()

Starts sending ZRC 2.0 Control Command Pressed action.

Prototype:

```
void
RF4CE_ZRC2_ControlCommandPressedReq(RF4CE_ZRC2_ControlCommandReqDescr_t
*request);
```

Where `request` is a `pointer to the ZRC 2.0 Control Command request descriptor`.

Descriptor:

```
struct _RF4CE_ZRC2_ControlCommandReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;    /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_ZRC2_ControlCommandReqParams_t params;    /*!< Request
parameters. */
```

```
    RF4CE_ZRC2_ControlCommandCallback_t callback;   /*!< Request
confirmation callback. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_ControlCommandCallback_t)
(RF4CE_ZRC2_ControlCommandReqDescr_t *req,
RF4CE_ZRC2_ControlCommandConfParams_t *conf);
```

Parameters type:

```
  typedef struct _RF4CE_ZRC2_ControlCommandReqParams_t
  {
      uint8_t pairingRef;                  /*!< Pairing reference. */
      SYS_DataPointer_t payload;           /*!< Supplied payload
  consisting of one or more RF4CE_ZRC2_Action_t and/or
                                                RF4CE_ZRC2_ActionVendor_t
  structures. */
} RF4CE_ZRC2_ControlCommandReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_ControlCommandConfParams_t
{
    RF4CE_ZRC2_ControlCommandConfStatus_t status; /*!< The status of
the operation. */
} RF4CE_ZRC2_ControlCommandConfParams_t;
```

4.1.1.1.2.12. RF4CE_ZRC2_ControlCommandReleasedReq()

Starts sending ZRC 2.0 Control Command Released action.

Prototype:

```
void
RF4CE_ZRC2_ControlCommandReleasedReq(RF4CE_ZRC2_ControlCommandReqDescr_
t *request);
```

Where `request` is a `pointer to the ZRC 2.0 Control Command request descriptor`.

Descriptor:

```
struct _RF4CE_ZRC2_ControlCommandReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;      /*!< Service field. */
#endif /* _HOST_ */
```

```
    RF4CE_ZRC2_ControlCommandReqParams_t params;    /*!< Request
parameters. */
    RF4CE_ZRC2_ControlCommandCallback_t callback;   /*!< Request
confirmation callback. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_ControlCommandCallback_t)
(RF4CE_ZRC2_ControlCommandReqDescr_t *req,
RF4CE_ZRC2_ControlCommandConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC2_ControlCommandReqParams_t
{
    uint8_t pairingRef;                 /*!< Pairing reference. */
    SYS_DataPointer_t payload;          /*!< Supplied payload
consisting of one or more RF4CE_ZRC2_Action_t and/or
                                            RF4CE_ZRC2_ActionVendor_t
structures. */
} RF4CE_ZRC2_ControlCommandReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_ControlCommandConfParams_t
{
    RF4CE_ZRC2_ControlCommandConfStatus_t status; /*!< The status of
the operation. */
} RF4CE_ZRC2_ControlCommandConfParams_t;
```

4.1.1.1.2.13. RF4CE_UnpairReq()

Performs unbinding with the remote node.

Prototype:

```
void RF4CE_UnpairReq(RF4CE_UnpairReqDescr_t *request);
```

Where `request` is a pointer `to the` unpair request descriptor `structure.`

Descriptor:

```
struct _RF4CE_UnpairReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;
#endif /* _HOST_ */
    RF4CE_UnpairReqParams_t params;  /*!< The request parameters */
    RF4CE_UnpairCallback_t callback; /*!< The request confirmation
callback */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_UnpairCallback_t) (RF4CE_UnpairReqDescr_t *req,
RF4CE_UnpairConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_UnpairReqParams_t
{
    uint8_t pairingRef; /*!< The existing pairing reference */
} RF4CE_UnpairReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_UnpairConfParams_t
{
    Bool8_t status; /*!< The status of the operation */
} RF4CE_UnpairConfParams_t;
```

4.1.1.1.2.14. RF4CE_StartReq()

Starts standard MAC + NWK + profile(s) Start procedure.

Prototype:

```
void RF4CE_ZRC_Start(RF4CE_StartReqDescr_t *request);
```

Where `request` is a pointer `to the` Start request descriptor structure.

Descriptor:

```
typedef struct _RF4CE_StartReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service; /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_StartConfCallback_t callback; /*!< Callback for confirmation.
*/
} RF4CE_StartReqDescr_t;
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_StartConfCallback_t) (RF4CE_StartReqDescr_t *req,
RF4CE_StartResetConfParams_t *conf);
```

Callback Parameters type:

```
typedef struct _RF4CE_StartResetConfParams_t
{
    uint8_t status;     /*!< The status of the START/RESET request. One
of the RF4CE_StartReset_Status_t values.  */
} RF4CE_StartResetConfParams_t;
```

4.1.1.1.2.15. RF4CE_ResetReq()

Starts standard MAC + NWK + profile(s) Reset procedure.

Prototype:

```
void RF4CE_ResetReq(RF4CE_ResetReqDescr_t *request);
```

Where `request` is a pointer `to the` Reset request descriptor structure.

Descriptor:

```
struct _RF4CE_ResetReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service; /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_ResetReqParams_t params;      /*!< Request parameters. */
    RF4CE_ResetConfCallback_t callback; /*!< Callback for confirmation.
*/
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ResetConfCallback_t) (RF4CE_ResetReqDescr_t *req,
RF4CE_StartResetConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_ResetReqParams_t
{
    Bool8_t setDefaultPIBNIB;  /*!< Set default values for NIB and PIB.
*/
} RF4CE_ResetReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_StartResetConfParams_t
```

```
{
    uint8_t status;     /*!< The status of the START/RESET request. One
of the RF4CE_StartReset_Status_t values.  */
} RF4CE_StartResetConfParams_t;
```

4.1.1.1.2.16. RF4CE_SetSupportedDevicesReq()

Sets Supported Device List.

Prototype:

```
void RF4CE_SetSupportedDevicesReq(RF4CE_SetSupportedDevicesReqDescr_t
*request);
```

Where `request` is a pointer to the Set request descriptor structure.

Descriptor:

```
struct _RF4CE_SetSupportedDevicesReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;
#endif /* _HOST_ */
    RF4CE_SetSupportedDevicesReqParams_t params;
    RF4CE_GDP_SetSupportedDevicesCallback_t callback;
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_GDP_SetSupportedDevicesCallback_t)
(RF4CE_SetSupportedDevicesReqDescr_t *req,
RF4CE_SetSupportedDevicesConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_SetSupportedDevicesReqParams_t
{
    uint8_t numDevices;
    uint8_t devices[3];
} RF4CE_SetSupportedDevicesReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_SetSupportedDevicesConfParams_t
{
    Bool8_t status;
} RF4CE_SetSupportedDevicesConfParams_t;
```

4.1.1.1.2.17. RF4CE_ZRC2_BindReq()

Starts asynchronous Binding Procedure.

Prototype:

```
void RF4CE_ZRC2_BindReq(RF4CE_ZRC2_BindReqDescr_t *request);
```

Where `request` is a pointer `to the B`ind request descriptor structure.

Descriptor:

```
typedef struct _RF4CE_ZRC2_BindReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;  /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_BindCallback_t callback; /*!< Callback on request
completion. */
} RF4CE_ZRC2_BindReqDescr_t;
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_BindCallback_t) (RF4CE_ZRC2_BindReqDescr_t
*req, RF4CE_ZRC2_BindConfParams_t *conf);
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_BindConfParams_t
{
    uint8_t status;     /*!< The status of binding. See
RF4CE_ZRC2_BindStatus_t. */
    uint8_t pairingRef; /*!< The pairing reference on successful
binding. */
    uint8_t profileId;  /*!< Actual profile the node is bound to:
RF4CE_ZRC_GDP1_COMPLIANT_PROTOCOL_ID or
RF4CE_ZRC_GDP2_COMPLIANT_PROTOCOL_ID */
} RF4CE_ZRC2_BindConfParams_t;
```

4.1.1.1.2.18. RF4CE_NWK_SetReq ()

Starts asynchronous Binding Procedure.

Prototype:

```
void RF4CE_NWK_SetReq(RF4CE_NWK_SetReqDescr_t *request);
```

Where `request` is a pointer `to the B`ind request descriptor structure.

Descriptor:

```
typedef struct _RF4CE_NWK_SetReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;   /*!< Service field */
#else
    void *context;
#endif /* _HOST_ */
    RF4CE_NWK_SetReqParams_t params;   /*!< Request containing
structure */
    RF4CE_NWK_SetConfCallback_t callback; /*!< Callback for
confirmation. */
} RF4CE_NWK_SetReqDescr_t;
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_NWK_SetConfCallback_t) (RF4CE_NWK_SetReqDescr_t
*req, RF4CE_NWK_SetConfParams_t *conf);
```

Function parameters type:

```
typedef struct _RF4CE_NWK_SetReqParams_t
{
    RF4CE_NWK_AttributeID_t attrId;     /*!< NIB attribute
identification. */
    RF4CE_NIB_AttributesAll_t data;     /*!< New attribute's value. */
} RF4CE_NWK_SetReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_NWK_SetConfParams_t
{
    uint8_t status;                     /*!< The result of the request
to write the NIB attribute. */
    RF4CE_NWK_AttributeID_t attrId;     /*!< NIB attribute
identification. */
} RF4CE_NWK_SetConfParams_t;
```

4.1.1.1.2.19. RF4CE_PairInd()

Indication to the HOST on Pair indication data.

Prototype:

```
void RF4CE_PairInd(RF4CE_PairingIndParams_t *indication);
```

Where `indication` is a pointer to the indication structure.

Function parameters type:

```
typedef struct _RF4CE_PairingIndParams_t
{
    RF4CE_PairingIndStatus_t status;    /*!< The pairing status */
    uint8_t pairingRef;              /*!< The pairing reference */
} RF4CE_PairingIndParams_t;
```

4.1.1.1.2.20. RF4CE_UnpairInd()

Indication to the HOST on Unpair indication data.

Prototype:

```
void RF4CE_UnpairInd(RF4CE_PairingReferenceIndParams_t *indication);
```

Where `indication` is a pointer to the indication structure

Function parameters type:

```
typedef struct _RF4CE_PairingReferenceIndParams_t
{
    uint8_t pairingRef;                /*!< The pairing reference */
} RF4CE_PairingReferenceIndParams_t;
```

4.1.1.1.2.21. RF4CE_CounterExpiredInd()

Indication to the HOST on Counter Expired error.

Prototype:

```
void RF4CE_CounterExpiredInd(RF4CE_PairingReferenceIndParams_t
*indication);
```

Where `indication` is a pointer to the indication structure.

Function parameters type:

```
typedef struct _RF4CE_PairingReferenceIndParams_t
{
    uint8_t pairingRef;                          /*!< The pairing
reference */
} RF4CE_PairingReferenceIndParams_t;
```

4.1.1.1.3. Controller only API

4.1.1.1.3.1. RF4CE_ZRC1_ControllerBindReq()

Starts Controller side Binding Procedure.

Prototype:

```
void RF4CE_ZRC1_ControllerBindReq(RF4CE_ZRC1_BindReqDescr_t *request);
```

Where `request` is a `pointer to the request.`

Descriptor:

```
struct _RF4CE_ZRC1_BindReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service; /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_ZRC1_BindCallback_t callback; /*!< Callback on request
completion. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC1_BindCallback_t) (RF4CE_ZRC1_BindReqDescr_t
*req, RF4CE_ZRC1_BindConfParams_t *conf);
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC1_BindConfParams_t
{
    uint8_t status;     /*!< The status of binding. See
RF4CE_ZRC1_BindStatus_t. */
    uint8_t pairingRef; /*!< The pairing reference on successful
binding. */
} RF4CE_ZRC1_BindConfParams_t;
```

4.1.1.1.3.2. RF4CE_ZRC2_ProxyBindReq()

Starts asynchronous Proxy Binding Procedure

Prototype:

```
void RF4CE_ZRC2_ProxyBindReq(RF4CE_ZRC2_ProxyBindReqDescr_t *request);
```

Where request is a pointer to the proxy bind request descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_ProxyBindReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;      /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_ProxyBindReqParams_t params;  /*!< Request parameters.
*/
    RF4CE_ZRC2_ProxyBindCallback_t callback; /*!< Callback on request
completion. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_ProxyBindCallback_t)
(RF4CE_ZRC2_ProxyBindReqDescr_t *req, RF4CE_ZRC2_BindConfParams_t
*conf);
```

Parameters type:

```
typedef struct _RF4CE_ZRC2_ProxyBindReqParams_t
{
    uint64_t address;   /*!< The known remote host's address. */
} RF4CE_ZRC2_ProxyBindReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_BindConfParams_t
{
    uint8_t status;     /*!< The status of binding. See
RF4CE_ZRC2_BindStatus_t. */
    uint8_t pairingRef; /*!< The pairing reference on successful
binding. */
    uint8_t profileId;  /*!< Actual profile the node is bound to:
RF4CE_ZRC_GDP1_COMPLIANT_PROTOCOL_ID or
RF4CE_ZRC_GDP2_COMPLIANT_PROTOCOL_ID */
} RF4CE_ZRC2_BindConfParams_t;
```

4.1.1.1.4. Target only API

4.1.1.1.4.1. RF4CE_ZRC1_TargetBindReq()

Starts Target side Binding Procedure.

Prototype:

```
void RF4CE_ZRC1_TargetBindReq(RF4CE_ZRC1_BindReqDescr_t *request);
```

Where `request` is a pointer to the reqest.

Descriptor:

```
struct _RF4CE_ZRC1_BindReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service; /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_ZRC1_BindCallback_t callback; /*!< Callback on request
completion. */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC1_BindCallback_t) (RF4CE_ZRC1_BindReqDescr_t
*req, RF4CE_ZRC1_BindConfParams_t *conf);
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC1_BindConfParams_t
{
    uint8_t status;     /*!< The status of binding. See
RF4CE_ZRC1_BindStatus_t. */
    uint8_t pairingRef; /*!< The pairing reference on successful
binding. */
} RF4CE_ZRC1_BindConfParams_t;
```

4.1.1.1.4.2. RF4CE_ZRC1_ControlCommandInd()

Indication to the HOST on ZRC 1.1 Control Command.

Prototype:

```
void RF4CE_ZRC1_ControlCommandInd(RF4CE_ZRC1_ControlCommandIndParams_t
*indication);
```

Where `indication` is a pointer to the indication structure.

Function Parameters type:

```
typedef struct _RF4CE_ZRC1_ControlCommandIndParams_t
{
    uint8_t pairingRef;        /*!< Pairing reference. */
    uint8_t flags;             /*!< RF4CE_ZRC1_USER_CONTROL_PRESSED or
RF4CE_ZRC1_USER_CONTROL_REPEATED or RF4CE_ZRC1_USER_CONTROL_RELEASED.
*/
    uint8_t commandCode;       /*!< Command code */
    SYS_DataPointer_t payload; /*!< Possible accompanying payload */
} RF4CE_ZRC1_ControlCommandIndParams_t;
```

4.1.1.1.4.3. RF4CE_ZRC1_VendorSpecificInd()

Indication to the HOST on ZRC 1.1 Vendor Specific.

Prototype:

```
void RF4CE_ZRC1_VendorSpecificInd(RF4CE_ZRC1_VendorSpecificIndParams_t
*indication);
```

Where `indication` is a pointer to the indication structure.

Function Parameters type:

```
typedef struct _RF4CE_ZRC1_VendorSpecificReqParams_t
{
    uint8_t pairingRef;         /*!< The pairing reference. */
    uint16_t vendorID;          /*!< The Vendor ID. */
    SYS_DataPointer_t payload;  /*!< The payload to be sent. */
} RF4CE_ZRC1_VendorSpecificReqParams_t;
```

4.1.1.1.4.4. RF4CE_ZRC2_CheckValidationInd()

ZRC 2.0 Check Validation indication.

Prototype:

```
void
RF4CE_ZRC2_StartValidationInd(RF4CE_ZRC2_CheckValidationIndParams_t
*indication);
```

Where `indication` is a pointer to the Check Validation indication structure.

Function Parameters type:

```
typedef struct _RF4CE_ZRC2_CheckValidationIndParams_t
{
    uint8_t pairingRef;      /*!< Pairing reference of the Check
Validation request */
    uint8_t isAutoValidated; /*!< true if automatical validation took
place */
} RF4CE_ZRC2_CheckValidationIndParams_t;
```

4.1.1.1.4.5. RF4CE_ZRC2_EnableBindingReq()

Enables Binding Procedure on the Target.

Prototype:

```
void RF4CE_ZRC2_EnableBindingReq(RF4CE_ZRC2_BindingReqDescr_t
*request);
```

Where `request` is a pointer to the Binding Request descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_BindingReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;       /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_BindingCallback_t callback;    /*!< Pointer to the
callback  */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_BindingCallback_t)
(RF4CE_ZRC2_BindingReqDescr_t *req, RF4CE_ZRC2_BindingConfParams_t
*conf);
```

Callback Parameters type:

```
typedef void (*RF4CE_ZRC2_BindingCallback_t)
(RF4CE_ZRC2_BindingReqDescr_t *req, RF4CE_ZRC2_BindingConfParams_t
*conf);
```

4.1.1.1.4.6. RF4CE_ZRC2_DisableBindingReq()

Disables Binding Procedure on the Target.

Prototype:

```
void RF4CE_ZRC2_DisableBindingReq(RF4CE_ZRC2_BindingReqDescr_t
*request);
```

Where `request` is a pointer to the Binding Request descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_BindingReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;       /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_BindingCallback_t callback;    /*!< Pointer to the
callback  */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_BindingCallback_t)
(RF4CE_ZRC2_BindingReqDescr_t *req, RF4CE_ZRC2_BindingConfParams_t
*conf);
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_BindingConfParams_t
{
    uint8_t status; /*!< One of the RF4CE_ZRC2GDP2_Status_t values */
} RF4CE_ZRC2_BindingConfParams_t;
```

4.1.1.1.4.7. RF4CE_ZRC2_ButtonBindingReq()

Starts Button Pressed Binding Procedure on the Target.

Prototype:

```
void RF4CE_ZRC2_ButtonBindingReq(
  RF4CE_ZRC2_ButtonBindingReqDescr_t *request);
```

Where `request` is a pointer to the Button Binding Request descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_ButtonBindingReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;          /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_ButtonBindingReqParams_t params;  /*!< Request
parameters. */
    RF4CE_ZRC2_ButtonBindingCallback_t callback; /*!< Pointer to the
callback  */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_ButtonBindingCallback_t)
(RF4CE_ZRC2_ButtonBindingReqDescr_t *req,
RF4CE_ZRC2_BindingConfParams_t *conf);
```

Function parameters type:

```
typedef struct _RF4CE_ZRC2_ButtonBindingReqParams_t
{
    uint32_t autoDiscDuration;       /*!< The maximum number of MAC
symbols NLME will be in auto discovery response mode. */
} RF4CE_ZRC2_ButtonBindingReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_BindingConfParams_t
{
    uint8_t status; /*!< One of the RF4CE_ZRC2GDP2_Status_t values */
} RF4CE_ZRC2_BindingConfParams_t;
```

4.1.1.1.4.8. RF4CE_ZRC2_ControlCommandInd()

ZRC 2.0 Control Command indication.

Prototype:

```
void RF4CE_ZRC1_ControlCommandInd(RF4CE_ZRC1_ControlCommandIndParams_t
*indication);
```

Where `indication` is a pointer to the ZRC Control Command indication data structure.

Function parameters type:

```
typedef struct _RF4CE_ZRC1_ControlCommandIndParams_t
{
    uint8_t pairingRef;         /*!< Pairing reference. */
    uint8_t flags;              /*!< RF4CE_ZRC1_USER_CONTROL_PRESSED or
RF4CE_ZRC1_USER_CONTROL_REPEATED or RF4CE_ZRC1_USER_CONTROL_RELEASED.
*/
    uint8_t commandCode;        /*!< Command code */
    SYS_DataPointer_t payload; /*!< Possible accompanying payload */
} RF4CE_ZRC1_ControlCommandIndParams_t;
```

4.1.1.1.4.9. RF4CE_ZRC2_StartValidationInd()

Indication to the HOST on ZRC 2.0 or GDP 2.0 validation beginning.

Prototype:

```
void
RF4CE_ZRC2_StartValidationInd(RF4CE_ZRC2_CheckValidationIndParams_t
*indication);
```

Where `indication` is a pointer to the indication structure.

Function parameters type:

```
typedef struct _RF4CE_ZRC1_ControlCommandIndParams_t
{
    uint8_t pairingRef;         /*!< Pairing reference. */
    uint8_t flags;              /*!< RF4CE_ZRC1_USER_CONTROL_PRESSED or
RF4CE_ZRC1_USER_CONTROL_REPEATED or RF4CE_ZRC1_USER_CONTROL_RELEASED.
*/
    uint8_t commandCode;        /*!< Command code */
    SYS_DataPointer_t payload; /*!< Possible accompanying payload */
} RF4CE_ZRC1_ControlCommandIndParams_t;
```

4.1.1.1.4.10. RF4CE_ZRC2_PairNtfyInd()

Indication to the HOST on Pair indication data.

Prototype:

```
void RF4CE_ZRC2_PairNtfyInd(RF4CE_PairingIndParams_t *indication);
```

Where `indication` is a pointer to the indication structure.

Function parameters type:

```
typedef struct _RF4CE_PairingIndParams_t
{
    RF4CE_PairingIndStatus_t status;    /*!< The pairing status */
    uint8_t pairingRef;              /*!< The pairing reference */
} RF4CE_PairingIndParams_t;
```

4.1.1.1.4.11. RF4CE_ZRC2_BindingFinishedNtfyInd()

Indication to the HOST on Pair indication data.

Prototype:

```
void
RF4CE_ZRC2_BindingFinishedNtfyInd(RF4CE_ZRC2_BindingFinishedNtfyIndPara
ms_t *indication);
```

Where `indication` is a pointer to the indication structure.

Function parameters type:

```
typedef struct _RF4CE_ZRC2_BindingFinishedNtfyIndParams_t
{
    uint8_t status;     /*!< The status of binding. See
RF4CE_ZRC2_BindStatus_t. */
    uint8_t pairingRef; /*!< The pairing reference on successful
binding. */
    uint8_t profileId;  /*!< Actual profile the node is bound to:
RF4CE_ZRC_GDP1_COMPLIANT_PROTOCOL_ID or
RF4CE_ZRC_GDP2_COMPLIANT_PROTOCOL_ID */
} RF4CE_ZRC2_BindingFinishedNtfyIndParams_t;
```

4.1.1.1.4.12. RF4CE_ZRC2_SetPushButtonStimulusReq()

Request to the host on push button pairing initiation.

Prototype:

```
void
RF4CE_ZRC2_SetPushButtonStimulusReq(RF4CE_ZRC2_ButtonBindingReqDescr_t
*request);
```

Where `request` is a pointer to the Push Button Stimulus descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_ButtonBindingReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;          /*!< Service field */
#endif /* _HOST_ */
    RF4CE_ZRC2_ButtonBindingReqParams_t params;  /*!< Request
parameters. */
    RF4CE_ZRC2_ButtonBindingCallback_t callback; /*!< Pointer to the
callback  */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_ButtonBindingCallback_t)
(RF4CE_ZRC2_ButtonBindingReqDescr_t *req,
RF4CE_ZRC2_BindingConfParams_t *conf);
```

Function parameters type:

```
typedef struct _RF4CE_ZRC2_ButtonBindingReqParams_t
{
    uint32_t autoDiscDuration;       /*!< The maximum number of MAC
symbols NLME will be in auto discovery response mode. */
} RF4CE_ZRC2_ButtonBindingReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_BindingConfParams_t
{
    uint8_t status; /*!< One of the RF4CE_ZRC2GDP2_Status_t values */
} RF4CE_ZRC2_BindingConfParams_t;
```

4.1.1.1.4.13. RF4CE_ZRC2_ClearPushButtonStimulusReq()

Clears Push Button Stimulus flag on the Target.

Prototype:

```
void
RF4CE_ZRC2_ClearPushButtonStimulusReq(RF4CE_ZRC2_ButtonBindingReqDescr_
t *request);
```

Where `request` is a pointer to the Push Button Stimulus descriptor structure.

Descriptor:

```
struct _RF4CE_ZRC2_ButtonBindingReqDescr_t
{
    RF4CE_NWK_RequestService_t service;          /*!< Service field */
    RF4CE_ZRC2_ButtonBindingReqParams_t params;  /*!< Request
parameters. */
    RF4CE_ZRC2_ButtonBindingCallback_t callback; /*!< Pointer to the
callback  */
};
```

Service field type Brief description:

See **Figure 18: Request Service type**

Callback type:

```
typedef void (*RF4CE_ZRC2_ButtonBindingCallback_t)
(RF4CE_ZRC2_ButtonBindingReqDescr_t *req,
RF4CE_ZRC2_BindingConfParams_t *conf);
```

Function parameters type:

```
typedef struct _RF4CE_ZRC2_ButtonBindingReqParams_t
{
    uint32_t autoDiscDuration;       /*!< The maximum number of MAC
symbols NLME will be in auto discovery response mode. */
} RF4CE_ZRC2_ButtonBindingReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC2_BindingConfParams_t
{
    uint8_t status; /*!< One of the RF4CE_ZRC2GDP2_Status_t values */
} RF4CE_ZRC2_BindingConfParams_t;
```

4.1.2. API for ZigBee RF4CE MSO Profile

The ZigBee RF4CE Cable Profile for Remote Control (MSO) is an extension of the RF4CE protocol that is used to control any cable device supporting this profile.

The MSO profile in fact is not based on any of the existing RF4CE profiles; and thus represents a fully standalone profile operating within the RF4CE network.

4.1.2.1. Enumerations

4.1.2.1.1. RF4CE_MSO_RIBAttributeID_t

RF4CE MSO protocol RIB attributes IDs.

```
typedef enum _RF4CE_MSO_RIBAttributeID_t
{
    /* RW. Identifiers of the Peripherals */
    RF4CE_MSO_RIB_PEREFERAL_IDS = 0x00,
    /* RW. RF Statistics */
    RF4CE_MSO_RIB_RF_STATISTICS,
    /* RW. Versions of different parts of the device */
    RF4CE_MSO_RIB_VERSIONING,
    /* RW. Controller battery status information */
    RF4CE_MSO_RIB_BATTERY_STATUS,
    /* RO. The maximum time in us a unicast acknowledged
           multichannel transmission shall be retried in case the
           Short RF Retry configuration is set. */
    RF4CE_MSO_RIB_SHORT_RF_RETRY_PERIOD,
    /* RO. IR and RF codes for different keys */
    RF4CE_MSO_RIB_IRRF_DATABASE = 0xDB,
    /* RO. Configurable properties of the validation procedure */
    RF4CE_MSO_RIB_VALIDATION_CONFIGURATION,
    /* RW. General purpose remote storage */
    RF4CE_MSO_RIB_GENERAL_PURPOSE = 0xFF
} RF4CE_MSO_RIBAttributeID_t;
```

4.1.2.1.2. RF4CE_MSO_RIB_Versioning_t

RF4CE MSO protocol RIB attributes versioning indexes for access IDs.

```
typedef enum _RF4CE_MSO_RIB_Versioning_t
{
    RF4CE_MSO_RIB_VERSIONING_SW = 0x00,
    RF4CE_MSO_RIB_VERSIONING_HW = 0x01,
    RF4CE_MSO_RIB_VERSIONING_IRDB = 0x02
} RF4CE_MSO_RIB_Versioning_t;
```

4.1.2.1.3. RF4CE_MSO_ProfileAttributeID_t

RF4CE MSO protocol profile attributes for access IDs.

```
typedef enum _RF4CE_MSO_ProfileAttributeID_t
{
    /* Controller. The interval in ms at which user command repeat
                   frames will be transmitted for repeatable keys.*/
    RF4CE_MSO_APL_KEY_REPEAT_INTERVAL = 0,
    /* Controller. The maximum time in symbols that a device SHALL
                    wait (after the aplcResponseIdleWaitTime
                    expired) to receive a response command frame
                    following a request command frame. */
    RF4CE_MSO_APL_RESPONSE_WAIT_TIME,
    /* Controller. The maximum number of pairing candidates
                    selected from the NLME-DISCOVERY.response node
                    descriptor list. */
    RF4CE_MSO_APL_MAX_PAIRING_CANDIDATES,
    /* Controller. The maximum time in ms that a device can stay
                    in the validation procedure without receiving
                    the responses corresponding to its requests.
                    [Can be updated by RIB procedure at the start
                    of the validation procedure.] */
    RF4CE_MSO_APL_LINK_LOST_WAIT_TIME,
    /* Controller. The time period in ms between the regular check
                    validation requests that a controller transmits
                    in the validation procedure. [Can be updated by
                    RIB procedure at the start of the validation.]
                    */
    RF4CE_MSO_APL_AUTO_CHECK_VALIDATION_PERIOD,
    /* Controller. The value of the KeyExTransferCount parameter
                    passed to the pair request primitive during the
                    temporary pairing procedure. */
    RF4CE_MSO_APL_KEY_EXCHANGE_TRANSFER_TIME,
    /* Target. The duration in ms that a recipient of a user
                 control repeated command frame waits before
                 terminating a repeated operation. */
    RF4CE_MSO_APL_KEY_REPEAT_WAIT_TIME = 0x10,
    /* Target. The maximum time in ms that a device can stay in
                 the validation procedure. */
    RF4CE_MSO_APL_VALIDATION_WAIT_TIME,
    /* Target. The maximum time in ms that a device can stay in
                 the validation procedure, without receiving a first
                 validation watchdog kick. */
    RF4CE_MSO_APL_VALIDATION_INITIAL_WATCHDOG_TIME,
    /* Target and Controller. The user-defined character string to
                                carry application-related
                                information. */
    RF4CE_MSO_APL_USER_STRING = 0x20
} RF4CE_MSO_ProfileAttributeID_t;
```

4.1.2.1.4. RF4CE_MSO_ProfileAttributeStatus_t

RF4CE MSO profile attributes GET/SET status enumeration.

```
typedef enum _RF4CE_MSO_ProfileAttributeStatus_t
{
    RF4CE_MSO_PA_SUCCESS = 0,
    RF4CE_MSO_PA_UNSUPPORTED_ID,
    RF4CE_MSO_PA_INVALID
} RF4CE_MSO_ProfileAttributeStatus_t;
```

4.1.2.1.5. RF4CE_MSO_RIBAttributeStatus_t

RF4CE MSO RIB attributes status codes.

```
typedef enum _RF4CE_MSO_RIBAttributeStatus_t
{
    RF4CE_MSO_RIB_SUCCESS = 0,
    RF4CE_MSO_RIB_INVALID_PARAMETER,
    RF4CE_MSO_RIB_UNSUPPORTED_ATTRIBUTE,
    RF4CE_MSO_RIB_INVALID_INDEX
} RF4CE_MSO_RIBAttributeStatus_t;
```

4.1.2.2. API

This chapter describes common RF4CE MSO Profile including data types, functions and corresponding parameters.

A common place for all functions in this chapter would be Request Service type:

```
typedef struct _RF4CE_NWK_RequestService_t
{
    SYS_QueueElement_t serviceData; /*!< Helper field to allow that
structure object to be queued. */
    uint8_t requestID; /*!< Request ID. */
} RF4CE_NWK_RequestService_t;
```

Figure 19: Request Service Type

Descriptors sctructure is common and contains service field descriptor, Callback type and Parameters type.

4.1.2.2.1. Internal profile attributes

4.1.2.2.1.1. Functions

4.1.2.2.1.1.1. RF4CE_MSO_GetProfileAttributeReq()

Starts RF4CE MSO Get Profile Attribute.

Prototype:

```
void
RF4CE_MSO_GetProfileAttributeReq(RF4CE_MSO_GetProfileAttributeReqDescr_
t *request);
```

Where `request` is a pointer to the Get Profile Attribute request descriptor structure.

Descriptor:

```
struct _RF4CE_MSO_GetProfileAttributeReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;      /*!< Service field */
#endif /* _HOST_ */
    RF4CE_MSO_GetProfileAttributeReqParams_t params;  /*!< Request
parameters */
    RF4CE_MSO_GetProfileAttributeCallback_t callback; /*!< Request
callback */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*RF4CE_MSO_GetProfileAttributeCallback_t)
(RF4CE_MSO_GetProfileAttributeReqDescr_t *req,
RF4CE_MSO_GetProfileAttributeConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_MSO_GetProfileAttributeReqParams_t
{
    uint8_t id;        /*!< One of the RF4CE_MSO_ProfileAttributeID_t
values */
} RF4CE_MSO_GetProfileAttributeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_MSO_GetProfileAttributeConfParams_t
{
    uint8_t status;    /*!< One of the
RF4CE_MSO_ProfileAttributeStatus_t values */
    uint8_t id;        /*!< One of the RF4CE_MSO_ProfileAttributeID_t
values */
    RF4CE_MSO_ProfileAttributesUnion_t attribute; /*!< The resulting
value */
} RF4CE_MSO_GetProfileAttributeConfParams_t;
```

4.1.2.2.1.1.2. RF4CE_MSO_SetProfileAttributeReq()

Starts RF4CE MSO Set Profile Attribute.

Prototype:

```
void
RF4CE_MSO_SetProfileAttributeReq(RF4CE_MSO_SetProfileAttributeReqDescr_
t *request);
```

Where `request` is a pointer to the Set Profile Attribute request descriptor structure.

Descriptor:

```
struct _RF4CE_MSO_SetProfileAttributeReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;       /*!< Service field */
#endif /* _HOST_ */
    RF4CE_MSO_SetProfileAttributeReqParams_t params;  /*!< Request
parameters */
```

```
    RF4CE_MSO_SetProfileAttributeCallback_t callback; /*!< Request
callback */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*RF4CE_MSO_SetProfileAttributeCallback_t)
(RF4CE_MSO_SetProfileAttributeReqDescr_t *req,
RF4CE_MSO_SetProfileAttributeConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_MSO_SetProfileAttributeReqParams_t
{
    uint8_t id;        /*!< One of the RF4CE_MSO_ProfileAttributeID_t
values */
    RF4CE_MSO_ProfileAttributesUnion_t attribute; /*!< The value for
that attribute */
} RF4CE_MSO_SetProfileAttributeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_MSO_SetProfileAttributeConfParams_t
{
    uint8_t status;    /*!< One of the
RF4CE_MSO_ProfileAttributeStatus_t values */
    uint8_t id;        /*!< One of the RF4CE_MSO_ProfileAttributeID_t
values */
} RF4CE_MSO_SetProfileAttributeConfParams_t;
```

4.1.2.2.2. Remote Information Base Support

4.1.2.2.2.1. HOST Side API

4.1.2.2.2.1.1. RF4CE_MSO_GetRIBInd()

Prototype:

```
void RF4CE_MSO_GetRIBInd(
  RF4CE_MSO_GetRIBAttributeReqDescr_t *request);
```

Brief description:

Get RF4CE MSO RIB Attribute from the HOST.

Parameters:

`request`: pointer to the request descriptor structure.

Return value:

None.

4.1.2.2.2.1.2. RF4CE_MSO_SetRIBInd()

Prototype:

```
void RF4CE_MSO_SetRIBInd(
  RF4CE_MSO_SetRIBAttributeReqDescr_t *request);
```

Brief description:

Set RF4CE MSO RIB Attribute to the HOST.

Parameters:

`request`: pointer to the request descriptor structure.

Return value:

None.

4.1.2.2.2.2. Functions

4.1.2.2.2.2.1. RF4CE_MSO_GetRIBAttributeReq()

Initializes the RF4CE MSO RIB Attribute GET Request.

Prototype:

```
void RF4CE_MSO_GetRIBAttributeReq(RF4CE_MSO_GetRIBAttributeReqDescr_t
*request);
```

Where `request` is a pointer to the request descriptor structure.

Descriptor:

```
struct _RF4CE_MSO_GetRIBAttributeReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;            /*!< Service field.
*/
#endif /* _HOST_ */
    RF4CE_MSO_GetRIBAttributeReqParams_t params;  /*!< Request
parameters. */
    RF4CE_MSO_GetRIBAttributeCallback_t callback; /*!< Request
callback. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*RF4CE_MSO_GetRIBAttributeCallback_t)
(RF4CE_MSO_GetRIBAttributeReqDescr_t *req,
RF4CE_MSO_GetRIBAttributeConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_MSO_GetRIBAttributeReqParams_t
{
    uint8_t pairingRef;        /*!< Pairing reference for the request.
*/
    uint8_t attributeID;       /*!< Requested attribute ID. */
    uint8_t attributeIndex;    /*!< Requested attribute Index. */
    uint8_t valueLength;       /*!< Requested attribute length in
bytes. */
} RF4CE_MSO_GetRIBAttributeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_MSO_GetRIBAttributeConfParams_t
{
    uint8_t status;            /*!< Requested attribute status: one of
the RF4CE_MSO_RIBAttributeStatus_t codes */
    uint8_t attributeId;       /*!< Requested attribute ID. */
    uint8_t attributeIndex;    /*!< Requested attribute Index. */
    SYS_DataPointer_t payload; /*!< The requested attribute value. */
} RF4CE_MSO_GetRIBAttributeConfParams_t;
```

4.1.2.2.2.2.2. RF4CE_MSO_SetRIBAttributeReq()

Initializes the RF4CE MSO RIB Attribute SET Request.

Prototype:

```
void RF4CE_MSO_GetRIBAttributeReq(RF4CE_MSO_GetRIBAttributeReqDescr_t
*request);
```

Where `request` is a pointer to the request descriptor structure.

Descriptor:

```
struct _RF4CE_MSO_GetRIBAttributeReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;           /*!< Service field.
*/
#endif /* _HOST_ */
    RF4CE_MSO_GetRIBAttributeReqParams_t params;  /*!< Request
parameters. */
    RF4CE_MSO_GetRIBAttributeCallback_t callback; /*!< Request
callback. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*RF4CE_MSO_GetRIBAttributeCallback_t)
(RF4CE_MSO_GetRIBAttributeReqDescr_t *req,
RF4CE_MSO_GetRIBAttributeConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_MSO_GetRIBAttributeReqParams_t
{
    uint8_t pairingRef;         /*!< Pairing reference for the request.
*/
    uint8_t attributeID;        /*!< Requested attribute ID. */
    uint8_t attributeIndex;     /*!< Requested attribute Index. */
    uint8_t valueLength;        /*!< Requested attribute length in
bytes. */
} RF4CE_MSO_GetRIBAttributeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_MSO_GetRIBAttributeConfParams_t
{
    uint8_t status;             /*!< Requested attribute status: one of
the RF4CE_MSO_RIBAttributeStatus_t codes */
    uint8_t attributeId;        /*!< Requested attribute ID. */
    uint8_t attributeIndex;     /*!< Requested attribute Index. */
    SYS_DataPointer_t payload; /*!< The requested attribute value. */
} RF4CE_MSO_GetRIBAttributeConfParams_t;
```

4.1.2.2.3. Binding Support

4.1.2.2.3.1. HOST Side API

4.1.2.2.3.1.1. RF4CE_MSO_CheckValidationInd()

Reflects the FIRST Check Validation Request to the HOST.

Prototype:

```
void RF4CE_MSO_CheckValidationInd(RF4CE_MSO_CheckValidationIndParams_t
*indication);
```

Where `indication` is a pointer to the indication structure.

Parameters type:

```
typedef struct _RF4CE_MSO_CheckValidationIndParams_t
{
    uint8_t pairingRef;
    uint8_t flags;
} RF4CE_MSO_CheckValidationIndParams_t;
```

4.1.2.2.3.2. Controller only API

4.1.2.2.3.2.1. RF4CE_MSO_BindReq()

Starts binding procedure.

Prototype:

```
void RF4CE_MSO_GetRIBAttributeReq(RF4CE_MSO_GetRIBAttributeReqDescr_t
*request);
```

Where `request` is a pointer to the request descriptor structure.

Descriptor:

```
struct _RF4CE_MSO_BindReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service; /*!< Service field. */
#endif /* _HOST_ */
    RF4CE_MSO_BindCallback_t callback;  /*!< Callback on request
completion. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*RF4CE_MSO_BindCallback_t) (RF4CE_MSO_BindReqDescr_t
*req, RF4CE_MSO_BindConfParams_t *conf);
```

Callback Parameters type:

```
typedef struct _RF4CE_MSO_BindConfParams_t
{
    uint8_t status;     /*!< One of the RF4CE_MSO_BindStatus_t values
*/
    uint8_t pairingRef; /*!< The pairing reference value on the
successful binding */
} RF4CE_MSO_BindConfParams_t;
```

4.1.2.2.4. User Control Command Support

4.1.2.2.4.1. HOST Side API

4.1.2.2.4.1.1. RF4CE_MSO_UserControlInd()

MSO User Control command indication to HOST.

Prototype:

```
void RF4CE_MSO_CheckValidationInd(RF4CE_MSO_CheckValidationIndParams_t
*indication);
```

Where `indication` is a pointer to the indication structure.

Parameters type:

```
typedef struct _RF4CE_MSO_CheckValidationIndParams_t
{
    uint8_t pairingRef;
    uint8_t flags;
} RF4CE_MSO_CheckValidationIndParams_t;
```

Prototype:

```
void RF4CE_MSO_UserControlInd(
  RF4CE_MSO_UserControlIndParams_t *indication);
```

Brief description:

Parameters:

`indication`: pointer to the indication structure.

Return value:

None.

4.1.2.2.4.2. Controller only API

4.1.2.2.4.2.1. RF4CE_MSO_UserControlReq()

Initiates MSO User Control request.

Prototype:

```
void RF4CE_MSO_GetRIBAttributeReq(RF4CE_MSO_GetRIBAttributeReqDescr_t
*request);
```

Where `request` is a pointer to the request descriptor structure.

Descriptor:

```
struct _RF4CE_MSO_GetRIBAttributeReqDescr_t
{
#ifndef _HOST_
    RF4CE_NWK_RequestService_t service;           /*!< Service field.
*/
#endif /* _HOST_ */
    RF4CE_MSO_GetRIBAttributeReqParams_t params;  /*!< Request
parameters. */
    RF4CE_MSO_GetRIBAttributeCallback_t callback; /*!< Request
callback. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*RF4CE_MSO_GetRIBAttributeCallback_t)
(RF4CE_MSO_GetRIBAttributeReqDescr_t *req,
RF4CE_MSO_GetRIBAttributeConfParams_t *conf);
```

Parameters type:

```
typedef struct _RF4CE_MSO_GetRIBAttributeReqParams_t
{
    uint8_t pairingRef;        /*!< Pairing reference for the request.
*/
    uint8_t attributeID;       /*!< Requested attribute ID. */
    uint8_t attributeIndex;    /*!< Requested attribute Index. */
    uint8_t valueLength;       /*!< Requested attribute length in
bytes. */
} RF4CE_MSO_GetRIBAttributeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_MSO_GetRIBAttributeConfParams_t
{
    uint8_t status;            /*!< Requested attribute status: one of
the RF4CE_MSO_RIBAttributeStatus_t codes */
    uint8_t attributeId;       /*!< Requested attribute ID. */
    uint8_t attributeIndex;    /*!< Requested attribute Index. */
    SYS_DataPointer_t payload; /*!< The requested attribute value. */
} RF4CE_MSO_GetRIBAttributeConfParams_t;
```

4.2. ZigBee NWK sublayer

4.2.1. Basic Configuration

4.2.1.1. Attribute setting

4.2.1.1.1. Functions

4.2.1.1.1.1. ZBPRO_NWK_SetKeyReq()

Initiates Set Network Key request.

Prototype:

```
void ZBPRO_NWK_SetKeyReq(ZBPRO_NWK_SetKeyReqDescr_t *req);
```

Where `request` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_NWK_SetKeyReqDescr_t
{
    ZbProNwkGetSetServiceField_t    service;
    ZBPRO_NWK_SetKeyReqParams_t     params;
    ZBPRO_NWK_SetKeyCallback_t      callback;
} ZBPRO_NWK_SetKeyReqDescr_t;
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*ZBPRO_NWK_SetKeyCallback_t) (ZBPRO_NWK_SetKeyReqDescr_t
*const reqDescr,ZBPRO_NWK_SetKeyConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_NWK_SetKeyReqParams_t
{
    ZbProSspKey_t                   key;
    ZbProSspNwkKeySeqNum_t          keyCounter;
} ZBPRO_NWK_SetKeyReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_NWK_SetKeyConfParams_t
{
    ZbProSspNwkKeySeqNum_t          keyCounter;
    ZBPRO_NWK_Status_t              status;
} ZBPRO_NWK_SetKeyConfParams_t;
```

5. API for ZigBee PRO

5.1. ZigBee Home Automation Profile

This section specifies the APIs to communicate with the ZigBee Home Automation Application Profile on BroadBee ZigBee PRO software stack. All of these API's are designed to work asynchronously and go through the MailBoxes between ZigBee CPU and application CPUs.

5.1.1. Supported Device Types

There are many Home Automation devices that can be supported, and currently BroadBee supports the devices listed on Table 1. Depend on the needs from customers, more devices could be added.

Domain	Device ID	Home Automation Device
Generic	0x0000	On/Off Switch
Generic	0x0001	Level Control Switch
Generic	0x0004	Scene Selector
Generic	0x0005	Configuration Tool
Generic	0x0006	Remote Control
Generic	0x0007	Combined Interface
Generic	0x000B	Door Lock Controller
Lighting	0x0103	On/Off Light Switch
Lighting	0x0104	Dimmer Switch
Lighting	0x0105	Color Dimmer Switch
Closure	0x0201	Shade Controller
Closure	0x0203	Window Covering Controller
HVAC	0x0304	Pump Controller
IAS	0x0400	IAS Control and Indicating Equipment

Table 1: Home Automation Devices

5.1.2. Supported Clusters

In ZigBee, a cluster is a related collection of commands and attributes to define the interface to specific functionality. ZigBee Cluster Library acts as a repository of cluster functionality defined by ZigBee to work with all ZigBee devices. This has employed the client/server model. An entity that stores the attributes of a cluster is referred to as the server of that cluster. An entity that affects or manipulates those attributes is referred to as the client of that cluster.
Each device needs to support a certain number of clusters, and can be either server or client device. Considering of system's functionality, BroadBee has chosen the clusters on server and client devices as listed on Table 2. Depend on the needs from customers, more clusters could be added.

Func Domain	Cluster ID	Home Automation Cluster	Server	Client
General	0x0000	Basic	v	
General	0x0003	Identify	v	v
General	0x0004	Groups		v
General	0x0005	Scenes		v
General	0x0006	On/Off		v
General	0x0008	Level Control		v
Closure	0x0101	Door Lock		v
Closure	0x0102	Window Covering		v
HVAC	0x0200	Pump Configuration and Control		v
Lighting	0x0300	Color Control		v
Security/Safety	0x0500	IAS Zones		v
Security/Safety	0x0501	IAS ACE	v	
Security/Safety	0x0502	IAS WD		v

Table 2: Home Automation Clusters

5.1.3. Obligatory part type

```
typedef struct _ZbProZclLocalPrimitiveObligatoryPart_t
{
    /* Structured data, aligned at 32 bits. */

    ZBPRO_APS_Address_t             remoteApsAddress;
            /*!< Addressing mode and Address of the remote node. For
the case of outgoing commands generated by this
                node, this field specifies destination address of the
recipient node or group. For the case of incoming
                commands received by this node, this field specifies
source address of the command originator; it may be
                then used as the destination address for the response
command. */

    ZBPRO_APS_Address_t             localApsAddress;
            /*!< Addressing mode and Address of the local node. This
field is used only for the case of incoming frames.
                For the case of incoming commands received by this
node, this field specifies the original destination
                addressing mode and address (or group identifier) which
this command was sent to; it may be used by the
                local node to distinguish unicast and nonunicast
incoming commands. This field is not used for the case
                of outgoing commands; in this case APS layer
automatically specifies the local address of this
                (originating) node. */

    /* 16-bit data. */

    ZBPRO_ZCL_ClusterId_t           clusterId;
            /*!< ZCL Cluster identifier. This field must be assigned
with the identifier of the cluster to which this
```

```
                command is to be applied. Cluster-specific commands may
be applied only to their parent cluster, so this
                parameter may be omitted for them when called by the
higher layer (it will be assigned automatically);
                while Profile-wide commands may be applied to arbitrary
ZCL-based cluster and in this case this
                parameter must be specified. */

    ZBPRO_ZCL_ManufCode_t           manufCode;
          /*!< Manufacturer Code for manufacturer specific frames.
This parameter is valid only if \c manufSpecific is
                assigned with TRUE; otherwise it's ignored. */

    ZBPRO_ZCL_Timeout_t             respWaitTimeout;
          /*!< Timeout of waiting for response, in seconds. This
field is used only in ZCL Local Request and Response
                primitives. It instructs ZCL layer how long it shall
wait for specific or default response on outgoing
                command commenced with this Request or Response, in
seconds. Value 0xFFFF instructs ZCL Dispatcher to
                use default ZCL Timeout (note that such default value
may be different for different commands). For
                particular cases ZCL Dispatcher may override user
assignments with preprogrammed values. Value 0x0000 is
                not used and substituted with 0xFFFF if specified
externally in parameters of new Local Request (notice
                that internal usage of this value by ZCL Dispatcher is
different: it stands for instruction not to wait
                for response; this instruction may not be given to ZCL
Dispatcher externally with this parameter but it
                is generated internally by ZCL Dispatcher to itself in
particular cases ). */

    /* 8-bit data. */

    ZBPRO_APS_EndpointId_t          remoteEndpoint;
          /*!< Identifier of Endpoint on the remote node. This field
is equivalent to destination endpoint for
                outgoing commands and to source endpoint for incoming
commands. */

    ZBPRO_APS_EndpointId_t          localEndpoint;
          /*!< Identifier of Endpoint on the local node. This field
is equivalent to source endpoint for outgoing
                commands and to destination endpoint for incoming
commands. For the case when incoming command has
                broadcast DstEndpoint (0xFF) or Indirect (0x0) or
Groupcast (0x1) DstAddrMode, this parameter is
                assigned by APS layer with identifier of concrete local
endpoint to which this frame is indicated now,
                in the range from 0x01 to 0xFE; and in the case of
unicast DstEndpoint (from 0x01 to 0xFE) with Short
```

```
                (0x2) or Extended (0x3) DstAddrMode this parameter is
assigned with value of the DstEndpoint of the
                received frame. */

    ZBPRO_ZCL_CommandId_t           commandId;
            /*!< Command identifier 8-bit value local to particular
cluster, side and manufacturer of the command to be
                issued, in the case of Local Request, or of the command
received, in the cases of Local Confirm and
                Local Indication. When Default Response is to be issued
or is received, this field contains the
                parameter \c CommandId of the Default Response command,
i.e. identifier of the command to which this
                Default Response is related, but not the Default
Response command own identifier (0x0B). */

    ZBPRO_ZCL_TransSeqNum_t         transSeqNum;
            /*!< ZCL layer transaction 8-bit sequence number. This
field is used in all ZCL Local primitives except the
                Request (when request is generated the transaction
sequence number doesn't exist yet). Transaction
                sequence number is reported by local ZCL layer of
command recipient node (mostly ZCL Server) to its
                application in ZCL Local Indication primitive; if
application should respond, it issues ZCL Local
                Response to its ZCL layer and specifies this
transaction sequence number. Transaction sequence number is
                reported also by local ZCL layer of command originator
node to its application in ZCL Local Confirm
                primitive; by this way application (mostly ZCL Client)
is informed with the identifier of transaction
                started by its ZCL layer; it may be used for filtering
incoming responses if they are delivered to this
                client application with individual Local Indications
(for the case of multiple response operations, for
                example); note that response still may arrive prior to
the APS-ACK and due to this reason response may
                be indicated earlier than the corresponding request
transmission is confirmed. */

    ZBPRO_ZCL_Status_t              overallStatus;
            /*!< The overall status of operation execution. This field
is used only in ZCL Local Response and Confirm
                primitives. With this field, in Local Response, the
server node application specifies status of the
                requested operation execution; this status then is
reported (if needed/allowed) by the server node ZCL
                layer via the media to the client node. And in Local
Confirm, the local client ZCL layer reports to its
                application the overall status of the requested
operation. */
```

```
    /* 8-bit data / 1-bit flags. */

    ZBPRO_ZCL_FrameDirection_t      direction;
            /*!< Command direction; either Client-to-Server (0x0) or
Server-to-Client (0x1). For the case of new Local
                Request this field defines direction of the requested
command. But for the case of Local Indication this
                field defines direction opposite to that one of the
received command; it is done to simplify process of
                assigning parameters of Local Response on such
Indication - i.e., this field may be simply copied from
                the Indication parameters to the Response parameters
without need to be inverted by the higher layer. */

    ZBPRO_ZCL_FrameType_t           clusterSpecific;
            /*!< Frame type; either Profile-Wide command (0x0) or
Cluster Specific command (0x1). */

    ZBPRO_ZCL_FrameDomain_t         manufSpecific;
            /*!< Frame domain; either ZCL Standard (0x0) or
Manufacturer Specific (0x1). */

    Bool8_t                         useSpecifiedTsn;
            /*!< TRUE instigates ZCL layer to use the Transaction
Sequence Number (TSN) specified with \c transSeqNum
                parameter instead of automatically generated one (and
avoid generating of it). This flag is set to TRUE
                in parameters of ZCL Response, to instruct ZCL
Dispatcher to use TSN of the received command (that was
                reported with ZCL Indication). For the case of ZCL
Indication and ZCL Confirm this parameter is
                automatically assigned to TRUE by the stack. FALSE
instigates ZCL Dispatcher to automatically generate
                new TSN; it is the case of new ZCL Request. */

    ZBPRO_ZCL_DisableDefaultResp_t  disableDefaultResp;
            /*!< Disable Default Response field; either Default
Response is Disabled (0x1) for successful status or
                Enabled (0x0). This field is used only for outgoing
commands. By default it's set to Enabled (0x0). */

    ZBPRO_ZCL_ResponseType_t        useDefaultResponse;
            /*!< For outgoing commands: TRUE if the Default Response
command must be issued instead of Specific
                Response. For incoming commands: TRUE if Default
Response was received. In both cases \c CommandId holds
                not the Default Response command identifier (0x0B) but
the value of CommandId field of such Default
                Response (i.e., the identifier of a command on which
this Default Response is issued). */

    Bool8_t                         indNonUnicastRequest;
```

```
            /*!< TRUE if received command (produced this Local
Indication or Local Confirm) was sent to nonunicast
               destination address or broadcast destination endpoint.
Destination address is considered to be unicast
               if DstAddrMode is Short (0x2) or Extended (0x3), and
DstAddress is one of unicast addresses (i.e., under
               0xFFF8 for the short addressing and under
0xFFFFFFFFFFFFFFFB for extended addressing). For all other
               cases (including groupcast mode) destination address is
considered to be nonunicast. Note that there are
               no means to distinguish actually nonunicast ZCL
transmission performed as series of distinct unicast APS
               transmissions (each with its own different APS S/N) via
multiple-destination binding, for example.
               Broadcast destination endpoint has identifier 0xFF.
This field is assigned for Local Indication and
               Confirm parameters according to the received frame
parameters; it's used only for outgoing Default
               Responses (i.e., when \c useDefaultResponse equals to
TRUE) to instruct ZCL Dispatcher to abort issuing
               of the Default Response on command if it was sent on
nonunicast destination address. */

    Bool8_t                        indDisableDefaultResp;
            /*!< Stored value of the Disable Default Response field of
the received ZCL command frame; either Default
               Response was Disabled (0x1) for successful response
status or Enabled (0x0). This field is assigned for
               Local Indication and Confirm parameters according to
the received frame parameters; it's used only for
               outgoing Default Responses (i.e., when \c
useDefaultResponse equals to TRUE) to instruct ZCL Dispatcher
               to abort issuing of the Default Response with SUCCESS
status. */

} ZbProZclLocalPrimitiveObligatoryPart_t;
```

5.1.4. Profile wide type

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdConfigureReportingReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */

    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                            payload;
/*!< Reportable change - the minimum change to the attribute that will
result in a report being issued. Variable-length field. */
    /* 16-bit data. */
    ZBPRO_ZCL_AttributeId_t                      attributeID;
/*!< Attribute Identifier - If the direction field is 0x00, this field
contains the identifier of the attribute that is to be reported. If
instead the direction field is 0x01, the device shall expect reports of
values of this attribute.*/
    ZBPRO_ZCL_AttribureReportingInterval_t       minReportingInterval;
/*!< Minimum Reporting Interval - the minimum interval, in seconds,
between issuing reports of the specified attribute. */
    ZBPRO_ZCL_AttribureReportingInterval_t       maxReportingInterval;
/*!< Maximum Reporting Interval - the maximum interval, in seconds,
between issuing reports of the specified attribute. */
    ZBPRO_ZCL_AttribureReportingTimeoutPeriod_t   timeoutPeriod;
/*!< Timeout Period - the maximum expected time, in seconds, between
received reports for the attribute specified in the attribute
identifier field. */
    /* 8-bit data. */
    ZBPRO_ZCL_AttributeReportingDirection_t      directionReporting;
/*!< Direction - The direction field specifies whether values of the
attribute are to be reported, or whether reports of the attribute are
to be received. */
    ZBPRO_ZCL_AttrDataType_t                     attributeDataType;
/*!< Attribute data type - the data type of the attribute that is to be
reported. */
} ZBPRO_ZCL_ProfileWideCmdConfigureReportingReqParams_t;
```

5.1.5. Management Services

5.1.5.1. Functions

5.1.5.1.1. ZBPRO_APS_EndpointRegisterReq()

Brief description:

Function is used to add information about new endpoint to the stack.

Prototype:

```
void zbProApsEndpointRegisterReq(ZBPRO_APS_EndpointRegisterReqDescr_t
*const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_APS_EndpointRegisterReqDescr_t
{
    struct
    {
        SYS_QueueElement_t  qElem;
    } service;
    ZBPRO_APS_EndpointRegisterConfCallback_t    *callback;
#ifndef MAILBOX_HOST_SIDE
    zbProApsEndpointRegisterReqParams_t         params;
#else
    ZBPRO_APS_EndpointRegisterReqParams_t       params;
#endif
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void
ZBPRO_APS_EndpointRegisterConfCallback_t(ZBPRO_APS_EndpointRegisterReqD
escr_t *const reqDescr,
ZBPRO_APS_EndpointRegisterConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _zbProApsEndpointRegisterReqParams_t
{
    ZBPRO_APS_SimpleDescriptor_t simpleDescriptor;
    Bool8_t                      useInternalHandler;
    ZBPRO_APS_DataInd_t         *dataInd;
} zbProApsEndpointRegisterReqParams_t;

typedef struct _ZBPRO_APS_EndpointRegisterReqParams_t
{
    ZBPRO_APS_SimpleDescriptor_t simpleDescriptor;
    Bool8_t                      useInternalHandler;
} ZBPRO_APS_EndpointRegisterReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_EndpointRegisterConfParams_t
{
    ZBPRO_APS_Status_t  status;
} ZBPRO_APS_EndpointRegisterConfParams_t;
```

5.1.5.1.2. ZBPRO_APS_EndpointUnregisterReq()

Brief description:

Function is used to remove information about previously added endpoint from the stack.

Prototype:

```
void
ZBPRO_APS_EndpointUnregisterReq(ZBPRO_APS_EndpointUnregisterReqDescr_t
*const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_APS_EndpointUnregisterReqDescr_t
{
    struct
    {
        SYS_QueueElement_t  qElem;
    } service;

    ZBPRO_APS_EndpointUnregisterConfCallback_t    *callback;
    ZBPRO_APS_EndpointUnregisterReqParams_t       params;
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void
ZBPRO_APS_EndpointUnregisterConfCallback_t(ZBPRO_APS_EndpointUnregister
ReqDescr_t *const reqDescr,
ZBPRO_APS_EndpointRegisterConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_EndpointUnregisterReqParams_t
{
    ZBPRO_APS_EndpointId_t      endpoint;
} ZBPRO_APS_EndpointUnregisterReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_EndpointRegisterConfParams_t
{
    ZBPRO_APS_Status_t  status;
} ZBPRO_APS_EndpointRegisterConfParams_t;
```

5.1.6. Foundation commands

5.1.6.1. Functions

5.1.6.1.1. ZBPRO_ZCL_ProfileWideCmdReadAttributesReq()

Brief description:

Sends read attribute request (Refer to [4] section 2.4.1).

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdReadAttributesReq(
ZBPRO_ZCL_ProfileWideCmdReadAttrReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ProfileWideCmdReadAttrReqDescr_t
```

```
{
    /* 32-bit data. */

    ZBPRO_ZCL_ProfileWideCmdReadAttrConfCallback_t *callback;
/*!< ZCL Confirmation callback handler entry

point. */
    /* Structured data, aligned at 32 bits. */

    ZbProZclLocalPrimitiveDescrService_t            service;
/*!< ZCL Request Descriptor service field. */

    ZBPRO_ZCL_ProfileWideCmdReadAttrReqParams_t     params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_ProfileWideCmdReadAttrConfCallback_t(
     ZBPRO_ZCL_ProfileWideCmdReadAttrReqDescr_t   *const  reqDescr,
     ZBPRO_ZCL_ProfileWideCmdReadAttrConfParams_t *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdReadAttrReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_AttributeId_t                 attributeId;
/*!< Identifier of the attribute that is to be read. */
} ZBPRO_ZCL_ProfileWideCmdReadAttrReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdReadAttrConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                       payload;
/*!< Current value of the requested attribute. */
    /* 8-bit data. */
    ZBPRO_ZCL_AttrDataType_t                attrDataType;
/*!< Data type of the attribute. */
} ZBPRO_ZCL_ProfileWideCmdReadAttrConfParams_t;
```

5.1.6.1.2. ZBPRO_ZCL_ProfileWideCmdReadAttributesResponseReq()

Brief description:

Sends read attribute request (Refer to [4] section 2.4.1).

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdReadAttributesResponseReq(
 ZBPRO_ZCL_ProfileWideCmdReadAttrResponseReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ProfileWideCmdReadAttrReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_ProfileWideCmdReadAttrConfCallback_t *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t            service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ProfileWideCmdReadAttrReqParams_t     params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_ProfileWideCmdReadAttrConfCallback_t(
      ZBPRO_ZCL_ProfileWideCmdReadAttrReqDescr_t   *const  reqDescr,
      ZBPRO_ZCL_ProfileWideCmdReadAttrConfParams_t *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdReadAttrReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_AttributeId_t attributeId;  /*!< Identifier of the
attribute that is to be read. */
} ZBPRO_ZCL_ProfileWideCmdReadAttrReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdReadAttrConfParams_t
```

```
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                       payload;
/*!< Current value of the requested attribute. */
    /* 8-bit data. */
    ZBPRO_ZCL_AttrDataType_t                attrDataType;
/*!< Data type of the attribute. */
} ZBPRO_ZCL_ProfileWideCmdReadAttrConfParams_t;
```

5.1.6.1.3. ZBPRO_ZCL_ProfileWideCmdReadAttributesInd()

Brief description:

Primitive is used to send an indication to the application level when Read Attribute command is received (Refer to [4] section 2.4.2). Should be implemented on application level

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdReadAttributesInd(
     ZBPRO_ZCL_ProfileWideCmdReadAttrIndParams_t *const  indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdReadAttrIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* Structured / 32-bit data. */
    SYS_DataPointer_t                       payload;
/*!< Serialized array of Attribute Identifier fields. */
} ZBPRO_ZCL_ProfileWideCmdReadAttrIndParams_t;
```

Obligatory part type Brief description:

See **5.1.3. Obligatory part type**

5.1.6.1.4. ZBPRO_ZCL_ProfileWideCmdWriteAttributesReq()

Brief description:

Accepts ZCL Local Request to issue Write Attributes profile-wide command.

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdWriteAttributesReq(
     ZBPRO_ZCL_ProfileWideCmdWriteAttrReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ProfileWideCmdWriteAttrReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_ProfileWideCmdWriteAttrConfCallback_t *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t             service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ProfileWideCmdWriteAttrReqParams_t     params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_ProfileWideCmdWriteAttrConfCallback_t(
     ZBPRO_ZCL_ProfileWideCmdWriteAttrReqDescr_t   *const  reqDescr,
     ZBPRO_ZCL_ProfileWideCmdWriteAttrConfParams_t *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdWriteAttrReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                       payload;
/*!< Actual value of the attribute that is to be

written. */
    /* 16-bit data. */
    ZBPRO_ZCL_AttributeId_t                 attributeId;
/*!< Identifier of the attribute that is to be written. */
    /* 8-bit data. */
    ZBPRO_ZCL_AttrDataType_t                attrDataType;
/*!< Data type of the attribute. */
} ZBPRO_ZCL_ProfileWideCmdWriteAttrReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdWriteAttrConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Confirm. */
} ZBPRO_ZCL_ProfileWideCmdWriteAttrConfParams_t;
```

5.1.6.1.5. ZBPRO_ZCL_ProfileWideCmdWriteAttributesInd()

Brief description:

Handles ZCL Local Indication on reception of Write Attribute profile-wide command.

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdWriteAttributesInd(
     ZBPRO_ZCL_ProfileWideCmdWriteAttrIndParams_t *const  indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdWriteAttrIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */

    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* Structured / 32-bit data. */
    SYS_DataPointer_t                       payload;
/*!< Serialized array of Write Attribute Records. */
} ZBPRO_ZCL_ProfileWideCmdWriteAttrIndParams_t;
```

Obligatory part type Brief description:

See **5.1.3. Obligatory part type**

5.1.6.1.6. ZBPRO_ZCL_ProfileWideCmdConfigureReportingReq()

Brief description:

Accepts ZCL Local Request to issue Configure Reporting profile-wide command.

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdConfigureReportingReq(
ZBPRO_ZCL_ProfileWideCmdConfigureReportingReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ProfileWideCmdConfigureReportingReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_ProfileWideCmdConfigureReportingConfCallback_t
*callback; /*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t
service;  /*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ProfileWideCmdConfigureReportingReqParams_t
params;   /*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_ProfileWideCmdConfigureReportingConfCallback_t(
ZBPRO_ZCL_ProfileWideCmdConfigureReportingReqDescr_t *const  reqDescr,
ZBPRO_ZCL_ProfileWideCmdConfigureReportingConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdWriteAttrResponseReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    /* Structured / 32-bit data. */
    SYS_DataPointer_t                      payload;
/*!< Serialized array of Write Attribute Status Records. */
} ZBPRO_ZCL_ProfileWideCmdWriteAttrResponseReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdConfigureReportingConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
```

```
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_AttributeId_t                  attributeID; /*!<
Attribute Identifier - If the direction field is 0x00, field contains
the identifier of the attribute that  reported. If instead the
direction field is the device shall expect reports of values of this
attribute.*/
    /* 8-bit data. */
    ZBPRO_ZCL_Status_t                       statusReporting;      /*!<
Status field - specifies the status of the configure reporting
operation attempted on this attribute. */
    ZBPRO_ZCL_AttributeReportingDirection_t  directionReporting;   /*!<
Direction field specifies whether values of the attribute are reported
(0x00), or whether reports of the attribute are received (0x01). */
} ZBPRO_ZCL_ProfileWideCmdConfigureReportingConfParams_t;
```

5.1.6.1.7. ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReq()

Brief description:

Parses ZCL Frame Payload of profile-wide command Configure Reporting Response.

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReq(
    ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReqDescr_t *const
reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationConfCallback_t
*callback;  /*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t
service;   /*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReqParams_t
params;    /*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void
ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationConfCallback_t(
    ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReqDescr_t
*const  reqDescr,
    ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationConfParams_t
*const  confParams);
```

Parameters type:

```
typedef struct
_ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_AttributeId_t                   attributeID;    /*!<
Attribute identifier field shall contain the identifier of the
attribute whose reporting configuration details are to be read. */
    /* 8-bit data/ */
    ZBPRO_ZCL_AttributeReportingDirection_t   directionReporting;
/*!< Direction field specifies whether values of the attribute are
reported (0x00), or whether reports of the attribute are received
(0x01). */
} ZBPRO_ZCL_ProfileWideCmdReadReportingConfigurationReqParams_t;
```

Profile wide type Brief description:

*See **5.1.4. Profile wide type***

5.1.6.1.8. *ZBPRO_ZCL_ProfileWideCmdDiscoverAttributesResponseReq()*

Brief description:

Accepts ZCL Local Request to issue Discover Attributes Response profile-wide command.

Prototype:

```
void ZBPRO_ZCL_ProfileWideCmdDiscoverAttributesResponseReq(
     ZBPRO_ZCL_ProfileWideCmdDiscoverAttrRespDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ProfileWideCmdDiscoverAttrRespReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_ProfileWideCmdDiscoverAttrRespConfCallback_t   *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t        service;   /*!< ZCL
Request Descriptor service field. */
```

```
    ZBPRO_ZCL_ProfileWideCmdDiscoverAttrRespReqParams_t         params;
/*!< ZCL Request parameters structure. */
};
```

Parameters type:

```
typedef struct _ZBPRO_ZCL_ProfileWideCmdDiscoverAttrRespReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    Bool8_t isDiscoveryComplete; /*!< Is Discovery Complete value. */
    SYS_DataPointer_t payload;/*!< The discovered attributes values. */
} ZBPRO_ZCL_ProfileWideCmdDiscoverAttrRespReqParams_t;
```

5.1.7. Basic Cluster Server

There are no supported commands for this cluster.

5.1.8. On/Off Cluster Client

5.1.8.1. Functions

5.1.8.1.1. ZBPRO_ZCL_OnOffCmdOffReq()

Brief description:

Accepts ZCL Local Request to issue Off command.

Prototype:

```
void RF4CE_MSO_GetRIBAttributeReq(RF4CE_MSO_GetRIBAttributeReqDescr_t
*request);
```

Where `request` is a pointer to the request descriptor structure.

Descriptor:

```
struct _RF4CE_MSO_GetRIBAttributeReqDescr_t
{
    RF4CE_NWK_RequestService_t service;         /*!< Service field. */
    RF4CE_MSO_GetRIBAttributeReqParams_t params;/*!< Request
parameters. */
    RF4CE_MSO_GetRIBAttributeCallback_t callback; /*!< Request
callback. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void (*RF4CE_MSO_GetRIBAttributeCallback_t)
(RF4CE_MSO_GetRIBAttributeReqDescr_t *req,
RF4CE_MSO_GetRIBAttributeConfParams_t *conf);
```

Callback Parameters type:

```
typedef struct _RF4CE_MSO_GetRIBAttributeConfParams_t
{
    uint8_t status;            /*!< Requested attribute status: one of
the RF4CE_MSO_RIBAttributeStatus_t codes */
    uint8_t attributeId;       /*!< Requested attribute ID. */
    uint8_t attributeIndex;    /*!< Requested attribute Index. */
    SYS_DataPointer_t payload; /*!< The requested attribute value. */
} RF4CE_MSO_GetRIBAttributeConfParams_t;
```

5.1.8.1.2. ZBPRO_ZCL_OnOffCmdOnReq()

Brief description:

Accepts ZCL Local Request to issue On command.

Prototype:

```
void ZBPRO_ZCL_OnOffCmdOnReq(
                ZBPRO_ZCL_OnOffCmdReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_OnOffCmdReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_OnOffCmdConfCallback_t     *callback;         /*!< ZCL
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t  service;          /*!< ZCL
Request Descriptor service field. */
    ZBPRO_ZCL_OnOffCmdReqParams_t         params;           /*!< ZCL
Request parameters structure. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void ZBPRO_ZCL_OnOffCmdConfCallback_t(
                ZBPRO_ZCL_OnOffCmdReqDescr_t   *const  reqDescr,
                ZBPRO_ZCL_OnOffCmdConfParams_t *const  confParams);
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_OnOffCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_OnOffCmdConfParams_t;
```

5.1.8.1.3. ZBPRO_ZCL_OnOffCmdToggleReq()

Brief description:

Accepts ZCL Local Request to issue Toggle command.

Prototype:

```
void ZBPRO_ZCL_OnOffCmdToggleReq(
                ZBPRO_ZCL_OnOffCmdReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_OnOffCmdReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_OnOffCmdConfCallback_t     *callback;         /*!< ZCL
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t  service;          /*!< ZCL
Request Descriptor service field. */
    ZBPRO_ZCL_OnOffCmdReqParams_t         params;           /*!< ZCL
Request parameters structure. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void ZBPRO_ZCL_OnOffCmdConfCallback_t(
                ZBPRO_ZCL_OnOffCmdReqDescr_t   *const  reqDescr,
                ZBPRO_ZCL_OnOffCmdConfParams_t *const  confParams);
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_OnOffCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* No custom parameters. */
} ZBPRO_ZCL_OnOffCmdConfParams_t;
```

5.1.9. Scenes Cluster Client

5.1.9.1. Functions

5.1.9.1.1. ZBPRO_ZCL_ScenesCmdAddSceneReq()

Brief description:

Accepts ZCL Local Request to issue AddScene command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdAddSceneReq(
     ZBPRO_ZCL_ScenesCmdAddSceneReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ScenesCmdAddSceneReqDescr_t
{
    ZBPRO_ZCL_ScenesCmdAddSceneConfCallback_t    *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    ZbProZclLocalPrimitiveDescrService_t          service;       /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ScenesCmdAddSceneReqParams_t        params;        /*!<
ZCL Request parameters structure. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void ZBPRO_ZCL_ScenesCmdAddSceneConfCallback_t(
       ZBPRO_ZCL_ScenesCmdAddSceneReqDescr_t   *const  reqDescr,
       ZBPRO_ZCL_ScenesCmdAddSceneConfParams_t *const  confParams);
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdAddSceneResponseIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
    uint8_t  sceneId;
/*!< The Scene ID. */
} ZBPRO_ZCL_ScenesCmdAddSceneResponseIndParams_t;
```

5.1.9.1.2. ZBPRO_ZCL_ScenesCmdViewSceneReq()

Brief description:

Accepts ZCL Local Request to issue AddScene command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdViewSceneReq(
     ZBPRO_ZCL_ScenesCmdViewSceneReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ScenesCmdViewSceneReqDescr_t
{
    ZBPRO_ZCL_ScenesCmdViewSceneConfCallback_t   *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    ZbProZclLocalPrimitiveDescrService_t          service;       /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ScenesCmdViewSceneReqParams_t       params;        /*!<
ZCL Request parameters structure. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void ZBPRO_ZCL_ScenesCmdViewSceneConfCallback_t(
        ZBPRO_ZCL_ScenesCmdViewSceneReqDescr_t   *const  reqDescr,
        ZBPRO_ZCL_ScenesCmdViewSceneConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
    uint8_t  sceneId;
/*!< The Scene ID. */
} ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdViewSceneResponseIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
    uint8_t  sceneId;
/*!< The Scene ID. */
    uint16_t  transitionTime;
/*!< The Transition Time. Valid only with the status SUCCESS. */
    SYS_DataPointer_t payload;
/*!< The Variable part of the response:
first goes the Scene Name Length byte
followed by the Scene Name character string
followed by the Cluster Specific Extension
field set(s): 2 bytes Cluster ID followed
by 1 byte of data length followed by the data
itself, etc. Valid only with the status SUCCESS. */
} ZBPRO_ZCL_ScenesCmdViewSceneResponseIndParams_t;
```

5.1.9.1.3. *ZBPRO_ZCL_ScenesCmdRemoveSceneReq()*

Brief description:

Accepts ZCL Local Request to issue RemoveScene command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdRemoveSceneReq(
     ZBPRO_ZCL_ScenesCmdRemoveSceneReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ScenesCmdRemoveSceneReqDescr_t
{
    ZBPRO_ZCL_ScenesCmdRemoveSceneConfCallback_t    *callback;
/*!< ZCL Confirmation callback handler entry point. */
    ZbProZclLocalPrimitiveDescrService_t             service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ScenesCmdRemoveSceneReqParams_t        params;
/*!< ZCL Request parameters structure. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void ZBPRO_ZCL_ScenesCmdRemoveSceneConfCallback_t(
        ZBPRO_ZCL_ScenesCmdRemoveSceneReqDescr_t   *const  reqDescr,
        ZBPRO_ZCL_ScenesCmdRemoveSceneConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
    uint8_t  sceneId;
/*!< The Scene ID. */
} ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdAddSceneResponseIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint16_t  groupId;  /*!< The Group ID. */
    uint8_t  sceneId;  /*!< The Scene ID. */
} ZBPRO_ZCL_ScenesCmdAddSceneResponseIndParams_t;
```

5.1.9.1.4. *ZBPRO_ZCL_ScenesCmdRemoveAllScenesReq()*

Brief description:

Accepts ZCL Local Request to issue RemoveAllScenes command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdRemoveAllScenesReq(
     ZBPRO_ZCL_ScenesCmdRemoveAllScenesReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ScenesCmdRemoveAllScenesReqDescr_t
{
    ZBPRO_ZCL_ScenesCmdRemoveAllScenesConfCallback_t    *callback;
/*!< ZCL Confirmation callback handler entry point. */
    ZbProZclLocalPrimitiveDescrService_t                 service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ScenesCmdRemoveAllScenesReqParams_t        params;
/*!< ZCL Request parameters structure. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void ZBPRO_ZCL_ScenesCmdRemoveAllScenesConfCallback_t(
    ZBPRO_ZCL_ScenesCmdRemoveAllScenesReqDescr_t   *const  reqDescr,
    ZBPRO_ZCL_ScenesCmdRemoveAllScenesConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdGroupIdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
} ZBPRO_ZCL_ScenesCmdGroupIdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdRemoveAllScenesResponseIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
} ZBPRO_ZCL_ScenesCmdRemoveAllScenesResponseIndParams_t;
```

5.1.9.1.5. ZBPRO_ZCL_ScenesCmdStoreSceneReq()

Brief description:

Accepts ZCL Local Request to issue StoreScene command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdStoreSceneReq(
     ZBPRO_ZCL_ScenesCmdStoreSceneReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ScenesCmdStoreSceneReqDescr_t
{
    ZBPRO_ZCL_ScenesCmdStoreSceneConfCallback_t  *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    ZbProZclLocalPrimitiveDescrService_t          service;       /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ScenesCmdStoreSceneReqParams_t      params;        /*!<
ZCL Request parameters structure. */
};
```

Service field type Brief description:

See **Figure 19: Request Service Type**

Callback type:

```
typedef void ZBPRO_ZCL_ScenesCmdStoreSceneConfCallback_t(
        ZBPRO_ZCL_ScenesCmdStoreSceneReqDescr_t   *const  reqDescr,
        ZBPRO_ZCL_ScenesCmdStoreSceneConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
    uint8_t  sceneId;
/*!< The Scene ID. */
} ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdAddSceneResponseIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
    uint8_t  sceneId;
/*!< The Scene ID. */
} ZBPRO_ZCL_ScenesCmdAddSceneResponseIndParams_t;
```

5.1.9.1.6. ZBPRO_ZCL_ScenesCmdRecallSceneReq()

Brief description:

Accepts ZCL Local Request to issue RecallScene command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdRecallSceneReq(
     ZBPRO_ZCL_ScenesCmdRecallSceneReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ScenesCmdRecallSceneReqDescr_t
{
    ZBPRO_ZCL_ScenesCmdRecallSceneConfCallback_t    *callback;
/*!< ZCL Confirmation callback handler entry point. */
    ZbProZclLocalPrimitiveDescrService_t             service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ScenesCmdRecallSceneReqParams_t        params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_ScenesCmdRecallSceneConfCallback_t(
        ZBPRO_ZCL_ScenesCmdRecallSceneReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_ScenesCmdConfParams_t          *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
    uint8_t  sceneId;
/*!< The Scene ID. */
} ZBPRO_ZCL_ScenesCmdGroupIdSceneIdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Confirm. */
} ZBPRO_ZCL_ScenesCmdConfParams_t;
```

5.1.9.1.7. ZBPRO_ZCL_ScenesCmdGetSceneMembershipReq()

Brief description:

Accepts ZCL Local Request to issue GetSceneMembership command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdGetSceneMembershipReq(
     ZBPRO_ZCL_ScenesCmdGetSceneMembershipReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_ScenesCmdGetSceneMembershipReqDescr_t
{
    ZBPRO_ZCL_ScenesCmdGetSceneMembershipConfCallback_t    *callback;
/*!< ZCL Confirmation callback handler entry point. */
    ZbProZclLocalPrimitiveDescrService_t                   service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_ScenesCmdGetSceneMembershipReqParams_t       params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_ScenesCmdGetSceneMembershipConfCallback_t(
    ZBPRO_ZCL_ScenesCmdGetSceneMembershipReqDescr_t *const reqDescr,
    ZBPRO_ZCL_ScenesCmdConfParams_t                 *const confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdGroupIdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
} ZBPRO_ZCL_ScenesCmdGroupIdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Confirm. */
} ZBPRO_ZCL_ScenesCmdConfParams_t;
```

5.1.9.1.8. ZBPRO_ZCL_ScenesCmdGetSceneMembershipResponseInd()

Brief description:

Accepts ZCL Local Request to issue GetSceneMembership command specific to Scenes ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_ScenesCmdGetSceneMembershipResponseInd(
ZBPRO_ZCL_ScenesCmdGetSceneMembershipResponseIndParams_t *const
indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Indication parameters type:

```
typedef struct
_ZBPRO_ZCL_ScenesCmdGetSceneMembershipResponseIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint8_t  capacity;
/*!< The Capacity. */
    uint16_t  groupId;
/*!< The Group ID. */
    SYS_DataPointer_t payload;
/*!< The Variable part of the response contains the array of the scene
IDs. Valid only with the status SUCCESS. */
} ZBPRO_ZCL_ScenesCmdGetSceneMembershipResponseIndParams_t;
```

5.1.10. Identify Cluster Client

5.1.10.1. Functions

5.1.10.1.1. ZBPRO_ZCL_IdentifyCmdIdentifyReq()

Brief description:

Accepts ZCL Local Request to issue Identify command.

Prototype:

```
void ZBPRO_ZCL_IdentifyCmdIdentifyReq(
     ZBPRO_ZCL_IdentifyCmdIdentifyReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Function dqescription:

```
struct _ZBPRO_ZCL_IdentifyCmdIdentifyReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_IdentifyCmdIdentifyConfCallback_t *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;       /*!<
ZCL Request Descriptor service field. */
```

```
    ZBPRO_ZCL_IdentifyCmdIdentifyReqParams_t     params;        /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_IdentifyCmdIdentifyConfCallback_t(
        ZBPRO_ZCL_IdentifyCmdIdentifyReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_IdentifyCmdConfParams_t       *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdGroupIdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint16_t  groupId;
/*!< The Group ID. */
} ZBPRO_ZCL_ScenesCmdGroupIdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_ScenesCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Confirm. */
} ZBPRO_ZCL_ScenesCmdConfParams_t;
```

5.1.10.1.2. *ZBPRO_ZCL_IdentifyCmdIdentifyQueryReq()*

Brief description:

Accepts ZCL Local Request to issue Identify Query command.

Prototype:

```
void ZBPRO_ZCL_IdentifyCmdIdentifyQueryReq(
     ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Function dqescription:

```
struct _ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_IdentifyCmdIdentifyQueryConfCallback_t *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t              service;
/*!< ZCL Request Descriptor service field. */
```

```
    ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqParams_t     params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_IdentifyCmdIdentifyQueryConfCallback_t(
        ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_IdentifyCmdConfParams_t         *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_IdentifyCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_IdentifyCmdConfParams_t;
```

5.1.10.1.3. *ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseIndEB()*

Brief description:

Indicates parameters of received Identify Query Response command. Should be defined on the application level.

Prototype:

```
void ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseIndEB(
ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseIndParams_t *const
indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Indication parameters type:

```
typedef struct
_ZBPRO_ZCL_ScenesCmdGetSceneMembershipResponseIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
```

```
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint8_t  capacity;
/*!< The Capacity. */
    uint16_t  groupId;
/*!< The Group ID. */
    SYS_DataPointer_t payload;
/*!< The Variable part of the response contains the array of the scene
IDs. Valid only with the status SUCCESS. */
} ZBPRO_ZCL_ScenesCmdGetSceneMembershipResponseIndParams_t;
```

5.1.11. Identify Cluster Server

5.1.11.1. Functions

5.1.11.1.1. ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReq()

Brief description:

Sends Identify Query Response command to the destination device.

Prototype:

```
void ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReq(
     ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReqDescr_t *const
reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Function dqescription:

```
struct _ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseConfCallback_t *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t                      service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReqParams_t     params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseConfCallback_t(
 ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReqDescr_t *const  reqDescr,
 ZBPRO_ZCL_IdentifyCmdConfParams_t         *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
```

```
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* Custom parameters. */
    ZBPRO_ZCL_IdentifyParamIdentifyTime_t   timeout;
/*!< Timeout, in seconds. */
} ZBPRO_ZCL_IdentifyCmdIdentifyQueryResponseReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_IdentifyCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public
interface to local application. */
    /* No custom parameters. */
} ZBPRO_ZCL_IdentifyCmdConfParams_t;.
```

5.1.11.1.2. ZBPRO_ZCL_IdentifyCmdIdentifyQueryInd()

Brief description:

Indicates parameters of received Identify command. Should be defined on the application level.

Prototype:

```
void ZBPRO_ZCL_IdentifyCmdIdentifyQueryInd(
     ZBPRO_ZCL_IdentifyCmdIdentifyQueryIndParams_t *const  indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Indication parameters type:

```
typedef struct _ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
                   if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
                   /*!< Set of obligatory parameters of ZCL public
                   interface to local application. */
    /* No custom parameters. */
} ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqParams_t;
```

5.1.11.1.3. ZBPRO_ZCL_IdentifyCmdIdentifyQueryInd()

Brief description:

Indicates parameters of received Identify Query command. Should be defined on the application level.

Prototype:

```
void ZBPRO_ZCL_IdentifyCmdIdentifyQueryInd(
     ZBPRO_ZCL_IdentifyCmdIdentifyQueryIndParams_t *const  indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Indication parameters type:

```
typedef struct _ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
                   if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
                   /*!< Set of obligatory parameters of ZCL public
                   interface to local application. */
    /* No custom parameters. */
} ZBPRO_ZCL_IdentifyCmdIdentifyQueryReqParams_t;
```

5.1.12. Groups Cluster Client

5.1.12.1. Functions

5.1.12.1.1. ZBPRO_ZCL_GroupsCmdAddGroupReq()

Brief description:

Sends Add `Group` command to the destination device.

Prototype:

```
void ZBPRO_ZCL_GroupsCmdAddGroupReq(
ZBPRO_ZCL_GroupsCmdAddGroupReqDescr_t *const  reqDescr)
```

Where `reqDescr` is a pointer to the request descriptor structure.

Function dqescription:

```
struct _ZBPRO_ZCL_GroupsCmdAddGroupReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_GroupsCmdAddGroupConfCallback_t     *callback;       /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t           service;        /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_GroupsCmdAddGroupReqParams_t         params;         /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_GroupsCmdAddGroupConfCallback_t(
    ZBPRO_ZCL_GroupsCmdAddGroupReqDescr_t     *const  reqDescr,
    ZBPRO_ZCL_GroupsCmdAddGroupConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdAddGroupReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
```

```
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                         payload;
/*!< 0-16 characters of the Group Name. This field contains characters
only - without length byte. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapGroupsGroupID                groupID;
/*!< GroupID to add to the Group Table. */
} ZBPRO_ZCL_GroupsCmdAddGroupReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdAddGroupConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapGroupsGroupID              groupID;
/*!< The Group ID field is set to the Group ID field of the received
Add Group command */
    /* 8-bit data. */
    ZBPRO_ZCL_Status_t                      status;
/*!< The Status field is set to SUCCESS, DUPLICATE_EXISTS, or
INSUFFICIENT_SPACE as appropriate. */
} ZBPRO_ZCL_GroupsCmdAddGroupConfParams_t;.
```

5.1.12.1.2. ZBPRO_ZCL_GroupsCmdViewGroupReq()

Brief description:

Sends View Group command to the destination device.

Prototype:

```
void
ZBPRO_ZCL_GroupsCmdViewGroupReq(ZBPRO_ZCL_GroupsCmdViewGroupReqDescr_t
*const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_GroupsCmdViewGroupReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_GroupsCmdViewGroupConfCallback_t     *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t           service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_GroupsCmdViewGroupReqParams_t        params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_GroupsCmdViewGroupConfCallback_t(
    ZBPRO_ZCL_GroupsCmdViewGroupReqDescr_t     *const  reqDescr,
    ZBPRO_ZCL_GroupsCmdViewGroupConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdViewGroupReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapGroupsGroupID                groupID;
/*!< GroupID to view. */
} ZBPRO_ZCL_GroupsCmdViewGroupReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdViewGroupConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                         payload;
/*!< 0-16 characters of the Group Name. This field contains characters
only - without length byte. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapGroupsGroupID                groupID;
/*!< The Group ID field is set to the Group ID field of the received
Add Group command. */
    /* 8-bit data. */
    ZBPRO_ZCL_Status_t                        status;
/*!< The Status field is set to SUCCESS, DUPLICATE_EXISTS, or
INSUFFICIENT_SPACE as appropriate. */
} ZBPRO_ZCL_GroupsCmdViewGroupConfParams_t;
```

5.1.12.1.3. ZBPRO_ZCL_GroupsCmdRemoveGroupReq()

Brief description:

Sends Remove Group command to the destination device.

Prototype:

```
void
ZBPRO_ZCL_GroupsCmdRemoveGroupReq(ZBPRO_ZCL_GroupsCmdRemoveGroupReqDesc
r_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_GroupsCmdRemoveGroupReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_GroupsCmdRemoveGroupConfCallback_t   *callback;   /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t            service;    /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_GroupsCmdRemoveGroupReqParams_t       params;     /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_GroupsCmdRemoveGroupConfCallback_t(
    ZBPRO_ZCL_GroupsCmdRemoveGroupReqDescr_t     *const  reqDescr,
    ZBPRO_ZCL_GroupsCmdRemoveGroupConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdRemoveGroupReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */

    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapGroupsGroupID                groupID;
/*!< GroupID to view. */
} ZBPRO_ZCL_GroupsCmdRemoveGroupReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdRemoveGroupConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapGroupsGroupID                groupID;
/*!< The Group ID field is set to the Group ID field of the received
Add Group command. */
    /* 8-bit data. */
    ZBPRO_ZCL_Status_t                        status;
/*!< The Status field is set to SUCCESS, DUPLICATE_EXISTS, or
INSUFFICIENT_SPACE as appropriate. */
} ZBPRO_ZCL_GroupsCmdRemoveGroupConfParams_t;.
```

5.1.12.1.4. ZBPRO_ZCL_GroupsCmdRemoveAllGroupsReq()

Brief description:

Sends Remove All Groups command to the destination device.

Prototype:

```
void
ZBPRO_ZCL_GroupsCmdRemoveAllGroupsReq(ZBPRO_ZCL_GroupsCmdRemoveAllGroup
sReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_GroupsCmdRemoveAllGroupsReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_GroupsCmdRemoveAllGroupsConfCallback_t   *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t                service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_GroupsCmdRemoveAllGroupsReqParams_t       params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_GroupsCmdRemoveAllGroupsConfCallback_t(
  ZBPRO_ZCL_GroupsCmdRemoveAllGroupsReqDescr_t     *const  reqDescr,
  ZBPRO_ZCL_GroupsCmdRemoveAllGroupsConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdRemoveAllGroupsReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;   /*!<
Set of obligatory parameters of ZCL public interface to local
application. */
    /* There no custom fields.*/
} ZBPRO_ZCL_GroupsCmdRemoveAllGroupsReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdRemoveAllGroupsConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;   /*!<
Set of obligatory parameters of ZCL public interface to local
application. */
    /* There no custom fields.*/
} ZBPRO_ZCL_GroupsCmdRemoveAllGroupsConfParams_t;.
```

5.1.12.1.5. ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyReq()

Brief description:

Accepts ZCL Local Request to issue Add Group If Identifying command specific to Groups ZCL cluster.

Prototype:

```
void void
ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyReq(ZBPRO_ZCL_GroupsCmdAddGroupIfI
dentifyReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyConfCallback_t   *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t                   service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyReqParams_t       params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyConfCallback_t(
  ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyReqDescr_t   *const reqDescr,
  ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyConfParams_t *const confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                         payload;
/*!< 0-16 characters of the Group Name. This field contains characters
only - without length byte. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapGroupsGroupID                groupID;
/*!< The Group ID field is set to the Group ID field of the received
Add Group command. */
} ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;   /*!<
Set of obligatory parameters of ZCL public interface to local
application. */
    /* There no custom fields.*/
} ZBPRO_ZCL_GroupsCmdAddGroupIfIdentifyConfParams_t;.
```

5.1.12.1.6. ZBPRO_ZCL_GroupsCmdGetGroupMembershipResponseInd()

Brief description:

Structure for parameters of ZCL Local Confirmation on request to issue Get Group Membership command specific to Groups ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_GroupsCmdGetGroupMembershipResponseInd(
    ZBPRO_ZCL_GroupsCmdGetGroupMembershipIndParams_t   *const
indParams);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Request parameters type:

```
typedef struct _ZBPRO_ZCL_GroupsCmdGetGroupMembershipIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;   /*!<
Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 32-bit data. */
    SYS_DataPointer_t                         payload;             /*!<
The Group ID list field shall contain the identifiers either of all the
groups in the group table or all the groups from the group list field
of the received get group membership command which are in the group
table. Number of groups contained in this list can be calculated as:
SYS_GetPayloadSize(payload) / sizeof(ZBPRO_ZCL_SapGroupsGroupID) */
    /* 8-bit data */
    uint8_t                                   capacity;            /*!<
The Capacity field shall contain the remaining capacity of the group
table of the device. */
} ZBPRO_ZCL_GroupsCmdGetGroupMembershipIndParams_t;
```

5.1.13. Door Lock Cluster Client

5.1.13.1. Functions

5.1.13.1.1. ZBPRO_ZCL_DoorLockCmdLockReq()

Brief description:

Sends Lock Door command to the destination device.

Prototype:

```
void
ZBPRO_ZCL_DoorLockCmdLockReq(ZBPRO_ZCL_DoorLockCmdLockUnlockReqDescr_t
*const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_DoorLockCmdLockUnlockReqDescr_t
{
    ZBPRO_ZCL_DoorLockCmdLockConfCallback_t    *callback;
    ZbProZclLocalPrimitiveDescrService_t        service;
    ZBPRO_ZCL_DoorLockCmdLockUnlockReqParams_t  params;
};
```

Callback type:

```
typedef void ZBPRO_ZCL_DoorLockCmdLockConfCallback_t(
    ZBPRO_ZCL_DoorLockCmdLockUnlockReqDescr_t   *const reqDescr,
    ZBPRO_ZCL_DoorLockCmdLockUnlockConfParams_t *const confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_DoorLockCmdLockUnlockReqParams_t
{
    ZbProZclLocalPrimitiveObligatoryPart_t zclObligatoryPart;
    SYS_DataPointer_t                      codeString;
} ZBPRO_ZCL_DoorLockCmdLockUnlockReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_DoorLockCmdLockUnlockConfParams_t
{
    ZbProZclLocalPrimitiveObligatoryPart_t zclObligatoryPart;
} ZBPRO_ZCL_DoorLockCmdLockUnlockConfParams_t;
```

5.1.13.1.2. ZBPRO_ZCL_DoorLockCmdUnlockReq()

Brief description:

Sends Unlock Door command to the destination device.

Prototype:

```
void
ZBPRO_ZCL_DoorLockCmdUnlockReq(ZBPRO_ZCL_DoorLockCmdLockUnlockReqDescr_
t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_DoorLockCmdLockUnlockReqDescr_t
{
    ZBPRO_ZCL_DoorLockCmdLockConfCallback_t    *callback;
    ZbProZclLocalPrimitiveDescrService_t        service;
    ZBPRO_ZCL_DoorLockCmdLockUnlockReqParams_t  params;
};
```

Callback type:

```
typedef void ZBPRO_ZCL_DoorLockCmdLockConfCallback_t(
    ZBPRO_ZCL_DoorLockCmdLockUnlockReqDescr_t   *const reqDescr,
    ZBPRO_ZCL_DoorLockCmdLockUnlockConfParams_t *const confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_DoorLockCmdLockUnlockReqParams_t
{
    ZbProZclLocalPrimitiveObligatoryPart_t zclObligatoryPart;
    SYS_DataPointer_t                      codeString;
} ZBPRO_ZCL_DoorLockCmdLockUnlockReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_DoorLockCmdLockUnlockConfParams_t
{
    ZbProZclLocalPrimitiveObligatoryPart_t zclObligatoryPart;
} ZBPRO_ZCL_DoorLockCmdLockUnlockConfParams_t;.
```

5.1.14. Level Control Cluster Client

5.1.14.1. Functions

5.1.14.1.1. ZBPRO_ZCL_LevelControlCmdMoveToLevelReq()

Brief description:

Sends Move To Level command to the destination device.

Prototype:

```
void ZBPRO_ZCL_LevelControlCmdMoveToLevelReq(
     ZBPRO_ZCL_LevelControlCmdMoveToLevelReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_LevelControlCmdMoveToLevelReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_LevelControlCmdMoveToLevelConfCallback_t *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t                service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_LevelControlCmdMoveToLevelReqParams_t     params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_LevelControlCmdMoveToLevelConfCallback_t(
       ZBPRO_ZCL_LevelControlCmdMoveToLevelReqDescr_t *const  reqDescr,
       ZBPRO_ZCL_LevelControlCmdConfParams_t  *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdMoveToLevelReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t       zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    Bool8_t                                      withOnOff;
/*!< When assigned to TRUE, command 'Move To Level (with On/Off)' with
Command Id 0x04 is issued; otherwise command 'Move To Level' with
Command Id 0x00 is issued. */
    ZBPRO_ZCL_LevelControlParamLevel_t           level;
/*!< Level, in units specific to particular

device. */

    ZBPRO_ZCL_LevelControlParamTransitionTime_t  transitionTime;
/*!< Transition time, in 1/10ths of a second. When assigned with 0xFFFF
this parameter has no effect; the OnOffTransitionTime attribute is used
instead of it. */
} ZBPRO_ZCL_LevelControlCmdMoveToLevelReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_LevelControlCmdConfParams_t;.
```

5.1.14.1.2. ZBPRO_ZCL_LevelControlCmdMoveReq()

Brief description:

Sends Move command to the destination device.

Prototype:

```
void ZBPRO_ZCL_LevelControlCmdMoveReq(
     ZBPRO_ZCL_LevelControlCmdMoveReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_LevelControlCmdMoveReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_LevelControlCmdMoveConfCallback_t *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;       /*!<
ZCL Request Descriptor service field. */
```

```
    ZBPRO_ZCL_LevelControlCmdMoveReqParams_t     params;        /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_LevelControlCmdMoveConfCallback_t(
        ZBPRO_ZCL_LevelControlCmdMoveReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_LevelControlCmdConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdMoveReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */

    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    Bool8_t                                 withOnOff;
/*!< When assigned to TRUE, command 'Move (with On/Off)' with Command
Id 0x05 is issued; otherwise command 'Move' with Command Id 0x01 is
issued. */
    ZBPRO_ZCL_LevelControlParamDirection_t  moveMode;
/*!< Move mode, either Up or Down. */
    ZBPRO_ZCL_LevelControlParamRate_t       rate;
/*!< Rate, in units per second. When assigned with 0xFF this parameter
has no effect; DefaultMoveRate attribute is used instead of it. */
} ZBPRO_ZCL_LevelControlCmdMoveReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_LevelControlCmdConfParams_t;
```

5.1.14.1.3. ZBPRO_ZCL_LevelControlCmdStepReq()

Brief description:

Sends Step command to the destination device.

Prototype:

```
void ZBPRO_ZCL_LevelControlCmdStepReq(
     ZBPRO_ZCL_LevelControlCmdStepReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_LevelControlCmdStepReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_LevelControlCmdStepConfCallback_t *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;       /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_LevelControlCmdStepReqParams_t     params;        /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_LevelControlCmdStepConfCallback_t(
          ZBPRO_ZCL_LevelControlCmdStepReqDescr_t *const  reqDescr,
          ZBPRO_ZCL_LevelControlCmdConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdStepReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t       zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    Bool8_t                                      withOnOff;
/*!< When assigned to TRUE, command 'Step (with On/Off)' with Command
Id 0x06 is issued; otherwise command 'Step' with Command Id 0x02 is
issued. */
    ZBPRO_ZCL_LevelControlParamDirection_t       stepMode;
/*!< Step mode, either Up or Down. */
    ZBPRO_ZCL_LevelControlParamLevel_t           stepSize;
/*!< Step size, in units specific to particular device. */
    ZBPRO_ZCL_LevelControlParamTransitionTime_t  transitionTime;
/*!< Transition time, in 1/10ths of a second. When assigned with 0xFFFF
this parameter has no effect; device should move as fast as it is able.
*/
} ZBPRO_ZCL_LevelControlCmdStepReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_LevelControlCmdConfParams_t;
```

5.1.14.1.4. ZBPRO_ZCL_LevelControlCmdStopReq()

Brief description:

Sends Stop command to the destination device.

Prototype:

```
void ZBPRO_ZCL_LevelControlCmdStopReq(
     ZBPRO_ZCL_LevelControlCmdStopReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_LevelControlCmdStopReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_LevelControlCmdStopConfCallback_t *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;       /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_LevelControlCmdStopReqParams_t     params;        /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_LevelControlCmdStopConfCallback_t(
        ZBPRO_ZCL_LevelControlCmdStopReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_LevelControlCmdConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdStopReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    Bool8_t                                 withOnOff;
/*!< When assigned to TRUE, command 'Stop (with On/Off)' with Command
Id 0x07 is issued; otherwise command 'Stop' with Command Id 0x03 is
issued. */
} ZBPRO_ZCL_LevelControlCmdStopReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_LevelControlCmdConfParams_t;
```

5.1.15. Window Covering Cluster Client

5.1.15.1. Functions

5.1.15.1.1. ZBPRO_ZCL_WindowCoveringCmdUpOpenReq()

Brief description:

Accepts ZCL Local Request to issue Up/Open command specific to Window Covering ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_WindowCoveringCmdUpOpenReq(
     ZBPRO_ZCL_WindowCoveringCmdReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_WindowCoveringCmdReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_WindowCoveringCmdConfCallback_t   *callback;  /*!< ZCL
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;   /*!< ZCL
Request Descriptor service field. */
    ZBPRO_ZCL_WindowCoveringCmdReqParams_t         params;  /*!< ZCL
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_WindowCoveringCmdConfCallback_t(
        ZBPRO_ZCL_WindowCoveringCmdReqDescr_t   *const  reqDescr,
        ZBPRO_ZCL_WindowCoveringCmdConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_WindowCoveringCmdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Request. */
} ZBPRO_ZCL_WindowCoveringCmdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_WindowCoveringCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
```

```
   /* No custom parameters exist for this type of ZCL Local Confirm. */
} ZBPRO_ZCL_WindowCoveringCmdConfParams_t;
```

5.1.15.1.2. ZBPRO_ZCL_WindowCoveringCmdDownCloseReq()

Brief description:

Sends Down/Close command to the destination device.

Prototype:

```
void ZBPRO_ZCL_WindowCoveringCmdDownCloseReq(
     ZBPRO_ZCL_WindowCoveringCmdReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_WindowCoveringCmdReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_WindowCoveringCmdConfCallback_t   *callback;  /*!< ZCL
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;   /*!< ZCL
Request Descriptor service field. */
    ZBPRO_ZCL_WindowCoveringCmdReqParams_t         params;  /*!< ZCL
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_WindowCoveringCmdConfCallback_t(
        ZBPRO_ZCL_WindowCoveringCmdReqDescr_t   *const  reqDescr,
        ZBPRO_ZCL_WindowCoveringCmdConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_WindowCoveringCmdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Request. */
} ZBPRO_ZCL_WindowCoveringCmdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_WindowCoveringCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Confirm. */
} ZBPRO_ZCL_WindowCoveringCmdConfParams_t;.
```

5.1.15.1.3. ZBPRO_ZCL_WindowCoveringCmdStopReq()

Brief description:

Accepts ZCL Local Request to issue Stop command specific to Window Covering ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_WindowCoveringCmdStopReq(
     ZBPRO_ZCL_WindowCoveringCmdReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_WindowCoveringCmdReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_WindowCoveringCmdConfCallback_t   *callback;  /*!< ZCL
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;   /*!< ZCL
Request Descriptor service field. */
    ZBPRO_ZCL_WindowCoveringCmdReqParams_t         params;  /*!< ZCL
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_WindowCoveringCmdConfCallback_t(
        ZBPRO_ZCL_WindowCoveringCmdReqDescr_t   *const  reqDescr,
        ZBPRO_ZCL_WindowCoveringCmdConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_WindowCoveringCmdReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Request. */
} ZBPRO_ZCL_WindowCoveringCmdReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_WindowCoveringCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
   /* No custom parameters exist for this type of ZCL Local Confirm. */
} ZBPRO_ZCL_WindowCoveringCmdConfParams_t;
```

5.1.16. Color Control Cluster Client

5.1.16.1. Functions

5.1.16.1.1. ZBPRO_ZCL_LevelControlCmdMoveToLevelReq()

Brief description:

Accepts ZCL Local Request to issue Move To Level or 'Move To Level (with On/Off)' command.

Prototype:

```
void ZBPRO_ZCL_LevelControlCmdMoveToLevelReq(
     ZBPRO_ZCL_LevelControlCmdMoveToLevelReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_LevelControlCmdMoveToLevelReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_LevelControlCmdMoveToLevelConfCallback_t *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t                service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_LevelControlCmdMoveToLevelReqParams_t     params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_LevelControlCmdMoveToLevelConfCallback_t(
      ZBPRO_ZCL_LevelControlCmdMoveToLevelReqDescr_t *const  reqDescr,
      ZBPRO_ZCL_LevelControlCmdConfParams_t       *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdMoveToLevelReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t       zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    Bool8_t                                      withOnOff;
/*!< When assigned to TRUE, command 'Move To Level (with On/Off)' with
Command Id 0x04 is issued; otherwise command 'Move To Level' with
Command Id 0x00 is issued. */
    ZBPRO_ZCL_LevelControlParamLevel_t           level;
/*!< Level, in units specific to particular device. */
    ZBPRO_ZCL_LevelControlParamTransitionTime_t  transitionTime;
/*!< Transition time, in 1/10ths of a second. When assigned with 0xFFFF
this parameter has no effect; the OnOffTransitionTime attribute is used
instead of it. */
} ZBPRO_ZCL_LevelControlCmdMoveToLevelReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_LevelControlCmdConfParams_t;
```

5.1.16.1.2. ZBPRO_ZCL_LevelControlCmdMoveReq()

Brief description:

Sends Move command to the destination device.

Prototype:

```
void ZBPRO_ZCL_LevelControlCmdMoveReq(
     ZBPRO_ZCL_LevelControlCmdMoveReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_LevelControlCmdMoveReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_LevelControlCmdMoveConfCallback_t *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;       /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_LevelControlCmdMoveReqParams_t     params;        /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_LevelControlCmdMoveConfCallback_t(
        ZBPRO_ZCL_LevelControlCmdMoveReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_LevelControlCmdConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdMoveReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    Bool8_t                                 withOnOff;
/*!< When assigned to TRUE, command 'Move (with On/Off)' with Command
Id 0x05 is issued; otherwise command 'Move' with Command Id 0x01 is
issued. */
    ZBPRO_ZCL_LevelControlParamDirection_t  moveMode;
/*!< Move mode, either Up or Down. */
    ZBPRO_ZCL_LevelControlParamRate_t       rate;
/*!< Rate, in units per second. When assigned with 0xFF this parameter
has no effect; DefaultMoveRate attribute is used instead of it. */
} ZBPRO_ZCL_LevelControlCmdMoveReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_LevelControlCmdConfParams_t;
```

5.1.16.1.3. *ZBPRO_ZCL_LevelControlCmdStepReq()*

Brief description:

Sends Step command to the destination device.

Prototype:

```
void ZBPRO_ZCL_LevelControlCmdStepReq(
     ZBPRO_ZCL_LevelControlCmdStepReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_LevelControlCmdStepReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_LevelControlCmdStepConfCallback_t *callback;      /*!<
ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t         service;       /*!<
ZCL Request Descriptor service field. */
    ZBPRO_ZCL_LevelControlCmdStepReqParams_t     params;        /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_LevelControlCmdStepConfCallback_t(
        ZBPRO_ZCL_LevelControlCmdStepReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_LevelControlCmdConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdStepReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t       zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    Bool8_t                                      withOnOff;
/*!< When assigned to TRUE, command 'Step (with On/Off)' with Command
Id 0x06 is issued; otherwise command 'Step' with Command Id 0x02 is
issued. */
    ZBPRO_ZCL_LevelControlParamDirection_t       stepMode;
/*!< Step mode, either Up or Down. */
    ZBPRO_ZCL_LevelControlParamLevel_t           stepSize;
/*!< Step size, in units specific to particular device. */
    ZBPRO_ZCL_LevelControlParamTransitionTime_t  transitionTime;
/*!< Transition time, in 1/10ths of a second. When assigned with 0xFFFF
this parameter has no effect; device should move as fast as it is able.
*/
} ZBPRO_ZCL_LevelControlCmdStepReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_LevelControlCmdConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_LevelControlCmdConfParams_t;.
```

5.1.17. Pump Configuration and Control Cluster Client

There are no specific commands for this cluster.

5.1.18. IAS Zone Cluster Client

5.1.18.1. Functions

5.1.18.1.1. ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseReq()

Brief description:

Accepts ZCL Local Request to issue Zone Enroll response command specific to IAS Zone ZCL cluster.

Prototype:

```
void
ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseReq(ZBPRO_ZCL_IASZoneCmdZoneEnrol
lResponseReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseConfCallback_t     *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */

    ZbProZclLocalPrimitiveDescrService_t                      service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseReqParams_t         params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseConfCallback_t(
ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseReqDescr_t *const  reqDescr,
ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 8-bit data. */
    ZBPRO_ZCL_SapIASZoneEnrollResponseCodeType_t   enrollResponseCode;
/*!< Enroll response code. */
    ZBPRO_ZCL_SapIASZoneAttributeZoneID_t          zoneID;
/*!< ZoneID. */
} ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseConfParams_t
{
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* In specific requests there custom parameters must follow the
obligatory ones. */
} ZBPRO_ZCL_IASZoneCmdZoneEnrollResponseConfParams_t;
```

5.1.18.1.2. ZBPRO_ZCL_IASZoneCmdZoneEnrollRequestInd()

Brief description:

Handles ZCL Local Indication on reception of IAS Zone Enroll request command specific to IAS Zone ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_IASZoneCmdZoneEnrollRequestInd(
    ZBPRO_ZCL_IASZoneCmdZoneEnrollRequestIndParams_t   *const
indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_IASZoneCmdZoneEnrollRequestReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapIASZoneAttributeZoneType_t   zoneType;
/*!< Zone Type. */
    uint16_t     manufacturerCode;       /*!< Manufacturer Code. */
} ZBPRO_ZCL_IASZoneCmdZoneEnrollRequestReqParams_t;
```

Return value:

`None.`

5.1.18.1.3. *ZBPRO_ZCL_IASZoneCmdZoneStatusChangeNotificationInd()*

Brief description:

Handles ZCL Local Indication on reception of Zone Status Change Notification command specific to IAS Zone ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_IASZoneCmdZoneStatusChangeNotificationInd(
    ZBPRO_ZCL_IASZoneCmdZoneStatusChangeNotificationIndParams_t
*const   indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Indication parameters type:

```
typedef struct
_ZBPRO_ZCL_IASZoneCmdZoneStatusChangeNotificationReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t       zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapIASZoneAttributeZoneStatus_t     zoneStatus;      /*!<
The Zone Status field shall be the current value of the ZoneStatus
attribute. */
    uint16_t                                      delay;           /*!<
The Delay field is defined as the amount of time in quarter-seconds,
from the moment when a change takes place in one or more bits of the
Zone Status attribute and the successful transmission of the Zone
Status Change Notification. */
    /* 8-bit data. */
    uint8_t                                     extendedStatus;  /*!<
The Extended Status field is reserved for additional status information
and shall be set to zero. */
    ZBPRO_ZCL_SapIASZoneAttributeZoneID_t        zoneID;          /*!<
Zone ID is the index of the Zone in the CIE's zone table. If none is
programmed, the Zone ID default value SHALL be indicated in this field.
*/
} ZBPRO_ZCL_IASZoneCmdZoneStatusChangeNotificationReqParams_t;
```

Return value:

```
None.
```

5.1.19. IAS WD Cluster Client

5.1.19.1. Functions

5.1.19.1.1. ZBPRO_ZCL_IASWDCmdStartWarningReq()

Brief description:

Accepts ZCL Local Request to issue Start warning command command specific to IAS WD ZCL cluster.

Prototype:

```
void
ZBPRO_ZCL_IASWDCmdStartWarningReq(ZBPRO_ZCL_IASWDCmdStartWarningReqDesc
r_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_IASWDCmdStartWarningReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_IASWDCmdStartWarningConfCallback_t     *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t              service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_IASWDCmdStartWarningReqParams_t         params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_IASWDCmdStartWarningConfCallback_t(
  ZBPRO_ZCL_IASWDCmdStartWarningReqDescr_t     *const  reqDescr,
  ZBPRO_ZCL_IASWDCmdStartWarningConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_IASWDCmdStartWarningReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;     /*!<
Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 16-bit data. */
    ZBPRO_ZCL_SapIASWDParamWarningDuration_t  warningDuration;    /*!<
Requested duration of warning, in seconds. If both Strobe and Warning
Mode are "0" this field shall be ignored. */
    /* 8-bit data. */
    ZBPRO_ZCL_SapIASWDParamWarningMode_t      warningMode;        /*!<
4-bit enumeration for Warning Mode. */
    ZBPRO_ZCL_SapIASWDParamWarningStrobe_t    strobe;             /*!<
2-bit enumeration for Strobe. */
    ZBPRO_ZCL_SapIASWDParamSirenLevel_t       sirenLevel;         /*!<
2-bit enumeration for Siren Level. */
    ZBPRO_ZCL_SapIASWDParamStrobeDutyCycle_t  strobeDutyCycle;    /*!<
Indicates the length of the flash cycle. For example, if Strobe Duty
Cycle Field specifies �40,� then the strobe SHALL flash ON for
4/10ths a second and then turn OFF for 6/10ths of a second. */
    ZBPRO_ZCL_SapIASWDParamStrobeLevel_t      strobeLevel;        /*!<
Indicates the intensity of the strobe. */
} ZBPRO_ZCL_IASWDCmdStartWarningReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZbProZclLocalPrimitiveParamsPrototype_t
{
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* In specific requests there custom parameters must follow the
obligatory ones. */
} ZbProZclLocalPrimitiveParamsPrototype_t;
```

5.1.19.1.2. *ZBPRO_ZCL_IASWDCmdSquawkgReq()*

Brief description:

Accepts ZCL Local Request to issue Squawk command
 * command specific to IAS WD ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_IASWDCmdSquawkgReq(ZBPRO_ZCL_IASWDCmdSquawkReqDescr_t
*const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_IASWDCmdSquawkReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZCL_IASWDCmdSquawkConfCallback_t           *callback;
/*!< ZCL Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZclLocalPrimitiveDescrService_t              service;
/*!< ZCL Request Descriptor service field. */
    ZBPRO_ZCL_IASWDCmdSquawkReqParams_t               params;
/*!< ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZCL_IASWDCmdSquawkConfCallback_t(
        ZBPRO_ZCL_IASWDCmdSquawkReqDescr_t     *const  reqDescr,
        ZBPRO_ZCL_IASWDCmdSquawkConfParams_t   *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_IASWDCmdSquawkReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t    zclObligatoryPart;  /*!<
Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    /* 8-bit data. */
    ZBPRO_ZCL_SapIASWDParamSquawkMode_t       squawkMode;         /*!<
4-bit enumeration for Squawk Mode. */
    ZBPRO_ZCL_SapIASWDParamSquawkStrobe_t     strobe;             /*!<
The strobe field. */
    ZBPRO_ZCL_SapIASWDParamSquawkLevel_t      squawkLevel;        /*!<
2-bit enumeration for Squawk Level. */
} ZBPRO_ZCL_IASWDCmdSquawkReqParams_t;
```

Callback Parameters type:

```
ypedef struct _ZbProZclLocalPrimitiveParamsPrototype_t

{

    /* Structured data, aligned at 32 bits. */


    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public


interface to local application. */

wawwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwwww

    /* In specific requests there custom parameters must follow the
obligatory ones. */


} ZbProZclLocalPrimitiveParamsPrototype_t;
```

5.1.20. IAS ACE Cluster Server

5.1.20.1. Functions

5.1.20.1.1. ZBPRO_ZCL_SapIasAceArmRespReq()

Brief description:

IAS ACE Arm Response Command request.

Prototype:

```
void ZBPRO_ZCL_SapIasAceArmRespReq(ZBPRO_ZCL_SapIasAceArmRespReqDescr_t
*const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_SapIasAceArmRespReqDescr_t
{
    ZBPRO_ZCL_SapIasAceArmRespReqConfCallback_t         *callback;
    ZbProZclLocalPrimitiveDescrService_t                service;
    ZBPRO_ZCL_SapIasAceArmRespReqParams_t               params;
};
```

Callback type:

```
typedef void ZBPRO_ZCL_SapIasAceArmRespReqConfCallback_t(
        ZBPRO_ZCL_SapIasAceArmRespReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_SapIasAceRespReqConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceArmRespReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    ZBPRO_ZCL_IasAceArmNotification_t   notification;
} ZBPRO_ZCL_SapIasAceArmRespReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceRespReqConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters */
} ZBPRO_ZCL_SapIasAceRespReqConfParams_t;
```

5.1.20.1.2. ZBPRO_ZCL_SapIasAceArmInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Arm Command specific to IAS ACE ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_SapIasAceArmInd(
            ZBPRO_ZCL_SapIasAceArmIndParams_t *const indParams)
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceArmIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */

    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    ZBPRO_ZCL_IasAceArmMode_t       armMode;
    uint8_t                         zoneId;
    SYS_DataPointer_t               payload;    /*<! Arm/Disarm code as
UTF-8 character string w/o leading length. Refer to HA errata 1855 */
} ZBPRO_ZCL_SapIasAceArmIndParams_t;
```

Return value:

```
None.
```

5.1.20.1.3. ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReq()

Brief description:

IAS ACE Get Zone ID MAP Response Command request.

Prototype:

```
void
ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReq(ZBPRO_ZCL_SapIasAceGetZoneIdMapR
espReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReqDescr_t
{
    ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReqConfCallback_t *callback;
    ZbProZclLocalPrimitiveDescrService_t               service;
    ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReqParams_t      params;
};
```

Callback type:

```
typedef void ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReqConfCallback_t(
        ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReqDescr_t *const  reqDescr,
        ZBPRO_ZCL_SapIasAceRespReqConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters, 
if exist, must follow these fields. */

    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public

interface to local application. */
    /* Custom parameters. */
    ZBPRO_ZCL_IasAceGetZoneIdMap_t          map;
} ZBPRO_ZCL_SapIasAceGetZoneIdMapRespReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceRespReqConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters, 
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local 
application. */
    /* No custom parameters */
} ZBPRO_ZCL_SapIasAceRespReqConfParams_t;
```

5.1.20.1.4. ZBPRO_ZCL_SapIasAceGetZoneIdMapInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Get Zone ID MAP Command to IAS ACE ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_SapIasAceGetZoneIdMapInd(
     ZBPRO_ZCL_SapIasAceGetZoneIdMapIndParams_t *const indParams)
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceGetZoneIdMapIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters, 
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local 
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_SapIasAceGetZoneIdMapIndParams_t;
```

Return value:

```
None.
```

5.1.20.1.5. ZBPRO_ZCL_SapIasAceGetZoneInfoInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Get Zone Information Command specific to IAS ACE ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_SapIasAceGetZoneInfoInd(
            ZBPRO_ZCL_SapIasAceGetZoneInfoIndParams_t *const indParams)
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceGetZoneInfoIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint8_t                         zoneId;
} ZBPRO_ZCL_SapIasAceGetZoneInfoIndParams_t;
```

Return value:

```
None.
```

5.1.20.1.6. ZBPRO_ZCL_SapIasAceGetPanelStatusInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Get Panel Status Command specific to IAS ACE ZCL cluster.

Prototype:

```
void
ZBPRO_ZCL_SapIasAceGetPanelStatusInd(ZBPRO_ZCL_SapIasAceGetPanelStatusI
ndParams_t *const indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceGetPanelStatusIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_SapIasAceGetPanelStatusIndParams_t;
```

Return value:

```
None.
```

5.1.20.1.7. ZBPRO_ZCL_SapIasAceZoneStatusChangedReq()

Brief description:

IAS ACE Zone Status Changed Command request. Sends Get Zone Information Response command to the destination device.

Prototype:

```
void
ZBPRO_ZCL_SapIasAceZoneStatusChangedReq(ZBPRO_ZCL_SapIasAceZoneStatusCh
angedReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_SapIasAceZoneStatusChangedReqDescr_t
{
    ZBPRO_ZCL_SapIasAceZoneStatusChangedReqConfCallback_t *callback;
    ZbProZclLocalPrimitiveDescrService_t                service;
    ZBPRO_ZCL_SapIasAceZoneStatusChangedReqParams_t     params;
};
```

Callback type:

```
typedef void ZBPRO_ZCL_SapIasAceZoneStatusChangedReqConfCallback_t(
      ZBPRO_ZCL_SapIasAceZoneStatusChangedReqDescr_t *const  reqDescr,
      ZBPRO_ZCL_SapIasAceRespReqConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceZoneStatusChangedReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint8_t                                    zoneId;
    uint16_t                                   zoneStatus;
    ZBPRO_ZCL_SapIasAceAudibleNotification_t    audibleNotification;
    SYS_DataPointer_t                          payload;
/*!< as zoneLabel - UTF-8 ZigBee character string w/o leading size byte
*/
} ZBPRO_ZCL_SapIasAceZoneStatusChangedReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceRespReqConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters */
} ZBPRO_ZCL_SapIasAceRespReqConfParams_t;
```

5.1.20.1.8. ZBPRO_ZCL_SapIasAcePanelStatusChangedReq()

Brief description:

Sends Panel Status Changed command to the destination device.

Prototype:

```
void
ZBPRO_ZCL_SapIasAcePanelStatusChangedReq(ZBPRO_ZCL_SapIasAcePanelStatus
ChangedReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_ZCL_SapIasAcePanelStatusChangedReqDescr_t
{
    ZBPRO_ZCL_SapIasAcePanelStatusChangedReqConfCallback_t *callback;
    ZbProZclLocalPrimitiveDescrService_t                service;
    ZBPRO_ZCL_SapIasAcePanelStatusChangedReqParams_t    params;
};
```

Callback type:

```
typedef void ZBPRO_ZCL_SapIasAcePanelStatusChangedReqConfCallback_t(
    ZBPRO_ZCL_SapIasAcePanelStatusChangedReqDescr_t *const  reqDescr,
    ZBPRO_ZCL_SapIasAceRespReqConfParams_t *const  confParams);
```

Request parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAcePanelStatusChangedReqParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    ZBPRO_ZCL_SapIasAcePanelStatus_t            panelStatus;
    uint8_t                                     secondsRemaining;
    ZBPRO_ZCL_SapIasAceAudibleNotification_t    audibleNotification;
    ZBPRO_ZCL_IasAceAlarmStatus_t               alarmStatus;
} ZBPRO_ZCL_SapIasAcePanelStatusChangedReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceRespReqConfParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters */
} ZBPRO_ZCL_SapIasAceRespReqConfParams_t;
```

5.1.20.1.9. ZBPRO_ZCL_SapIasAceEmergencyInd()

Brief description:

Alarm indication for IAS ACE Emergency Command. Indicates parameters of received Emergency command. Should be defined on the application level.

Prototype:

```
void
ZBPRO_ZCL_SapIasAceEmergencyInd(ZBPRO_ZCL_SapIasAceAlarmIndParams_t
*const indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Request parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceAlarmIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_SapIasAceAlarmIndParams_t;
```

5.1.20.1.10. ZBPRO_ZCL_SapIasAceFireInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Fire Command specific to IAS ACE ZCL cluster. Indicates parameters of received Fire command. Should be defined on the application level

Prototype:

```
void ZBPRO_ZCL_SapIasAceFireInd(ZBPRO_ZCL_SapIasAceAlarmIndParams_t
*const indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceAlarmIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_SapIasAceAlarmIndParams_t;
```

Return value:

`None.`

5.1.20.1.11. ZBPRO_ZCL_SapIasAcePanicInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Panic Command specific to IAS ACE ZCL cluster. Indicates parameters of received Panic command. Should be defined on the application level.

Prototype:

```
void ZBPRO_ZCL_SapIasAcePanicInd(ZBPRO_ZCL_SapIasAceAlarmIndParams_t
*const indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceAlarmIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_SapIasAceAlarmIndParams_t;
```

Return value:

```
None.
```

5.1.20.1.12. ZBPRO_ZCL_SapIasAceBypassInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Bypass Command to IAS ACE ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_SapIasAceBypassInd(ZBPRO_ZCL_SapIasAceBypassIndParams_t
*const indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceBypassIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint8_t                         zoneNum;    /*<! Number of Zones */
    SYS_DataPointer_t               payload;    /*<! list of ZoneIds
followed by an Arm/Disarm code as UTF-8 character string w/o leading
length (Refer to HA errata 1855) */
} ZBPRO_ZCL_SapIasAceBypassIndParams_t;
```

Return value:

```
None.
```

5.1.20.1.13. ZBPRO_ZCL_SapIasAceEmergencyInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Emergency Command specific to IAS ACE ZCL cluster.

Prototype:

```
void
ZBPRO_ZCL_SapIasAceEmergencyInd(ZBPRO_ZCL_SapIasAceAlarmIndParams_t
*const indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceAlarmIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_SapIasAceAlarmIndParams_t;
```

Return value:

`None.`

5.1.20.1.14. ZBPRO_ZCL_SapIasAceGetBypassedZoneListInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Get Bypassed Zone List Command specific to IAS ACE ZCL cluster.

Prototype:

```
void ZBPRO_ZCL_SapIasAceGetBypassedZoneListInd(
            ZBPRO_ZCL_SapIasAceGetBypassedZoneListIndParams_t *const
indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceGetBypassedZoneListIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* No custom parameters. */
} ZBPRO_ZCL_SapIasAceGetBypassedZoneListIndParams_t;
```

Return value:

`None.`

5.1.20.1.15. ZBPRO_ZCL_SapIasAceGetZoneStatusInd()

Brief description:

Handles ZCL Local Indication on reception of IAS ACE Get Zone Status Command specific to IAS ACE ZCL cluster.

Prototype:

```
void
ZBPRO_ZCL_SapIasAceGetZoneStatusInd(ZBPRO_ZCL_SapIasAceGetZoneStatusInd
Params_t *const indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZCL_SapIasAceGetZoneStatusIndParams_t
{
    /* Obligatory fields. Do not change the order. Custom parameters,
if exist, must follow these fields. */
    ZbProZclLocalPrimitiveObligatoryPart_t  zclObligatoryPart;
/*!< Set of obligatory parameters of ZCL public interface to local
application. */
    /* Custom parameters. */
    uint8_t                         startingZoneId;
    uint8_t                         zoneIdNumMax;
    Bool8_t                         zoneStatusMaskFlag;
    BitField16_t                    zoneStatusMask;         // TODO Get
this type from the IAS Zone Cluster
} ZBPRO_ZCL_SapIasAceGetZoneStatusIndParams_t;
```

Return value:

```
None.
```

5.1.21. Network Services

5.1.21.1. Functions

5.1.21.1.1. ZBPRO_ZDO_StartNetworkReq()

Prototype:

```
void ZBPRO_ZDO_StartNetworkReq(ZBPRO_ZDO_StartNetworkReqDescr_t *const
reqDescr)
```

Brief description

Makes a device form/join the network. Before this request invocation a set of network parameters shall be set. After cold start it is required to set Extended Address, Channel mask, Security Keys, Device Type, Rx On When Idle flag and Extended PanId (if device type was set to ZBPRO_DEVICE_TYPE_COORDINATOR) using the HA Set Request primitive. When Enter Network request is called the ZigBee PRO stack will perform the following procedures:

1. If the device type was set to ZBPRO_DEVICE_TYPE_COORDINATOR the network formation is initiated.
    a. Energy Detection scan is performed to choose the best channel to work on.
    b. Active scan is performed to detect neighbor networks.

   c. The unique Pan Identifier is generated using the information from the previous step.
   d. All layers are initialized with accordance to the specified parameters.
2. Otherwise the network joining procedure is initiated:
   a. Active scan is performed to detect neighbor networks.
   b. The best network to join is determined using the information from previous step.
   c. The joining procedure is performed.
   d. Authentication procedure is performed.
   e. If the previous step was successful the device is now a part of the HA network. Otherwise a new join attempt (the number of attempts is limited) is performed for the second network found at the step a.

Usage example:

This section explains how the user application can perform typical operations with HA Enter Network request primitive.

1. Memory for the request parameters should be statically allocated somewhere in application:

```
ZBPRO_ZDO_StartNetworkReqDescr_t      zZdoStartNetworkReq;
```

2. Request parameters should be specified as it shown below:

```
void ZBPRO_ZDO_StartNetworkReq(ZBPRO_ZDO_StartNetworkReqDescr_t *const
                                   reqDescr);
```

3. The confirmation callback shall be defined somewhere within the application:

```
static void zbProZhaEzModeZdoStartConf(
       ZBPRO_ZDO_StartNetworkReqDescr_t *const reqDescr,
       ZBPRO_ZDO_StartNetworkConfParams_t *const confParams
{
   /* The conf structure contains obtained network parameters if the
      status field is equal to HA_SUCCESS_STATUS. */
}
```

4. After the parameters are set the request primitive can be called:

```
ZBPRO_ZDO_StartNetworkReq(&zhaEzMode->zdoStart);
```

5. After the haEnterNetworkConfirmCb() function is called and the returned status is equal to HA_SUCCESS_STATUS the procedure has been executed successfully and the device can start sending data.

5.1.21.1.2. ZBPRO_ZHA_EzModeReq()

Prototype:

```
void ZBPRO_ZHA_EzModeReq(ZBPRO_ZHA_EzModeReqDescr_t *reqDescr);
```

Brief description:

Implements EZ-Mode Finding and Binding procedure as described in [2] paragraph 8.3.5. This primitive should be used to perform both Target and Initiator procedures.

Usage example:

This section explains how the user application can perform typical operations with HA EZ-Mode Finding and Binding request primitive.

1. Memory for the request parameters should be statically allocated somewhere in application:

```
ZBPRO_ZHA_EzModeReqDescr_t                  zZhaEzModeReq;
```

2. Request parameters should be specified as it shown below:

```
typedef struct _ZBPRO_ZHA_EzModeReqParams_t
{
    uint32_t                roundTimeMs;
    uint8_t                 times;
    ZBPRO_ZHA_EzModeRole_t  ezRole;
    Bool8_t                 factoryFresh;
    ZBPRO_APS_EndpointId_t  endPoint;
}
```

3. The confirmation callback shall be defined somewhere within the application:

```
void (*ZBPRO_ZHA_EzModeCallback_t) (ZBPRO_ZHA_EzModeReqDescr_t *const
reqDescr,ZBPRO_ZHA_EzModeConfParams_t *const confParams);
{
  /* Put here a code to analyze the result status */
}
```

4. After the ZBPRO_ZHA_EzModeReq() function is called a new HA EZ-Mode Finding and Binding request can be issued using the same ZBPRO_ZHA_EzModeReq() structure.

5.1.21.1.3. ZBPRO_NWK_PermitJoiningReq()

Prototype:

```
void ZBPRO_NWK_PermitJoiningReq(ZBPRO_NWK_PermitJoiningReqDescr_t
*reqDescr);
```

Brief description:

Function implements the feature to allow others to join the network. Should be used for both cases to permit locally and to permit joining on remote devices.
If joining is prohibited on the device it will ignore all attempts to join the network performed from the other device (it will ignore Association requests and Network Join requests).
To open whole network a broadcast transmission of Permit Joining request can be used.

Usage example:

This section explains how the user application can perform typical operations with HA Permit Joining request primitive.

1. Memory for the request parameters should be statically allocated somewhere in application:

```
HA_PermitJoiningReq_t haPermitJoiningReq;
```

2. Request parameters should be specified:

```
haPermitJoiningReq.addrInfo.dstAddrInfo = THIS_DEVICE_ADDRESSING_INFO;
haPermitJoiningReq.addrInfo.srcEndpoint = THIS_APPLICATION_ENDPOINT;
haPermitJoiningReq.permitDuration = 0xFF; /* Permit joining until other
                                            value is set by the
                                            application. */
haPermitJoiningReq.tcSignificance = true;
haPermitJoiningReq.HA_PermitJoiningConf = haPermitJoiningConfirmCb;
```

3. The confirmation callback shall be defined somewhere within the application:

```
void haPermitJoiningConfirmCb(
                     HA_PermitJoiningReq_t *origReq,
                     HA_Status_t status)
{
  /* Put here a code to analyze the result status */
}
```

4. After the haPermitJoiningConfirmCb () function is called a new Permit Joining request can be issued using the same haPermitJoiningReq structure.

5.1.21.1.4. ZBPRO_APS_SetReq()

Prototype:

```
void ZBPRO_APS_SetReq(ZBPRO_APS_SetReqDescr_t *const reqDescr);Brief
description:
```

Function provides the User with the ability to set value of attributes related to the network activity. It may be used to set ZDO, APS, NWK and MAC layer attributes.

Usage example:

This section explains how the user application can perform typical operations with HA Set request primitive.

1. Memory for the Set request parameters should be statically allocated somewhere in application, for example:

```
ZBPRO_APS_SetReqDescr_t aspSet; /* The set request to be used by the
                                   application. */
```

2. Set request should be filled in with the appropriate parameters:

```
ZBPRO_APS_SetReq.id = MAC_MAX_CSMA_BACKOFFS;
ZBPRO_APS_SetReq.value = USER_SPECIFIED_MAC_MAX_CSMA_BACKOFFS;
ZBPRO_APS_SetReq.ZBPRO_APS_SetConf = haSetConfirmCb;
```

3. The confirmation callback shall be defined somewhere within the application:

```
ZBPRO_APS_SetConfParams_t {

    uint8_t                        id;
    ZBPRO_APS_Status_t             status;
{
  /* Put here a code to analyze the result status. */
}
```

The HA Set request can be called now, it will set the value of MAC_MAX_CSMA_BACKOFFS attribute to the MAC Information base and after that haSetConfirmCb() function will be called. The attrId and value parameters can be changed and the procedure can be performed again to set another attribute within the BroadBee stack.

5.1.21.1.5. ZBPRO_APS_GetReq()

Prototype:

```
void ZBPRO_APS_GetReq(ZBPRO_APS_GetReqDescr_t *const reqDescr);
```

Brief description:

Function provides the User with the ability to get value of attributes related to the network activity. It may be used to get ZDO, APS, NWK and MAC layer attributes value.

Usage example:

This section explains how the user application can perform typical operations with HA Get request primitive.

1. Memory for the Get request parameters should be statically allocated somewhere in application:

```
ZBPRO_APS_GetReqDescr_t ZBPRO_APS_GetReq = {
    HA_GetConf = haGetConfirmCb, /* The callback function can be
                                    specified only once if it is
                                    supposed to use with every Get
                                    request. */
};
```

2. Get request should be filled in with the appropriate parameters:

```
haGetReq.attrId = MAC_MAX_CSMA_BACKOFFS;
```

3. The confirmation callback shall be defined somewhere within the application:

```
typedef void (*ZBPRO_APS_GetConfCallback_t) (
     ZBPRO_APS_GetReqDescr_t   *const reqDescr,
        ZBPRO_APS_GetConfParams_t *const confParams);
{
   /* Put here a code to analyze the result status. */
   /* The value of requested attribute is placed in conf->value
        structure. */
}
```

4. The HA Get request can be called from the application to get the value of MAC_MAX_CSMA_BACKOFFS attribute from the MAC Information base. After the haSetConfirmCb() function is called a new Get request can be issued using the same haGetReq structure.

5.1.22. ZigBeePro APS sublayer

5.1.22.1. APS Binding

5.1.22.1.1. Binding support

5.1.22.1.1.1. Functions

5.1.22.1.1.1.1. ZBPRO_APS_BindReq

Brief description:

A Coordinator needs to bind with the devices system before it could communicate. This primitive allows the next higher layer to request to bind two devices together,or to bind a device to a group, by creating an entry in its local binding table, if supported.
On receipt of this primitive by a device and if the entry could be created, the device issues the ZBPRO_APS_BindReq.confirm primitive with the Status parameter set to SUCCESS.

Prototype:

```
void ZBPRO_APS_BindReq(ZBPRO_APS_BindUnbindReqDescr_t *reqDescr);
```
Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_APS_BindUnbindReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_APS_BindUnbindConfCallback_t *callback;           /*!<
Confirm callback function. */
    /* Structured / 32-bit data. */
    struct
    {
        SYS_QueueElement_t              queueElement;       /*!< APS
requests service field. */
        Bool8_t                         isBindRequest;      /*!< TRUE
for the case of APSME-BIND.request; FALSE for the
case of APSME-UNBIND.request. */
    } service;                              /*!< Service field. */
    /* Structured data. */
    ZBPRO_APS_BindUnbindReqParams_t params;/*!< Request parameters set.
*/
};
```

Callback type:

```
typedef void
ZBPRO_APS_BindUnbindConfCallback_t(ZBPRO_APS_BindUnbindReqDescr_t
*const reqDescr, ZBPRO_APS_BindUnbindConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_BindUnbindReqParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t     srcAddress;     /*!< The source IEEE
address for the binding entry. */
    /* Structured data, aligned at 32 bits. */
    ZBPRO_APS_Address_t     dstAddr;        /*!< The destination
address mode and address for the binding entry. */
    /* 16-bit data. */
    ZBPRO_APS_ClusterId_t   clusterId;      /*!< The identifier of the
cluster on the source device that is to be (un)bound to/from the
destination device. */
    /* 8-bit data. */
    ZBPRO_APS_EndpointId_t  srcEndpoint;    /*!< The source endpoint
for the binding entry. */
    ZBPRO_APS_EndpointId_t  dstEndpoint;    /*!< The destination
endpoint for the binding entry. */
} ZBPRO_APS_BindUnbindReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_BindUnbindConfParams_t
{
    ZBPRO_APS_Status_t       status;        /*!< The results of the
(un)binding request. */
} ZBPRO_APS_BindUnbindConfParams_t;
```

Return value:

None.

5.1.22.1.1.1.2. ZBPRO_APS_DataReq()

Brief description:

Function implements APSDE-DATA request primitive. Refer to ZigBee Spec r20 paragraph 2.2.4.1.1.

Prototype:

```
void ZBPRO_APS_DataReq(ZBPRO_APS_DataReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_APS_DataReqDescr_t
{
    struct
    {
        SYS_QueueElement_t          next;
        ZBPRO_APS_Timestamp_t       txTime;
        ZbProApsPeer_t              peer;
        uint8_t                     peerCnt;
        ZBPRO_APS_Status_t          overallStatus;
        Bool8_t                     isNhlUnicast;
    } service;

    ZBPRO_APS_DataReqParams_t       params;
    ZBPRO_APS_DataConfCallback_t    *callback;
};
```

Callback type:

```
typedef void ZBPRO_APS_DataConfCallback_t(ZBPRO_APS_DataReqDescr_t
*const reqDescr, ZBPRO_APS_DataConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_DataReqParams_t
{
    ZBPRO_APS_Address_t         dstAddress;
    ZBPRO_APS_ProfileId_t       profileId;
    ZBPRO_APS_ClusterId_t       clusterId;
    ZBPRO_APS_EndpointId_t      dstEndpoint;
    ZBPRO_APS_EndpointId_t      srcEndpoint;
    ZBPRO_APS_TxOptions_t       txOptions;
    ZBPRO_APS_NwkRadius_t       radius;
    SYS_DataPointer_t           payload;
} ZBPRO_APS_DataReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_DataConfParams_t
{
    ZBPRO_APS_Address_t     dstAddress;
    ZBPRO_APS_Timestamp_t   txTime;
    ZBPRO_APS_EndpointId_t  dstEndpoint;
    ZBPRO_APS_EndpointId_t  srcEndpoint;
    ZBPRO_APS_Status_t      status;
    Bool8_t                 isNhlUnicast;   /*<! false, if there was
more than one recipient */
} ZBPRO_APS_DataConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.3. ZBPRO_APS_UnbindReq()

Brief description:

Function implements APSDE-DATA request primitive. Refer to ZigBee Spec r20 paragraph 2.2.4.1.1.

Prototype:

```
void ZBPRO_APS_UnbindReq(ZBPRO_APS_BindUnbindReqDescr_t *reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _ZBPRO_APS_BindUnbindReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_APS_BindUnbindConfCallback_t *callback;           /*!<
Confirm callback function. */
    /* Structured / 32-bit data. */
    struct
    {
        SYS_QueueElement_t              queueElement;       /*!< APS
requests service field. */
        Bool8_t                         isBindRequest;      /*!< TRUE
for the case of APSME-BIND.request; FALSE for the case of APSME-
UNBIND.request. */
    } service;                      /*!< Service field. */
    /* Structured data. */
    ZBPRO_APS_BindUnbindReqParams_t     params;             /*!<
Request parameters set. */
};
```

Callback type:

```
typedef void
ZBPRO_APS_BindUnbindConfCallback_t(ZBPRO_APS_BindUnbindReqDescr_t
*const reqDescr, ZBPRO_APS_BindUnbindConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_BindUnbindReqParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t     srcAddress;     /*!< The source IEEE
address for the binding entry. */
    /* Structured data, aligned at 32 bits. */
    ZBPRO_APS_Address_t     dstAddr;        /*!< The destination
address mode and address for the binding entry. */
    /* 16-bit data. */
    ZBPRO_APS_ClusterId_t   clusterId;      /*!< The identifier of the
cluster on the source device that is to be (un)bound to/from the
destination device. */
    /* 8-bit data. */
    ZBPRO_APS_EndpointId_t  srcEndpoint;    /*!< The source endpoint
for the binding entry. */
    ZBPRO_APS_EndpointId_t  dstEndpoint;    /*!< The destination
endpoint for the binding entry. */
} ZBPRO_APS_BindUnbindReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_BindUnbindConfParams_t
{
    ZBPRO_APS_Status_t       status;        /*!< The results of the
(un)binding request. */
} ZBPRO_APS_BindUnbindConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.4. ZBPRO_APS_GetKeyReq()

Brief description:

APSME-GET(aps key).request primitive function.

Prototype:

```
void ZBPRO_APS_GetKeyReq(ZBPRO_APS_GetKeyReqDescr_t *const reqDescr);;
```
Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_GetKeyReqDescr_t
{
    ZbProApsGetSetServiceField_t    service;
    ZBPRO_APS_GetKeyReqParams_t     params;
    ZBPRO_APS_GetKeyConfCallback_t  callback;
} ZBPRO_APS_GetKeyReqDescr_t;
```

Callback type:

```
typedef void (*ZBPRO_APS_GetKeyConfCallback_t)
(ZBPRO_APS_GetKeyReqDescr_t   *const reqDescr,
ZBPRO_APS_GetKeyConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_GetKeyReqParams_t
{
    ZBPRO_APS_ExtAddr_t             deviceAddr;
} ZBPRO_APS_GetKeyReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_GetKeyConfParams_t
{
    ZbProSspKey_t                   key;
    ZBPRO_APS_Status_t              status;
} ZBPRO_APS_GetKeyConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.5. ZBPRO_APS_AddGroupReq()

Brief description:

APSME-ADD-GROUP.request primitive function.

Prototype:

```
void ZBPRO_APS_AddGroupReq(ZBPRO_APS_AddGroupReqDescr_t *reqDescr);
```
Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_AddGroupReqDescr_t
{
    ZbProApsGroupManagerServiceField_t  service;
    ZBPRO_APS_AddGroupReqParams_t       params;
    ZBPRO_APS_AddGroupConfCallback_t    *callback;
} ZBPRO_APS_AddGroupReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_AddGroupConfCallback_t(ZBPRO_APS_AddGroupReqDescr_t   *const
reqDescr, ZBPRO_APS_AddGroupConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_AddGroupReqParams_t
{
    /* The 16-bit address of the group being added. */
    ZBPRO_APS_GroupId_t groupAddr;
    /* The endpoint to which the given group is being added.
     * Valid range for endpoint is 0x1..0xfe. */
    ZBPRO_APS_EndpointId_t endpoint;
} ZBPRO_APS_AddGroupReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_AddGroupConfParams_t
{
    /* Status of execution. */
    ZBPRO_APS_Status_t status;
    /* The 16-bit address of the group being added. */
    ZBPRO_APS_GroupId_t groupAddr;
    /* The endpoint to which the given group is being added. */
    ZBPRO_APS_EndpointId_t endpoint;
} ZBPRO_APS_AddGroupConfParams_t;
```

Return value:

`None.`

5.1.22.1.1.1.6. ZBPRO_APS_RemoveGroupReq()

Brief description:

APSME-REMOVE-GROUP.request primitive function.

Prototype:

```
void ZBPRO_APS_RemoveGroupReq(ZBPRO_APS_RemoveGroupReqDescr_t
*reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_RemoveGroupReqDescr_t
{
    ZbProApsGroupManagerServiceField_t   service;
    ZBPRO_APS_RemoveGroupReqParams_t     params;
    ZBPRO_APS_RemoveGroupConfCallback_t  *callback;
} ZBPRO_APS_RemoveGroupReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_RemoveGroupConfCallback_t(ZBPRO_APS_RemoveGroupReqDescr_t
*const reqDescr, ZBPRO_APS_RemoveGroupConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_RemoveGroupReqParams_t
{
    /* The 16-bit address of the group being removed. */
    ZBPRO_APS_GroupId_t groupAddr;
    /* The endpoint to which the given group is being removed.
     * Valid range for endpoint is 0x1..0xfe. */
    ZBPRO_APS_EndpointId_t endpoint;
} ZBPRO_APS_RemoveGroupReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_RemoveGroupConfParams_t
{
    /* Status of execution. */
    ZBPRO_APS_Status_t status;
    /* The 16-bit address of the group being removed. */
    ZBPRO_APS_GroupId_t groupAddr;
    /* The endpoint which is to be removed from the group. */
    ZBPRO_APS_EndpointId_t endpoint;
} ZBPRO_APS_RemoveGroupConfParams_t;
```

Return value:

None.

5.1.22.1.1.1.7. ZBPRO_APS_SetKeyReq()

Brief description:

APSME-GET(aps key).request primitive function.

Prototype:

```
void ZBPRO_APS_SetKeyReq(ZBPRO_APS_SetKeyReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_SetKeyReqDescr_t
{
    ZbProApsGetSetServiceField_t    service;
    ZBPRO_APS_SetKeyReqParams_t     params;
    ZBPRO_APS_SetKeyConfCallback_t  callback;
} ZBPRO_APS_SetKeyReqDescr_t;
```

Callback type:

```
typedef void (*ZBPRO_APS_SetKeyConfCallback_t)
(ZBPRO_APS_SetKeyReqDescr_t   *const reqDescr,
ZBPRO_APS_SetKeyConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_SetKeyReqParams_t
{
    ZbProSspKey_t                   newKeyValue;
    ZBPRO_APS_ExtAddr_t             deviceAddr;
} ZBPRO_APS_SetKeyReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_SetKeyConfParams_t
{
    ZBPRO_APS_Status_t              status;
} ZBPRO_APS_SetKeyConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.8. ZBPRO_APS_RemoveAllGroupsReq()

Brief description:

APSME-REMOVE-ALL-GROUPS.request primitive function.

Prototype:

```
void ZBPRO_APS_RemoveAllGroupsReq(ZBPRO_APS_RemoveAllGroupsReqDescr_t
*reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_RemoveAllGroupsReqDescr_t
{
    ZbProApsGroupManagerServiceField_t       service;
    ZBPRO_APS_RemoveAllGroupsReqParams_t     params;
    ZBPRO_APS_RemoveAllGroupsConfCallback_t  *callback;
} ZBPRO_APS_RemoveAllGroupsReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_RemoveAllGroupsConfCallback_t(ZBPRO_APS_RemoveAllGroupsReqDes
cr_t *const reqDescr, ZBPRO_APS_RemoveAllGroupsConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_RemoveAllGroupsReqParams_t
{
    /* The endpoint to which all groups are being removed.
     * Valid range for endpoint is 0x1..0xfe. */
    ZBPRO_APS_EndpointId_t endpoint;
} ZBPRO_APS_RemoveAllGroupsReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_RemoveAllGroupsConfParams_t
{
    /* Status of execution. */
    ZBPRO_APS_Status_t status;
    /* The endpoint which is to be removed from all groups. */
    ZBPRO_APS_EndpointId_t endpoint;
} ZBPRO_APS_RemoveAllGroupsConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.9. ZBPRO_APS_DataInd()

Brief description:

APSME-REMOVE-ALL-GROUPS.request primitive function.

Prototype:

```
void ZBPRO_APS_DataInd_t(ZBPRO_APS_DataIndParams_t *const indParams);
```
Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_APS_DataIndParams_t
{
    ZBPRO_APS_Address_t     dstAddress;
    ZBPRO_APS_Address_t     srcAddress;
    ZBPRO_APS_ProfileId_t   profileId;
    ZBPRO_APS_ClusterId_t   clusterId;
    ZBPRO_APS_EndpointId_t  dstEndpoint;
    ZBPRO_APS_EndpointId_t  srcEndpoint;
    SYS_DataPointer_t       payload;
    ZBPRO_APS_Timestamp_t   rxTime;
    ZBPRO_APS_Status_t      status;
    ZBPRO_APS_Status_t      securityStatus;
    ZBPRO_NWK_Lqi_t         lqi;
    ZBPRO_APS_EndpointId_t  localEndpoint;
} ZBPRO_APS_DataIndParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.10. ZBPRO_APS_TransportKeyReq()

Brief description:

An APSME-TRANSPORT-KEY.request entry point.

Prototype:

```
void ZBPRO_APS_TransportKeyReq(ZBPRO_APS_TransportKeyReqDescr_t *req)
```
Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_TransportKeyReqDescr_t
{
    /* Fields are arranged to minimize paddings */
    ZBPRO_APS_TransportKeyConfCallback_t    *callback;      /*!<
Confirm callback function.  */
    ZBPRO_APS_TransportKeyReqParams_t       params;         /*!<
Request parameters set.     */
    struct
    {
        union
        {
            SYS_QueueElement_t          queueElement;
            ZBPRO_APS_TunnelReqDescr_t      tunnelReq;
        };
    } service;
} ZBPRO_APS_TransportKeyReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_TransportKeyConfCallback_t(ZBPRO_APS_TransportKeyReqDescr_t
*const reqDescr, ZBPRO_APS_SecurityServicesConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_TransportKeyReqParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t     destAddress;     /*!< The extended 64-bit
address of the destination device. */
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t     parentPartnerAddress;  /*!< The extended
64-bit address of the parent of the destination device given by the
DestAddress parameter. */
    /* 16x8-bit data. */
    ZbProSspKey_t           key;            /*!< The key to be
transported. */
    /* 8-bit data. */
    ZbProSspNwkKeySeqNum_t  keySeqNumber;   /*!< A sequence number
assigned to a network key by the Trust Center and used to distinguish
network keys for purposes of key updates and incoming frame security
operations. */
    Bool8_t                 useParent;      /*!< Indicates if the
destination device's parent shall be used to forward the key to the
destination device: TRUE = Use parent; FALSE = Do not use parent. */
    Bool8_t                 initiator;      /*!< Indicates if the
destination device of this key requested it: TRUE = If the destination
requested the key; FALSE = Otherwise. */
    /* 8-bit data. */
    ZBPRO_APS_KeyType_t     keyType;               /*!< Identifies the
type of key material that should be transported. */
    Bool8_t                 apsSecure;
} ZBPRO_APS_TransportKeyReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_SecurityServicesConfParams_t
{
    ZBPRO_APS_Status_t  status;     /*!< The result of the attempt to
perform APSME Security Service Request operation. */
} ZBPRO_APS_SecurityServicesConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.11. ZBPRO_APS_UpdateDeviceReq()

Brief description:

Accepts APSME-UPDATE-DEVICE.request from ZDO Security Manager to ZigBee Pro APS and starts its processing.

Prototype:

```
void ZBPRO_APS_UpdateDeviceReq(ZBPRO_APS_UpdateDeviceReqDescr_t *req)
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_UpdateDeviceReqDescr_t
{
    /* Fields are arranged to minimize paddings */
    ZBPRO_APS_UpdateDeviceConfCallback_t *callback;         /*!<
Confirm callback function.  */
    struct
    {
        SYS_QueueElement_t                queueElement;     /*!< APS
requests service field. */
    } service;
    ZBPRO_APS_UpdateDeviceReqParams_t     params;           /*!<
Request parameters set.     */
} ZBPRO_APS_UpdateDeviceReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_UpdateDeviceConfCallback_t(ZBPRO_APS_UpdateDeviceReqDescr_t
*const reqDescr, ZBPRO_APS_SecurityServicesConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_UpdateDeviceReqParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t             destAddress;            /*!< The
extended 64-bit address of the device that shall be sent the update
information. */
    ZBPRO_APS_ExtAddr_t             deviceAddress;          /*!< The
extended 64-bit address of the device whose status is being updated. */
    /* 16-bit data. */
    ZBPRO_APS_ShortAddr_t           deviceShortAddress;     /*!< The
16-bit network address of the device whose status is being updated. */
    /* 8-bit data. */
    ZBPRO_APS_UpdateDeviceStatus_t  status;                 /*!<
Indicates the updated status of the device given by the DeviceAddress
parameter. */
} ZBPRO_APS_UpdateDeviceReqParams_t;
```

Callback Parameters type:

```
typedef enum _ZBPRO_APS_SecurityInitialStatus_t
{
    ZBPRO_APS_PRECONFIGURED_NETWORK_KEY             = 0x00,
    ZBPRO_APS_PRECONFIGURED_TRUST_CENTER_KEY        = 0x01,
    ZBPRO_APS_PRECONFIGURED_TRUST_CENTER_MASTER_KEY = 0x02, /* Not
supported */
    ZBPRO_APS_NOT_PRECONFIGURED                     = 0x03  /* Not
supported */
} ZBPRO_APS_SecurityInitialStatus_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.12. ZBPRO_APS_RemoveDeviceReq()

Brief description:

ccepts APSME-REMOVE-DEVICE.request from ZDO Security Manager to ZigBee Pro APS and starts its processing.

Prototype:

```
void ZBPRO_APS_RemoveDeviceReq(ZBPRO_APS_RemoveDeviceReqDescr_t *
reqDescr)
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_RemoveDeviceReqDescr_t
{
    /* Fields are arranged to minimize paddings */
    ZBPRO_APS_RemoveDeviceConfCallback_t *callback;         /*!<
Confirm callback function.  */
    struct
    {
        SYS_QueueElement_t              queueElement;     /*!< APS
requests service field. */
    } service;
    ZBPRO_APS_RemoveDeviceReqParams_t     params;           /*!<
Request parameters set.     */
} ZBPRO_APS_RemoveDeviceReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_RemoveDeviceConfCallback_t(ZBPRO_APS_RemoveDeviceReqDescr_t
*const reqDescr, ZBPRO_APS_SecurityServicesConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_RemoveDeviceReqParams_t
{
    ZBPRO_APS_ExtAddr_t  parentAddress;     /*!< The extended 64-bit
address of the device that is the parent of the child device that is
requested to be removed, or the router device that is requested to be
removed. */
    ZBPRO_APS_ExtAddr_t  targetAddress;     /*!< The extended 64-bit
address of the target device that is requested to be removed. If a
router device is requested to be removed, then the \e ParentAddress
shall be the same as the \e TargetAddress. */
} ZBPRO_APS_RemoveDeviceReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_SecurityServicesConfParams_t
{
    ZBPRO_APS_Status_t  status;     /*!< The result of the attempt to
perform APSME Security Service Request operation. */
} ZBPRO_APS_SecurityServicesConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.13. ZBPRO_APS_RequestKeyReq()

Brief description:

Accepts APSME-REQUEST-KEY.request from ZDO Security Manager to ZigBee Pro APS and starts its processing.

Prototype:

```
void ZBPRO_APS_RequestKeyReq(ZBPRO_APS_RequestKeyReqDescr_t *req)
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_RequestKeyReqDescr_t
{
    /* Fields are arranged to minimize paddings */
    ZBPRO_APS_RequestKeyConfCallback_t *callback;       /*!< Confirm
callback function.  */
    struct
    {
        SYS_QueueElement_t              queueElement;   /*!< APS
requests service field. */
    } service;
    ZBPRO_APS_RequestKeyReqParams_t     params;         /*!< Request
parameters set.     */
} ZBPRO_APS_RequestKeyReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_RequestKeyConfCallback_t(ZBPRO_APS_RequestKeyReqDescr_t
*const reqDescr, ZBPRO_APS_SecurityServicesConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_RequestKeyReqParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t  destAddress;       /*!< The extended 64-bit
address of the device to which the request-key command should be sent.
*/
    ZBPRO_APS_ExtAddr_t  partnerAddress;    /*!< If the \e KeyType
parameter indicates an application key, this parameter shall indicate
an extended 64-bit address of a device that shall receive the same key
as the device requesting the key. */
    /* 8-bit data. */
    ZBPRO_APS_KeyType_t  keyType;          /*!< The type of key being
requested. */
} ZBPRO_APS_RequestKeyReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_SecurityServicesConfParams_t
{
    ZBPRO_APS_Status_t  status;     /*!< The result of the attempt to
perform APSME Security Service Request operation. */
} ZBPRO_APS_SecurityServicesConfParams_t;
```

Return value:

None.

5.1.22.1.1.1.14. ZBPRO_APS_SwitchKeyReq()

Brief description:

Accepts APSME-SWITCH-KEY.request from ZDO Security Manager to ZigBee Pro APS and starts its processing.

Prototype:

```
void ZBPRO_APS_SwitchKeyReq(ZBPRO_APS_SwitchKeyReqDescr_t *reqDescr)
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _ZBPRO_APS_SwitchKeyReqDescr_t
{
    /* Fields are arranged to minimize paddings */
    ZBPRO_APS_SwitchKeyConfCallback_t *callback;        /*!< Confirm
callback function.  */
    struct
    {
        SYS_QueueElement_t            queueElement;    /*!< APS
requests service field. */
    } service;
    ZBPRO_APS_SwitchKeyReqParams_t     params;          /*!< Request
parameters set.     */
} ZBPRO_APS_SwitchKeyReqDescr_t;
```

Callback type:

```
typedef void
ZBPRO_APS_SwitchKeyConfCallback_t(ZBPRO_APS_SwitchKeyReqDescr_t
*const reqDescr, ZBPRO_APS_SecurityServicesConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_SwitchKeyReqParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t     destAddress;     /*!< The extended 64-bit
address of the device to which the switch-key command is sent. */
    /* 8-bit data. */
    ZbProSspNwkKeySeqNum_t  keySeqNumber;    /*!< A sequence number
assigned to a network key by the Trust Center and used to distinguish
network keys. */
} ZBPRO_APS_SwitchKeyReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_SecurityServicesConfParams_t
{
    ZBPRO_APS_Status_t  status;     /*!< The result of the attempt to
perform APSME Security Service Request operation. */
} ZBPRO_APS_SecurityServicesConfParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.15. ZBPRO_APS_UpdateDeviceInd()

Brief description:

UpdateDevice Indication function.

Prototype:

```
void ZBPRO_APS_UpdateDeviceInd(ZBPRO_APS_UpdateDeviceIndParams_t *const
indParams)
```

Where `indParams` is a pointer to the indication parameters structure.

Parameters type:

```
typedef struct _ZBPRO_APS_UpdateDeviceIndParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t            srcAddress;            /*!< The
extended 64-bit address of the device originating the update-device
command. */
    ZBPRO_APS_ExtAddr_t            deviceAddress;         /*!< The
extended 64-bit address of the device whose status is being updated. */
    /* 16-bit data. */
    ZBPRO_APS_ShortAddr_t          deviceShortAddress;    /*!< The
16-bit network address of the device whose status is being updated. */
    /* 8-bit data. */
    ZBPRO_APS_UpdateDeviceStatus_t  status;               /*!<
Indicates the updated status of the device given by the DeviceAddress
parameter. */
} ZBPRO_APS_UpdateDeviceIndParams_t;
```

Return value:

```
None.
```

5.1.22.1.1.1.16. ZBPRO_APS_RemoveDeviceInd()

Brief description:

Handles APSME-REMOVE-DEVICE.indication.

Prototype:

```
void ZBPRO_APS_RemoveDeviceInd(ZBPRO_APS_RemoveDeviceIndParams_t
*indParams)
```

Where `indParams` is a pointer to the indication parameters structure.

Parameters type:

```
typedef struct _ZBPRO_APS_RemoveDeviceIndParams_t
{
    ZBPRO_APS_ExtAddr_t  srcAddress;        /*!< The extended 64-bit
address of the device requesting that a child device be removed. */
    ZBPRO_APS_ExtAddr_t  childAddress;      /*!< The extended 64-bit
address of the child device that is requested to be removed. */
} ZBPRO_APS_RemoveDeviceIndParams_t;
```

Return value:

`None.`

5.1.22.1.1.1.17. ZBPRO_APS_SwitchKeyInd()

Brief description:

Handles APSME-SWITCH-KEY.indication.

Prototype:

```
void ZBPRO_APS_SwitchKeyInd(ZBPRO_APS_SwitchKeyIndParams_t *indParams)
```

Where `indParams` is a pointer to the indication parameters structure.

Parameters type:

```
typedef struct _ZBPRO_APS_SwitchKeyIndParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t     srcAddress;      /*!< The extended 64-bit
address of the device that sent the switch-key command. */
    /* 8-bit data. */
    ZbProSspNwkKeySeqNum_t  keySeqNumber;    /*!< A sequence number
assigned to a network key by the Trust Center and used to distinguish
network keys. */
} ZBPRO_APS_SwitchKeyIndParams_t;
```

Return value:

`None.`

5.1.23. ZigBee Generic Device Profile Sublayer

5.1.23.1. API

5.1.23.1.1. Starting profile

5.1.23.1.2. Common API

This chapter describes common GDP 2 functions and their descriptors including data types and required parameters.

A common place for all functions in this chapter would be Request Service type:

```
typedef struct _RF4CE_NWK_RequestService_t
{
    SYS_QueueElement_t serviceData; /*!< Helper field to allow that
structure object to be queued. */
    uint8_t requestID; /*!< Request ID. */
} RF4CE_NWK_RequestService_t;
```

Figure 20: Request Service type for GDP

Descriptors sctructure is common and contains service field descriptor, Callback type and Parameters type.

5.1.23.1.2.1. RF4CE_GDP2_HeartbeatReq()

Accepts request from the application to issue the Heartbeat GDP command.

Prototype:

```
void rf4ceGdp2HeartbeatReq(Rf4ceGdp2HeartbeatReqDescr_t *const
reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor object.

Descriptor:

```
struct _Rf4ceGdp2HeartbeatReqDescr_t
{
    /* 32-bit data. */
    Rf4ceGdp2HeartbeatConfCallback_t  callback;          /*!< Entry
point of the confirmation callback function. */

#ifndef _HOST_
    /* Structured data. */
    RF4CE_NWK_RequestService_t        service; /*!< Service field. */
#endif

    /* Structured data. */
    Rf4ceGdp2HeartbeatReqParams_t     params;            /*!< Request
parameters structured object. */

};
```

Service field type Brief description:

See **Figure 20: Request Service type for GDP**

Callback type:

```
typedef void (*Rf4ceGdp2HeartbeatConfCallback_t)
(Rf4ceGdp2HeartbeatReqDescr_t   *const reqDescr,
Rf4ceGdp2HeartbeatConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _Rf4ceGdp2HeartbeatReqParams_t
{
    /* 8-bit data. */
    RF4CE_GDP2_PollingTriggerId_t  pollingTriggerId;        /*!<
Identifier of the Polling Trigger to be fired. */

    uint8_t                        pairingRef;              /*!<
Pairing reference of the linked Poll Server node. */

} Rf4ceGdp2HeartbeatReqParams_t;
```

Callback Parameters type:

```
typedef struct _Rf4ceGdp2HeartbeatConfParams_t
{
    /* 8-bit data. */
    RF4CE_ZRC2GDP2_Status_t  status;        /*!< Status to be
confirmed. */

} Rf4ceGdp2HeartbeatConfParams_t;
```

5.1.23.1.3. *Controller only API*

5.1.23.1.3.1. RF4CE_GDP2_ClientNotificationInd()

ZRC 2.0 Client notification request indication.

Prototype:

```
void rf4ceGdp2IdentifyInd(RF4CE_GDP2_ClientNotificationIndParams_t
*const clientNotificationIndParams);
```

Where `indication` is a pointer to the Client Notification indication structure.

Parameters type:

```
typedef struct _RF4CE_GDP2_ClientNotificationIndParams_t
{
    /* Structured data. */
    SYS_DataPointer_t                        payload;
/*!< Client Notification Payload field. */

    /* 8-bit data. */
    RF4CE_ZRC2_ClientNotificationSubTypeId_t
clientNotificationSubType;    /*!< Client Notification Sub-Type field.
*/
```

```
    uint8_t                                  pairingRef;
/*!< Pairing reference of the linked Poll

Server node. */
} RF4CE_GDP2_ClientNotificationIndParams_t;
```

5.1.23.1.4. Target only API

5.1.23.1.4.1. RF4CE_GDP2_HeartbeatInd()

ZRC 2.0 Heartbeat request indication.

Prototype:

```
void RF4CE_GDP2_HeartbeatInd(RF4CE_GDP2_HeartbeatIndParams_t *const
indParams);
```

Where indication is a pointer to the Pairing Reference indication structure.

Parameters type:

```
typedef struct _RF4CE_GDP2_HeartbeatIndParams_t
{
    /* 8-bit data. */
    RF4CE_GDP2_PollingTriggerId_t  pollingTriggerId;        /*!<
Identifier of the fired Polling Trigger. */

    uint8_t                        pairingRef;              /*!<
Pairing reference of the linked Poll Client node. */

} RF4CE_GDP2_HeartbeatIndParams_t;
```

5.1.24. ZigBee ZDO Sublayer

5.1.24.1. Device Features

5.1.24.1.1. Discovery

5.1.24.1.1.1. Functions

5.1.24.1.1.1.1. ZBPRO_ZDO_MatchDescReq()

Brief description:

The ability of a device to locate services of interest.

Prototype:

```
void ZBPRO_APS_BindReq(ZBPRO_APS_BindUnbindReqDescr_t *reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_APS_BindUnbindReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_APS_BindUnbindConfCallback_t *callback;          /*!<
Confirm callback function. */
    /* Structured / 32-bit data. */
    struct
    {
        SYS_QueueElement_t              queueElement;      /*!< APS
requests service field. */
        Bool8_t                         isBindRequest;     /*!< TRUE
for the case of APSME-BIND.request; FALSE for the case of APSME-
UNBIND.request. */
    } service;                                             /*!<
Service field. */
    /* Structured data. */
    ZBPRO_APS_BindUnbindReqParams_t     params;            /*!<
Request parameters set. */
};
```

Callback type:

```
typedef void
ZBPRO_APS_BindUnbindConfCallback_t(ZBPRO_APS_BindUnbindReqDescr_t
*const reqDescr, ZBPRO_APS_BindUnbindConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_APS_BindUnbindReqParams_t
{
    /* 64-bit data. */
    ZBPRO_APS_ExtAddr_t     srcAddress;     /*!< The source IEEE
address for the binding entry. */

    /* Structured data, aligned at 32 bits. */
    ZBPRO_APS_Address_t     dstAddr;        /*!< The destination
address mode and address for the binding entry. */
    /* 16-bit data. */
    ZBPRO_APS_ClusterId_t   clusterId;      /*!< The identifier of the
cluster on the source device that is to be (un)bound to/from the
destination device. */
    /* 8-bit data. */
    ZBPRO_APS_EndpointId_t  srcEndpoint;    /*!< The source endpoint
for the binding entry. */
    ZBPRO_APS_EndpointId_t  dstEndpoint;    /*!< The destination
endpoint for the binding entry. */
} ZBPRO_APS_BindUnbindReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_APS_BindUnbindConfParams_t
{
    ZBPRO_APS_Status_t       status;        /*!< The results of the
(un)binding request. */
} ZBPRO_APS_BindUnbindConfParams_t;
```

Return value:

```
None.
```

5.1.24.1.1.1.2. ZBPRO_ZDO_AddrResolvingReq()

Brief description:

Accepts ZDO Local Request to issue ZDP NWK_Addr_req or IEEE_Addr_req command.

Prototype:

```
void ZBPRO_ZDO_AddrResolvingReq(
                ZBPRO_ZDO_AddrResolvingReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_AddrResolvingReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZDO_AddrResolvingConfCallback_t *callback;        /*!< ZDO
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZdoLocalRequest_t                 service;         /*!< ZDO
Request Descriptor service field. */
    ZBPRO_ZDO_AddrResolvingReqParams_t     params;          /*!< ZDO
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZDO_AddrResolvingConfCallback_t(
                ZBPRO_ZDO_AddrResolvingReqDescr_t   *const  reqDescr,
                ZBPRO_ZDO_AddrResolvingConfParams_t *const
confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_AddrResolvingReqParams_t
{
    /* Structured data, aligned at 32 bits. */
    ZBPRO_ZDO_Address_t              zdpDstAddress;        /*!<
Destination address. May be either unicast or broadcast to all devices
for which macRxOnWhenIdle = TRUE. Must be unicast address for the case
when IEEE_Addr_req is requested. */
    ZBPRO_ZDO_Address_t              addrOfInterest;       /*!< Either
the IEEE address to be matched by the Remote Device, or NWK address
that is used for IEEE address mapping. This field denotes also if
NWK_Addr_req or IEEE_Addr_req shall be issued. */
    /* 8-bit data. */
    ZBPRO_ZDO_AddrResolvingReqType_t  requestType;          /*!<
Request type for this command. */
    uint8_t                          startIndex;           /*!< The
starting index for the requested elements of the associated devices
list. Used only if \c requestType is equal to 'Extended response'
(0x01). */
} ZBPRO_ZDO_AddrResolvingReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_AddrResolvingConfParams_t
```

```
{
    /* 64-bit data. */
    ZBPRO_ZDO_ExtAddr_t  extAddrRemoteDev;      /*!< 64-bit address for
the Remote Device. */
    /* Structured / 32-bit data. */
    SYS_DataPointer_t    payload;               /*!< The \c
NWKAddrAssocDevList field. A list of 16-bit addresses, one
corresponding to each associated device to Remote Device. This field
set to EMPTY if \c status is not SUCCESS, or the original request of
the 'Single Device Response' type, or the list is empty. */
    /* 16-bit data. */
    ZBPRO_ZDO_NwkAddr_t  nwkAddrRemoteDev;      /*!< 16-bit address for
the Remote Device. */
    /* 8-bit data. */
    ZBPRO_ZDO_Status_t   status;                /*!< The status of the
Address Resolving request command. */
    uint8_t              numAssocDev;           /*!< Count of the
number of 16-bit short addresses to follow. This field is set to ZERO
if \c status field is not SUCCESS, or the original request is of the
'Single Device Response' type, or there are no associated devices on
the Remote Device. */
    uint8_t              startIndex;            /*!< Starting index
into the list of associated devices for this report. This field is set
to ZERO if \c status field is not SUCCESS, or the original request is
of the 'Single Device Response' type, or there are no associated
devices on the Remote Device. */
} ZBPRO_ZDO_AddrResolvingConfParams_t;
```

Return value:

None.

5.1.24.1.1.1.3. ZBPRO_ZDO_NodeDescReq()

Brief description:

ZDO ZDP Node_Desc_req function.

Prototype:

```
void ZBPRO_ZDO_NodeDescReq(ZBPRO_ZDO_NodeDescReqDescr_t *const
reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_NodeDescReqDescr_t
{
    ZbProZdoLocalRequest_t              service;
    ZBPRO_ZDO_NodeDescConfCallback_t   *callback;
    ZBPRO_ZDO_NodeDescReqParams_t       params;
};
```

Callback type:

```
typedef void
ZBPRO_ZDO_NodeDescConfCallback_t(ZBPRO_ZDO_NodeDescReqDescr_t   *const
reqDescr, ZBPRO_ZDO_NodeDescConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_NodeDescReqParams_t
{
    ZBPRO_ZDO_Address_t                 zdpDstAddress;
    ZBPRO_ZDO_NwkAddr_t                 nwkAddrOfInterest;
    SYS_Time_t                          respWaitTimeout;    /* Response
waiting timeout, in milliseconds. * Zero means 'Use default ZDO
timeout'. */
} ZBPRO_ZDO_NodeDescReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_NodeDescConfParams_t
{
    ZBPRO_ZDO_Status_t                  status;
    ZBPRO_ZDO_NwkAddr_t                 nwkAddrOfInterest;
    ZbProZdoNodeDescriptor_t            nodeDescriptor;
} ZBPRO_ZDO_NodeDescConfParams_t;
```

Return value:

None.

5.1.24.1.1.1.4. ZBPRO_ZDO_PowerDescReq()

Brief description:

ZDO ZDP Power_Desc_req function.

Prototype:

```
void ZBPRO_ZDO_PowerDescReq(ZBPRO_ZDO_PowerDescReqDescr_t *const
reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_PowerDescReqDescr_t
{
    ZbProZdoLocalRequest_t              service;
    ZBPRO_ZDO_PowerDescConfCallback_t  *callback;
    ZBPRO_ZDO_PowerDescReqParams_t      params;
};
```

Callback type:

```
typedef void
ZBPRO_ZDO_PowerDescConfCallback_t(ZBPRO_ZDO_PowerDescReqDescr_t
*const reqDescr, ZBPRO_ZDO_PowerDescConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_PowerDescReqParams_t
{
    ZBPRO_ZDO_Address_t                 zdpDstAddress;
    ZBPRO_ZDO_NwkAddr_t                 nwkAddrOfInterest;
    SYS_Time_t                          respWaitTimeout;    /* Response
waiting timeout, in milliseconds. * Zero means 'Use default ZDO
timeout'. */
} ZBPRO_ZDO_PowerDescReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_PowerDescConfParams_t
{
    ZBPRO_ZDO_Status_t                  status;
    ZBPRO_ZDO_NwkAddr_t                 nwkAddrOfInterest;
    ZbProZdoPowerDescriptor_t           powerDescriptor;
} ZBPRO_ZDO_PowerDescConfParams_t;
```

Return value:

```
None.
```

5.1.24.1.1.1.5. ZBPRO_ZDO_SimpleDescReq()

Brief description:

ZDO ZDP Simple_Desc_req function.

Prototype:

```
void ZBPRO_ZDO_SimpleDescReq(ZBPRO_ZDO_SimpleDescReqDescr_t *const
reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_SimpleDescReqDescr_t
{
    ZbProZdoLocalRequest_t              service;
    ZBPRO_ZDO_SimpleDescConfCallback_t *callback;
    ZBPRO_ZDO_SimpleDescReqParams_t     params;
};
```

Callback type:

```
typedef void
ZBPRO_ZDO_SimpleDescConfCallback_t(ZBPRO_ZDO_SimpleDescReqDescr_t
*const reqDescr, ZBPRO_ZDO_SimpleDescConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_SimpleDescReqParams_t
{
    ZBPRO_ZDO_Address_t                 zdpDstAddress;
    ZBPRO_ZDO_NwkAddr_t                 nwkAddrOfInterest;
    ZBPRO_ZDO_Endpoint_t                endpoint;
    SYS_Time_t                          respWaitTimeout;    /* Response
waiting timeout, in milliseconds. * Zero means 'Use default ZDO
timeout'. */
} ZBPRO_ZDO_SimpleDescReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_SimpleDescConfParams_t
{
    ZBPRO_ZDO_Status_t                  status;
    ZBPRO_ZDO_NwkAddr_t                 nwkAddrOfInterest;
    uint8_t                             length;
    ZBPRO_APS_SimpleDescriptor_t        simpleDescriptor;
} ZBPRO_ZDO_SimpleDescConfParams_t;
```

Return value:

```
None.
```

5.1.24.1.1.1.6. ZBPRO_ZDO_ActiveEpReq()

Brief description:

ZDO ZDP Active_Ep request function.

Prototype:

```
void ZBPRO_ZDO_ActiveEpReq(ZBPRO_ZDO_ActiveEpReqDescr_t *const
reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_ActiveEpReqDescr_t
{
    ZbProZdoLocalRequest_t               service;
    ZBPRO_ZDO_ActiveEpConfCallback_t    *callback;
    ZBPRO_ZDO_ActiveEpReqParams_t        params;
};
```

Callback type:

```
typedef void
ZBPRO_ZDO_ActiveEpConfCallback_t(ZBPRO_ZDO_ActiveEpReqDescr_t *const
reqDescr, ZBPRO_ZDO_ActiveEpConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_ActiveEpReqParams_t
{
    ZBPRO_ZDO_Address_t     zdpDstAddress;
    ZBPRO_ZDO_NwkAddr_t     nwkAddrOfInterest;
    SYS_Time_t              respWaitTimeout;    /* Response waiting
timeout, in milliseconds. * Zero means 'Use default ZDO timeout'. */
} ZBPRO_ZDO_ActiveEpReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_ActiveEpConfParams_t
{
    ZBPRO_ZDO_Status_t      status;
    ZBPRO_ZDO_NwkAddr_t     nwkAddrOfInterest;
    uint8_t                 activeEpCount;
    SYS_DataPointer_t       activeEpList;
} ZBPRO_ZDO_ActiveEpConfParams_t;
```

Return value:

```
None.
```

5.1.24.1.1.1.7. ZBPRO_ZDO_DeviceAnnceReq()

Brief description:

Accepts ZDO Local Request to issue ZDP Device_Annce command.

Prototype:

```
void ZBPRO_ZDO_DeviceAnnceReq(
                ZBPRO_ZDO_DeviceAnnceReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_DeviceAnnceReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZDO_DeviceAnnceConfCallback_t *callback;      /*!< ZDO
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZdoLocalRequest_t              service;        /*!< ZDO
Request Descriptor service field. */
    ZBPRO_ZDO_DeviceAnnceReqParams_t    params;         /*!< ZDO
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZDO_DeviceAnnceConfCallback_t(
                ZBPRO_ZDO_DeviceAnnceReqDescr_t   *const  reqDescr,
                ZBPRO_ZDO_DeviceAnnceConfParams_t *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_DeviceAnnceReqParams_t
{
    /* 64-bit data. */
    ZBPRO_ZDO_ExtAddr_t     extAddr;        /*!< IEEE address for the
Local Device. */
    /* 16-bit data. */
    ZBPRO_ZDO_NwkAddr_t     nwkAddr;        /*!< NWK address for the
Local Device. */
    /* 8-bit data. */
    ZBPRO_ZDO_Capability_t  capability;     /*!< Capability of the
local device. */
} ZBPRO_ZDO_DeviceAnnceReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_DeviceAnnceConfParams_t
{
    /* 8-bit data. */
    ZBPRO_ZDO_Status_t  status;         /*!< Status field. */
} ZBPRO_ZDO_DeviceAnnceConfParams_t;
```

Return value:

```
None.
```

5.1.24.1.1.1.8. ZBPRO_ZDO_EndDeviceBindReq()

Brief description:

Accepts ZDO Local Request to issue ZDP End_Device_Bind_req command.

Prototype:

```
void ZBPRO_ZDO_EndDeviceBindReq(
                ZBPRO_ZDO_EndDeviceBindReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_EndDeviceBindReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZDO_EndDeviceBindConfCallback_t *callback;        /*!< ZDO
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZdoLocalRequest_t                 service;         /*!< ZDO
Request Descriptor service field. */
    ZBPRO_ZDO_EndDeviceBindReqParams_t     params;          /*!< ZDO
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZDO_EndDeviceBindConfCallback_t(
                ZBPRO_ZDO_EndDeviceBindReqDescr_t *const  reqDescr,
                ZBPRO_ZDO_BindConfParams_t        *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_EndDeviceBindReqParams_t
{
    /* 64-bit data. */
    ZBPRO_ZDO_ExtAddr_t    srcIeeeAddress;          /*!< The IEEE
address of the device generating the request. */
    /* Structured / 32-bit data. */
    SYS_DataPointer_t      clusterList;             /*!< List of Input
ClusterIDs followed with the list of Output ClusterIDs to be used for
matching. The first part, the InClusterList, is the desired list to be
matched by the ZigBee coordinator with the Remote Device�s output
clusters (the elements of the InClusterList are supported input
clusters of the Local Device). The second part, the OutClusterList, is
to be matched with the Remote Device�s input clusters. */
    /* 16-bit data. */
    ZBPRO_ZDO_NwkAddr_t    bindingTarget;           /*!< The address of
the target for the binding. This can be either the primary binding
cache device or the short address of the local device. */
    ZBPRO_ZDO_ProfileId_t  profileId;               /*!< Profile
identifier which is to be matched between two End_Device_Bind_req
received at the ZigBee Coordinator within the timeout value pre-
configured in the ZigBee Coordinator. */
```

```
    /* 8-bit data. */
    ZBPRO_ZDO_Endpoint_t   srcEndpoint;            /*!< The endpoint
on the device generating the request. */
    uint8_t                numInClusters;          /*!< The number of
Input Clusters provided for end device binding within the
InClusterList. */
    uint8_t                numOutClusters;         /*!< The number of
Output Clusters provided for matching within the OutClusterList. */
} ZBPRO_ZDO_EndDeviceBindReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_BindConfParams_t
{
    /* 8-bit data. */
    ZBPRO_ZDO_Status_t  status;         /*!< The status of the
End_Device_Bind_req, Bind_req, or Unbind_req command. */
} ZBPRO_ZDO_BindConfParams_t;
```

Return value:

```
None.
```

5.1.24.1.1.1.9. ZBPRO_ZDO_BindReq()

Brief description:

Accepts ZDO Local Request to issue ZDP Bind_req command.

Prototype:

```
void ZBPRO_ZDO_BindReq(
                ZBPRO_ZDO_BindUnbindReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_BindUnbindReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZDO_BindUnbindConfCallback_t *callback;       /*!< ZDO
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZdoLocalRequest_t             service;         /*!< ZDO
Request Descriptor service field. */
    ZBPRO_ZDO_BindUnbindReqParams_t    params;          /*!< ZDO
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZDO_BindUnbindConfCallback_t(
                ZBPRO_ZDO_BindUnbindReqDescr_t *const  reqDescr,
                ZBPRO_ZDO_BindConfParams_t     *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_BindUnbindReqParams_t
{
    /* 64-bit data. */
    ZBPRO_ZDO_ExtAddr_t    srcAddress;          /*!< The IEEE address
for the source. */
    /* Structured data, aligned at 32 bits. */
    ZBPRO_ZDO_Address_t    zdpDstAddress;       /*!< Destination
address. It shall be unicast only, and shall be that of a Primary
binding table cache or to the SrcAddress itself. */
    ZBPRO_ZDO_Address_t    dstAddress;          /*!< The destination
address for the binding entry, and the addressing mode for the
destination address used in this command. The destination address may
be either 16-bit group address for DstAddress and DstEndp not present,
or 64-bit extended address for DstAddress and DstEndp present. */
    /* 16-bit data. */
    ZBPRO_ZDO_ClusterId_t  clusterId;           /*!< The identifier of
the cluster on the source device that is bound to the destination. */
    /* 8-bit data. */
    ZBPRO_ZDO_Endpoint_t   srcEndp;             /*!< The source
endpoint for the binding entry. */
    ZBPRO_ZDO_Endpoint_t   dstEndp;             /*!< The destination
endpoint for the binding entry. This parameter is treated only if the
DstAddrMode field has a value of 0x03; otherwise it's ignored. */
} ZBPRO_ZDO_BindUnbindReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_BindConfParams_t
{
    /* 8-bit data. */
    ZBPRO_ZDO_Status_t  status;         /*!< The status of the
End_Device_Bind_req, Bind_req, or Unbind_req command. */
} ZBPRO_ZDO_BindConfParams_t;
```

Return value:

None.

5.1.24.1.1.1.10. ZBPRO_ZDO_UnbindReq()

Brief description:

Accepts ZDO Local Request to issue ZDP Unbind_req command.

Prototype:

```
void ZBPRO_ZDO_UnbindReq(
                ZBPRO_ZDO_BindUnbindReqDescr_t *const  reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZDO_BindUnbindReqDescr_t
{
    /* 32-bit data. */
    ZBPRO_ZDO_BindUnbindConfCallback_t *callback;       /*!< ZDO
Confirmation callback handler entry point. */
    /* Structured data, aligned at 32 bits. */
    ZbProZdoLocalRequest_t              service;        /*!< ZDO
Request Descriptor service field. */
    ZBPRO_ZDO_BindUnbindReqParams_t     params;         /*!< ZDO
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZDO_BindUnbindConfCallback_t(
                ZBPRO_ZDO_BindUnbindReqDescr_t *const  reqDescr,
                ZBPRO_ZDO_BindConfParams_t     *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZDO_BindUnbindReqParams_t
{
    /* 64-bit data. */
    ZBPRO_ZDO_ExtAddr_t    srcAddress;          /*!< The IEEE address
for the source. */
    /* Structured data, aligned at 32 bits. */
    ZBPRO_ZDO_Address_t    zdpDstAddress;       /*!< Destination
address. It shall be unicast only, and shall be that of a Primary
binding table cache or to the SrcAddress itself. */
    ZBPRO_ZDO_Address_t    dstAddress;          /*!< The destination
address for the binding entry, and the addressing mode for the
destination address used in this command. The destination address may
be either 16-bit group address for DstAddress and DstEndp not present,
or 64-bit extended address for DstAddress and DstEndp present. */
    /* 16-bit data. */
    ZBPRO_ZDO_ClusterId_t  clusterId;           /*!< The identifier of
the cluster on the source device that is bound to the destination. */
    /* 8-bit data. */
    ZBPRO_ZDO_Endpoint_t   srcEndp;             /*!< The source
endpoint for the binding entry. */
    ZBPRO_ZDO_Endpoint_t   dstEndp;             /*!< The destination
endpoint for the binding entry. This parameter is treated only if the
DstAddrMode field has a value of 0x03; otherwise it's ignored. */
} ZBPRO_ZDO_BindUnbindReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZDO_BindConfParams_t
{
    /* 8-bit data. */
    ZBPRO_ZDO_Status_t  status;         /*!< The status of the
End_Device_Bind_req, Bind_req, or Unbind_req command. */
} ZBPRO_ZDO_BindConfParams_t;
```

Return value:

```
None.
```

5.2. CIE Device

The IAS CIE device is the central Control and Indicating Equipment for an Intruder Alarm System. It receives inputs from sensors (Zones) and control equipment (ACE), and sends output to a warning device (WD).

Note

This CIE Device is intended primarily for testing purposes, and contains certain functional limitations.
If additional functionality is required, you can fully access IAS Zone, IAS ACE, and IAS WD clusters by switching CIE Device off.

5.2.1. Device features

5.2.1.1. Indications processing

CIE Device intercepts indications of the following Home Automation clusters:

IAS Zone:

ZBPRO_ZCL_IASZoneCmdZoneEnrollRequestInd() [5.1.18.1.2.]
ZBPRO_ZCL_IASZoneCmdZoneStatusChangeNotificationInd() [5.1.18.1.3.]

IAS ACE:

ZBPRO_ZCL_SapIasAceArmInd() [5.1.20.1.2.]
ZBPRO_ZCL_SapIasAceBypassInd() [5.1.20.1.12.
ZBPRO_ZCL_SapIasAceEmergencyInd() [5.1.20.1.13.]
ZBPRO_ZCL_SapIasAceFireInd() [5.1.20.1.10.]
ZBPRO_ZCL_SapIasAcePanicInd() [5.1.20.1.11.]
ZBPRO_ZCL_SapIasAceGetZoneIdMapInd() [5.1.20.1.4.]
ZBPRO_ZCL_SapIasAceGetZoneInfoInd() [5.1.20.1.5.]
ZBPRO_ZCL_SapIasAceGetPanelStatusInd() [5.1.20.1.6.]
ZBPRO_ZCL_SapIasAcePanelStatusChangedReq() [5.1.20.1.8.]
ZBPRO_ZCL_SapIasAceGetZoneStatusInd()[5.1.20.1.15.]

Note

When CIE Device has been registered, these indications will not come out of stack.
But when the CIE Device has not been registered or has been unregistered, all listed indications will be broadcasted to Application.

5.2.1.2. Functions

5.2.1.2.1. ZBPRO_ZHA_CieDeviceRegisterReq()

Brief description:

Privitive for the CIE Device registration. Only one CIE Device could be registered. All other Register Requests will return the status NOT_PERMITTED in the confirmation..

Prototype:

```
void ZBPRO_ZHA_CieDeviceRegisterReq(
    ZBPRO_ZHA_CieDeviceRegisterReqDescr_t * const descr);
```

Where `descr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZHA_CieDeviceRegisterReqDescr_t
{
    /* 32-bit data. */
    ZbProZhaCieDeviceReqDesr_t                     service;    /*!<
Internal field. Shoul not be used by Application. */
    ZBPRO_ZHA_CieDeviceRegisterConfCallback_t     *callback;   /*!< ZCL
Confirmation callback handler entry point. */
    ZBPRO_ZHA_CieDeviceRegisterReqParams_t         params;     /*!< ZCL
Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZHA_CieDeviceRegisterConfCallback_t(
        ZBPRO_ZHA_CieDeviceRegisterReqDescr_t   *const  reqDescr,
        ZBPRO_ZHA_CieDeviceRegisterConfParams_t *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZHA_CieDeviceRegisterReqParams_t
{
    ZBPRO_APS_EndpointId_t       endpoint;
} ZBPRO_ZHA_CieDeviceRegisterReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZHA_CieDeviceRegisterConfParams_t
{
    ZBPRO_ZHA_CieDeviceStatus_t  status;
} ZBPRO_ZHA_CieDeviceRegisterConfParams_t;
```

Return value:

```
None.
```

5.2.1.2.2. ZBPRO_ZHA_CieDeviceUnregisterReq()

Brief description:

Privitive for the CIE Device unregistration.

Prototype:

```
void ZBPRO_ZHA_CieDeviceUnregisterReq(
     ZBPRO_ZHA_CieDeviceUnregisterReqDescr_t * const descr);
```

Where `descr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZHA_CieDeviceUnregisterReqDescr_t
{
    /* 32-bit data. */
    ZbProZhaCieDeviceReqDesr_t                       service;    /*!<
Internal field. Shoul not be used by Application. */
    ZBPRO_ZHA_CieDeviceUnregisterConfCallback_t     *callback;   /*!<
ZCL Confirmation callback handler entry point. */
    ZBPRO_ZHA_CieDeviceUnregisterReqParams_t         params;     /*!<
ZCL Request parameters structure. */
};
```

Callback type:

```
typedef void ZBPRO_ZHA_CieDeviceUnregisterConfCallback_t(
        ZBPRO_ZHA_CieDeviceUnregisterReqDescr_t   *const  reqDescr,
        ZBPRO_ZHA_CieDeviceUnregisterConfParams_t *const  confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZHA_CieDeviceUnregisterReqParams_t
{
    ZBPRO_APS_EndpointId_t       endpoint;
} ZBPRO_ZHA_CieDeviceUnregisterReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZHA_CieDeviceUnregisterConfParams_t
{
    ZBPRO_ZHA_CieDeviceStatus_t  status;
} ZBPRO_ZHA_CieDeviceUnregisterConfParams_t;
```

Return value:

```
None.
```

5.2.1.2.3. ZBPRO_ZHA_CieDeviceEnrollReq()

Brief description:

Privitive for the CIE Device Enroll procedure executing.

Prototype:

```
void ZBPRO_ZHA_CieDeviceEnrollReq(ZBPRO_ZHA_CieEnrollReqDescr_t
*reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZHA_CieEnrollReqDescr_t
{
    SYS_QueueElement_t   queueElement; // NOTE: may be need to replace
ZbProZhaServiceField_t...
    ZBPRO_ZHA_CieEnrollReqParams_t   params;
    ZBPRO_ZHA_CieEnrollCallback_t    callback;
};
```

Callback type:

```
typedef void (*ZBPRO_ZHA_CieEnrollCallback_t)
(ZBPRO_ZHA_CieEnrollReqDescr_t *const reqDescr,
ZBPRO_ZHA_CieEnrollConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZHA_CieEnrollReqParams_t
{
    ZBPRO_APS_Address_t   addr;   // NOTE: can be broadcast
    SYS_Time_t                          scanDurationMs;
    SYS_Time_t                          permitEnrollDurationMs;
    Bool8_t                             autoREsponseMode;
} ZBPRO_ZHA_CieEnrollReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZHA_CieEnrollConfParams_t
{
    ZBPRO_ZHA_CieEnrollStatus_t status;
} ZBPRO_ZHA_CieEnrollConfParams_t;
```

Return value:

`None.`

5.2.1.2.4. *ZBPRO_ZHA_CieDeviceEnrollInd()*

Brief description:

Indication for finishing enrolling process for one zone.

Prototype:

```
void ZBPRO_ZHA_CieDeviceEnrollInd(ZBPRO_ZHA_CieEnrollIndParams_t *const
indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZHA_CieEnrollIndParams_t
{
    uint8_t zoneID;                          /*!< ZoneID. */
    ZBPRO_ZCL_SapIASZoneAttributeZoneType_t zoneType;       /*!< Type
of zone. */
    ZBPRO_APS_ExtAddr_t zoneAddr; /*!< Zone device extended address. */
    ZBPRO_APS_EndpointId_t ep;    /*!< Zone device endpoint */
} ZBPRO_ZHA_CieEnrollIndParams_t;
```

Return value:

`None.`

5.2.1.2.5. *ZBPRO_ZHA_CieDeviceSetPanelStatusReq()*

Brief description:

This function proceed some signal from user to change a CIE Panel Status (it could be some button for example.)

Prototype:

```
void
ZBPRO_ZHA_CieDeviceSetPanelStatusReq(ZBPRO_ZHA_CieSetPanelStatusReqDesc
r_t * const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZHA_CieSetPanelStatusReqDescr_t
{
    ZBPRO_ZHA_CieSetPanelStatusReqParams_t   params;
    ZBPRO_ZHA_CieSetPanelStatusCallback_t    callback;
};
```

Callback type:

```
typedef void (*ZBPRO_ZHA_CieSetPanelStatusCallback_t)
(ZBPRO_ZHA_CieSetPanelStatusReqDescr_t *const reqDescr,
ZBPRO_ZHA_CieSetPanelStatusConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZHA_CieSetPanelStatusReqParams_t
{
    ZbProZhaCiePanelStatus_t  status;  /*new status for the CIE Panel*/
} ZBPRO_ZHA_CieSetPanelStatusReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZHA_CieSetPanelStatusConfParams_t
{
    ZbProZhaCiePanelStatus_t  status;
} ZBPRO_ZHA_CieSetPanelStatusConfParams_t;
```

Return value:

`None.`

5.2.1.2.6. ZBPRO_ZHA_CieDeviceSetPanelStatusInd()

Brief description:

Indication of the Set Panel Status command finished.

Prototype:

```
void ZBPRO_ZHA_CieDeviceSetPanelStatusInd(
    ZBPRO_ZHA_CieSetPanelStatusIndParams_t   *const   indParams);
```

Where `indParams` is a pointer to the indication descriptor structure.

Parameters type:

```
typedef struct _ZBPRO_ZHA_CieSetPanelStatusIndParams_t
{
    ZbProZhaCiePanelStatus_t             currentStatus;
    ZBPRO_ZHA_CieSetPanelStatusResult_t  result;
} ZBPRO_ZHA_CieSetPanelStatusIndParams_t;
```

Return value:

`None.`

5.2.1.2.7. ZBPRO_ZHA_CieZoneSetBypassStateReq()

Brief description:

Set Zone Bypass status command. It is used to set Bypass status to the Zone from Zone Table.

Prototype:

```
void ZBPRO_ZHA_CieZoneSetBypassStateReq(
    ZBPRO_ZHA_CieZoneSetBypassStateReqDescr_t   *const descr);
```

Where `reqDescr` is a pointer to the request descriptor structure..

Descriptor:

```
struct _ZBPRO_ZHA_CieZoneSetBypassStateReqDescr_t
{
    ZBPRO_ZHA_CieZoneSetBypassStateReqParams_t   params;
    ZBPRO_ZHA_CieZoneSetBypassStateCallback_t    callback;
};
```

Callback type:

```
typedef void (*ZBPRO_ZHA_CieZoneSetBypassStateCallback_t)
(ZBPRO_ZHA_CieZoneSetBypassStateReqDescr_t *const reqDescr,
        ZBPRO_ZHA_CieZoneSetBypassStateConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _ZBPRO_ZHA_CieZoneSetBypassStateReqParams_t
{
    uint8_t                            zoneID;          /*!< ID of the
zone, which state must be changed. */
    ZBPRO_ZHA_CieZoneBypassState_t     bypassState;     /*!< Bypass
Status, which must be set. */
} ZBPRO_ZHA_CieZoneSetBypassStateReqParams_t;
```

Callback Parameters type:

```
typedef struct _ZBPRO_ZHA_CieZoneSetBypassStateConfParams_t
{
    ZBPRO_ZHA_CieSetBypassStateResult_t  result;         /*!< Result of
the Set Bypass State request. */
} ZBPRO_ZHA_CieZoneSetBypassStateConfParams_t;
```

Return value:

```
None.
```

6. API for ZigBee System

6.1. Software Download & Start ZigBee CPU

ZigBee hardware block will not have own non-volatile memory, and there is no place to hold the BroadBee software permanently. So, the BroadBee software will be saved on the flash memory of the Host. After the power on, the Host software has to download the BroadBee binary file into ZigBee internal instruction memory.

The procedure to download the BroadBee software into ZigBee block should be

1) Set the RF4CE_CPU_CTRL_CTRL.CPU_RST bit.
2) Clear the RF4CE_CPU_CTRL_CTRL.START_ARC bit to hold the ZigBee CPU.
3) Clear the RF4CE_CPU_CTRL_CTRL.CPU_RST bit.
4) Write the first 128K bytes of the binary BroadBee software from non-volatile memory into the RF4CE_CPU_PROG0_MEM_WORD[0..32767].
5) Write the second 128K bytes of the binary BroadBee software from non-volatile memory into the RF4CE_CPU_PROG1_MEM_WORD[0..32767].
6) Set the RF4CE_CPU_CTRL_CTRL.START_ARC bit to start the ZigBee's ARC CPU.

When the START_ARC bit has been set, the ZigBee software will start boot-up, will disable the access from Host into the ARC internal instruction and data memories to prevent any potential corruption from Host by setting RF4CE_CPU_CTRL_UB_ACCESS_LOCK.ICM_LOCK = 1 and RF4CE_CPU_CTRL_UB_ACCESS_LOCK.DCM_LOCK = 1, and will start the normal tasks of the ZigBee software stacks. Since then, all communication between ZigBee and Host systems will be done through the hardware mailboxes.

6.1.1. Functions

6.1.1.1. BroadBee_SwDownloadAndStart()

Prototype:

```
BroadBee_HostAccessResult_t BroadBee_SwDownloadAndStart(
                    uint32_t *BroadBeeBinary, uint16_t length);
```

Brief description:

This function is to download the BroadBee binary file into the internal instruction memory of ZigBee hardware, and start the ZigBee CPU. Note that **this function is synchronous.**

Parameters:

`BroadBeeBinary`: Pointer to the BroadBee software binary file to be downloaded
`length`: The number of bytes to be downloaded.

Return value:

Return the result by one of BroadBee_HostAccessResult_t values.

6.2. IEEE Addresses

BroadBee has been implementing Dual Stack software for ZigBee PRO and RF4CE software stacks, and ideally we need two different IEEE addresses; one for ZigBee-PRO stack and another for RF4CE stack. These 64-bit MAC addresses should be assigned by Host system and sent to ZigBee using the API's defined on this section. These two IEEE Addresses are saved into the non-volatile memory on the host file system by BroadBee stack, and there is no need to set these after these are configured properly during the system configuration.

6.2.1. Functions

6.2.1.1. ZBPRO_MAC_SetReq()

Brief description:

Set the IEEE address for ZigBee-PRO stack.

Prototype:

```
void ZBPRO_MAC_SetReq(MAC_SetReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _MAC_SetReqDescr_t
{
    /* 32-bit data. */
    MAC_SetConfCallback_t *callback; /*!< Entry point of the
confirmation callback function. */
    /* Structured data. */
    MacServiceField_t    service;  /*!< MAC requests service field. */
    MAC_SetReqParams_t   params;   /*!< Request parameters structured
object. */
};
```

Callback type:

```
typedef void MAC_SetConfCallback_t(MAC_SetReqDescr_t *const reqDescr,
MAC_SetConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _MAC_SetReqParams_t
{
    /* 64-bit data. */
    MAC_PibAttributeValue_t  attributeValue;        /*!< The value to
write to the indicated PIB attribute. */
    /* 32-bit data. */
    SYS_DataPointer_t        payload;               /*!< The value of
attribute with variable data size. */
    /* 8-bit data. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute to write. */
    /* TODO: This field is redundant. Wrap it with a conditional build
key. */
```

```
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table of the specified PIB attribute to
write. */
} MAC_SetReqParams_t;
```

Callback Parameters type:

```
typedef struct _MAC_SetConfParams_t
{
    /* 8-bit data. */
    MAC_Status_t             status;                /*!< The result of
the request to write the PIB attribute. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute that was written. */
    /* TODO: This field is redundant. Wrap it with a conditional build
key. */
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table of the specified PIB attribute to write. */
} MAC_SetConfParams_t;
```

6.2.1.2. ZBPRO_MAC_GetReq()

Brief description:

Get the IEEE address for ZigBee-PRO stack.

Prototype:

```
void ZBPRO_MAC_GetReq(MAC_GetReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _MAC_GetReqDescr_t
{
    /* 32-bit data. */
    MAC_GetConfCallback_t *callback; /*!< Entry point of the
confirmation callback function. */
    /* Structured data. */
    MacServiceField_t      service; /*!< MAC requests service field. */
    MAC_GetReqParams_t params; /*!< Request parameters structured
object. */
};
```

Callback type:

```
typedef void MAC_GetConfCallback_t(MAC_GetReqDescr_t *const reqDescr,
MAC_GetConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _MAC_GetReqParams_t
{
    /* 8-bit data. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute to read. */
```

```
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table of the specified PIB attribute to read. */
} MAC_GetReqParams_t;
```

Callback Parameters type:

```
typedef struct _MAC_GetConfParams_t
{
    /* 64-bit data. */
    MAC_PibAttributeValue_t  attributeValue;        /*!< The value of
the indicated PIB attribute that was read. */
    /* 32-bit data. */
    SYS_DataPointer_t        payload;               /*!< The value of
attribute with variable data size. */
    /* 8-bit data. */
    MAC_Status_t             status;                /*!< The result of
the request for PIB attribute information. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute that was read. */
    /* TODO: This field is redundant. Wrap it with a conditional build
key. */
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table or array of the specified PIB attribute to read. */
} MAC_GetConfParams_t;
```

6.2.1.3. RF4CE_MAC_SetReq()

Brief description:

Set the IEEE address for RF4CE stack. Initiates asynchronous procedure to set appropriate NIB attribute.

Prototype:

```
void RF4CE_MAC_SetReq(MAC_SetReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _MAC_SetReqDescr_t
{
    /* 32-bit data. */
    MAC_SetConfCallback_t *callback;        /*!< Entry point of the
confirmation callback function. */
    /* Structured data. */
    MacServiceField_t      service; /*!< MAC requests service field. */
    MAC_SetReqParams_t     params;  /*!< Request parameters structured
object. */
};
```

Callback type:

```
void MAC_SetConfCallback_t(MAC_SetReqDescr_t *const reqDescr,
MAC_SetConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _MAC_SetReqParams_t
```

```
{
    /* 64-bit data. */
    MAC_PibAttributeValue_t  attributeValue;        /*!< The value to
write to the indicated PIB attribute. */
    /* 32-bit data. */
    SYS_DataPointer_t        payload;               /*!< The value of
attribute with variable data size. */
    /* 8-bit data. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute to write. */
    /* TODO: This field is redundant. Wrap it with a conditional build
key. */
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table of the specified PIB attribute to write. */
} MAC_SetReqParams_t;
```

Callback Parameters type:

```
typedef struct _MAC_SetConfParams_t
{
    /* 8-bit data. */
    MAC_Status_t             status;                /*!< The result of
the request to write the PIB attribute. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute that was written. */
    /* TODO: This field is redundant. Wrap it with a conditional build
key. */
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table of the specified PIB attribute to write. */
} MAC_SetConfParams_t;
```

Return value:

None

6.2.1.4. RF4CE_MAC_GetReq()

Brief description:

Get the IEEE address for RF4CE stack. Accepts MLME-GET.request for the RF4CE context of the MAC.

Prototype:

```
void RF4CE_MAC_GetReq(MAC_GetReqDescr_t *const reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
struct _MAC_GetReqDescr_t
{
    /* 32-bit data. */
    MAC_GetConfCallback_t *callback;        /*!< Entry point of the
confirmation callback function. */
    /* Structured data. */
    MacServiceField_t      service; /*!< MAC requests service field. */
    MAC_GetReqParams_t     params;  /*!< Request parameters structured
object. */
};
```

Callback type:

```
void MAC_GetConfCallback_t(MAC_GetReqDescr_t *const reqDescr,
MAC_GetConfParams_t *const confParams);
```

Parameters type:

```
typedef struct _MAC_GetReqParams_t
{
    /* 8-bit data. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute to read. */
    /* TODO: This field is redundant. Wrap it with a conditional build
key. */
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table of the specified PIB attribute to read. */
} MAC_GetReqParams_t;
```

Callback Parameters type:

```
typedef struct _MAC_GetConfParams_t
{
    /* 64-bit data. */
    MAC_PibAttributeValue_t  attributeValue;        /*!< The value of
the indicated PIB attribute that was read. */
    /* 32-bit data. */
    SYS_DataPointer_t        payload;               /*!< The value of
attribute with variable data size. */
    /* 8-bit data. */
    MAC_Status_t             status;                /*!< The result of
the request for PIB attribute information. */
    MAC_PibAttributeId_t     attribute;             /*!< The identifier
of the PIB attribute that was read. */
    /* TODO: This field is redundant. Wrap it with a conditional build
key. */
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table or array of the specified PIB attribute to read. */
} MAC_GetConfParams_t;
```

Return value:

None

6.3. BroadBee Files in Host

BroadBee needs to save and retrieve the network and other system information in non-volatile memory for the power loss cases. Since the ZigBee hardware doesn't have own non-volatile memory, BroadBee needs to borrow the memory space from the host system. The actual data structure in the non-volatile memory will have a version number for compatibility in case of future upgrades.

6.3.1. Functions

BroadBee stack can read a file from NVM by "NVM_ReadFileInd()". The result shall be returned through a call back function.
BroadBee can write or update a file in two different ways.
For a simple update or generation of a file, BroadBee will call "NVM_WriteFileInd()", and Host should generate, update, write a temporaty file into NVM, and rename the temporary file to the final filename in NVM.
To update multiple contents of a file, BrodBee will call "NVM_OpenFileInd()", and call "NVM_WriteFileInd()" multiple times, then call "NVM_CloseFileInd()". Host should generate a temporary file for "NVM_OpenFileInd()", and update and write the temporary file for each "NVM_WriteFileInd()", then rename the temporary file to the final filename for the "NVM_CloseFileInd()". With this method, the multiple writings to the final file can be atomic.

6.3.1.1. NVM_ReadFileInd()

Brief description:

Read File indication. Function is to read data from files in Host Non-volatile memory.

Prototype:

```
void NVM_ReadFileInd(NVM_ReadFileIndDescr_t *indDescr);
```

Where `indDescr` pointer to the indication parameters.

Descriptor:

```
typedef struct _NVM_ReadFileIndDescr_t
{
  struct
    {
        SYS_QueueElement_t          next;
    } service;
    NVM_ReadFileIndParams_t params;
    NVM_ReadFileResp_t      callback;
} NVM_ReadFileIndDescr_t;
```

Parameters type:

```
typedef struct _NVM_ReadFileIndParams_t
{
    uint32_t            fileIndex;
    uint32_t            address;
    uint32_t            length;
} NVM_ReadFileIndParams_t;
```

Return value:

```
None
```

6.3.1.2. NVM_OpenFileInd()

Brief description:

This function is to open file placed in Host Non-volatile memory for writing.
Host application should create a temporary copy of the file and all following Write File indications (addressed to the same file index) will modify only this temporary copy.

Prototype:

```
void NVM_ReadFileInd(NVM_ReadFileIndDescr_t *indDescr);
```

Where `indDescr` pointer to the indication parameters.

Descriptor:

```
typedef struct _NVM_ReadFileIndDescr_t
{
  struct
    {
        SYS_QueueElement_t          next;
    } service;
    NVM_ReadFileIndParams_t params;
    NVM_ReadFileResp_t      callback;
} NVM_ReadFileIndDescr_t;
```

Parameters type:

```
typedef struct _NVM_ReadFileIndParams_t
{
    uint32_t            fileIndex;
    uint32_t            address;
    uint32_t            length;
} NVM_ReadFileIndParams_t;
```

Return value:

```
None
```

6.3.1.3. NVM_WriteFileInd()

Brief description:

This function is to write data into a file in Host.
Standalone Write File indication will cause atomic updating of the file but if it follows the Open File indication Host application should modify only temporary copy of the file.

Prototype:

```
void NVM_WriteFileInd(NVM_WriteFileIndDescr_t *indDescr);
```

Where `indDescr` pointer to the indication parameters.

Descriptor:

```
typedef struct _NVM_WriteFileIndDescr_t
{
    struct
    {
        SYS_QueueElement_t          next;
    } service;
    NVM_WriteFileIndParams_t  params;
    NVM_WriteFileResp_t       callback;
} NVM_WriteFileIndDescr_t
```

Parameters type:

```
typedef struct _NVM_WriteFileIndParams_t
{
    uint32_t            fileIndex;
    uint32_t            address;
    SYS_DataPointer_t   payload;
} NVM_WriteFileIndParams_t;
```

Return value:

None

6.3.1.4. NVM_CloseFileInd()

Brief description:

This function is to close file writing session. Host application should replace the file with the updated temporary copy which was made during Open File operation.

Prototype:

`void NVM_CloseFileInd(NVM_CloseFileIndDescr_t *indDescr);`
Where `indDescr` pointer to the indication parameters.

Descriptor:

```
typedef struct _NVM_CloseFileIndDescr_t
{
    struct
    {
        SYS_QueueElement_t          next;
    } service;
    NVM_CloseFileIndParams_t  params;
    NVM_CloseFileResp_t       callback;
} NVM_CloseFileIndDescr_t;
```

Parameters type:

```
typedef struct _NVM_CloseFileIndParams_t
{
    uint32_t             fileIndex;
} NVM_CloseFileIndParams_t;
```

Return value:
None

6.4. OTP Access

ZigBee hardware doesn't have own OTP, but has the allocation of 20 bits from the OTP of Host system to save analog parameters from ATE for the better system performance. Once after the power-on-reset, ZigBee hardware will read this 20-bit OTP data and save into a ZigBee own register, MAC_OTP_CAL_DATA.OTP_CAL_DAT[19:0] that is read-only.

BroadBee will use this register and will not need to access the OTP on Host side for the ZigBee analog circuitry calibration.

As result, there is nothing to do by Host side for this for the normal operation.

6.5. Watch Dog Timer Interrupt

When a Watch Dog situation happens in ZigBee hardware, BroadBee stack doesn't believe the software operation as well as the ZigBee hardware, and hardware interrupt to Host system will be triggered. The ZigBee hardware triggers this interrupt as well as the wake up interrupt to Host if it is in the sleep mode. When Host system receives this interrupt, it should follow through the process described in "7.1 Software Download & Start ZigBee CPU" section, to reset ZigBee CPU, download the BroadBee software, and restart the ZigBee CPU.

BroadBee will restart and continue the normal operation based on the information saved on the non-volatile memory on the Host side. All important network information will be retrieved from NVM, and there is no need to bind the remote systems that had been bound already before.

6.6. Power Saving Mode on Host

ZigBee hardware is on AON (Always ON) island and it is always alive, but Host could be in the power saving mode. BroadBee stack will check the power status of Host before send any message through the Mailbox hardware. If Host is in the sleep mode, then BroadBee stack should decide if it needs to wake the Host up from the sleep mode or to defer this message until Host is alive.

If the message can be deferred until Host is alive or can be ignored, BroadBee will not send this message. However, if the message needs to be delivered immediately, then

1) BroadBee stack will write the message into the MailBox,
2) BroadBee stack will generate the MBOX_Z2H_FULL_INTR interrupt request to the Host side, then
3) BroadBee will generate WAKE_CPU interrupt to Host.
4) BroadBee will keep this WAKE_CPU interrupt request until MBOX_Z2H_EMPTY_INTR interrupt is received. MBOX_Z2H_EMPTY_INTR interrupt service routine will always clear this WAKE_CPU interrupt request bit.

To remove some race conditions to check the Host power status, BroadBee will set the WAKE_CPU interrupt to Host always regardless the Host power status after set the MBOX_Z2H_FULL_INTR interrupt. Host system should block the WAKE_CPU interrupt while it is alive, and unmake the interrupt and clear interrupt request bit, before it goes to power saving mode.

When the Host is in the power saving mode and BroadBee has received an action code from a remote control, BroadBee checks the HDMI-CEC action code aginst the information on the "wakeUpActionCodeFilter" that is given by the Host application. If the corresponding bit on the "wakeUpActionCodeFilter" is set, then BroadBee will waken the

Host up to send the action code. If the corresponding bit is cleared, then this action code will be ignored.

6.6.1. Functions

6.6.1.1. RF4CE_ZRC_SetWakeUpActionCodeReq()

Brief description:

This function is to set the “wakeUpActionCodeFilter” into the BroadBee stack. BroadBee will keep this value both in stack and NVM to recover in case of power failure.

Prototype:

```
void
RF4CE_ZRC_SetWakeUpActionCodeReq(RF4CE_ZRC_SetWakeUpActionCodeReqDescr_
t *reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _RF4CE_ZRC_SetWakeUpActionCodeReqDescr_t
{
    RF4CE_ZRC_SetWakeUpActionCodeReqParams_t params;
    void (*callback) (RF4CE_ZRC_SetWakeUpActionCodeReqDescr_t *,
RF4CE_ZRC_SetWakeUpActionCodeConfParams_t *);
} RF4CE_ZRC_SetWakeUpActionCodeReqDescr_t;
```

Parameters type:

```
typedef struct _RF4CE_ZRC_SetWakeUpActionCodeReqParams_t
{
    uint8_t
wakeUpActionCodeFilter[RF4CE_WAKE_UP_ACTION_CODE_FILTER_LENGTH];
} RF4CE_ZRC_SetWakeUpActionCodeReqParams_t;
```

Callback Parameters type:

```
typedef struct _RF4CE_ZRC_SetWakeUpActionCodeConfParams_t
{
    uint8_t
wakeUpActionCodeFilter[RF4CE_WAKE_UP_ACTION_CODE_FILTER_LENGTH];
    uint8_t status;
} RF4CE_ZRC_SetWakeUpActionCodeConfParams_t;
```

Return value:

None. Actual return of the “wakeUpActionCodeFilter” value and the status will be returned through a callback function.

6.6.1.2. RF4CE_GetWakeUpActionCodeReq()

Brief description:

This function is to read the “wakeUpActionCodeFilter” from the BroadBee stack.

Prototype:

```
void
RF4CE_ZRC_GetWakeUpActionCodeReq(RF4CE_ZRC_GetWakeUpActionCodeReqDescr_
t *reqDescr);
```

Where `reqDescr` is a pointer to the request descriptor structure.

Descriptor:

```
typedef struct _RF4CE_ZRC_GetWakeUpActionCodeReqDescr_t
{
    void (*callback) (RF4CE_ZRC_GetWakeUpActionCodeReqDescr_t *,
RF4CE_ZRC_GetWakeUpActionCodeConfParams_t *);
} RF4CE_ZRC_GetWakeUpActionCodeReqDescr_t;
```

Parameters type:

```
typedef struct _RF4CE_ZRC_SetWakeUpActionCodeConfParams_t
{
    uint8_t
wakeUpActionCodeFilter[RF4CE_WAKE_UP_ACTION_CODE_FILTER_LENGTH];
    uint8_t status;
} RF4CE_ZRC_SetWakeUpActionCodeConfParams_t;
```

Return value:

None. Actual return of the “wakeUpActionCodeFilter” value and the status will be returned through a callback function.

7. Basic Application Software Guidance

This section is to show how to use the API functions to do some very basic operations on ZigBee-PRO Home Automation profile and RF4CE Remote Control profile. Depending on the customers' applications, it may use many more functions from this document to perform more sophisticate tasks on the ZigBee network.

7.1. RF4CE Remote Control Profile

7.1.1. Form Network

To form a ZigBee RF4CE network, we need to call the function, **RF4CE_StartReq()**. This function scanes all given channels for the specified energy level and perform network discovery to find a clear channel with less noise. Then it forms a ZigBee-RF4CE network with a unique PAN ID which doesn't conflict with the neighbor network. The device acts as the Target of the ZigBee-RF4CE network.

7.1.2. Binding

A remote controller needs to bind with the target system before it could communicate. When a controller wants to bind with a target, it should broadcast the discovery request command to the target. To trigger the binding procedure, **RF4CE_ZRC1_TargetBindReq** should be called for ZRC1.1 and either **RF4CE_ZRC2_EnableBindingReq** or **RF4CE_ZRC2_ButtonBindingReq** could be called for ZRC2.0 depending on the different binding ways. The target's RF4CE stack processes pairing with the controller, informs the application software with a temporary pairing by **RF4CE_PairInd()**, and goes through configuration process for ZRC2.0 addtionally. To for the compatility, the application should always provide the handler for **RF4CE_ZRC2_CheckValidationInd()**, furthermore, it responds by **RF4CE_ZRC2_CheckValidationResp()** with PENDING parameter during the validation process, and with SUCCESS parameter when it has completed successfully. For ZRC1.1, the handler never has the chanse to be called. Then, the controller is permanently bound with the target, and the target system accepts the user control commands from the remote controller. The detail of two different binding methods is shown on the Figure 13 on the next page.

7.1.3. Action Control Command

The bound remote controller sends the RC commands for the pressed key on the remote controller. This is not only for the remote controller to control TV or set-top box, but also HID (Human Input Device) and Home Automation controller as well. When BroadBee receives a user control command, it delivers it to the application software by calling the function, **RF4CE_ZRC1_ControlCommandInd()/RF4CE_ZRC2_ControlCommandInd()**. The application software should respond to the received user control command per the application specific task.

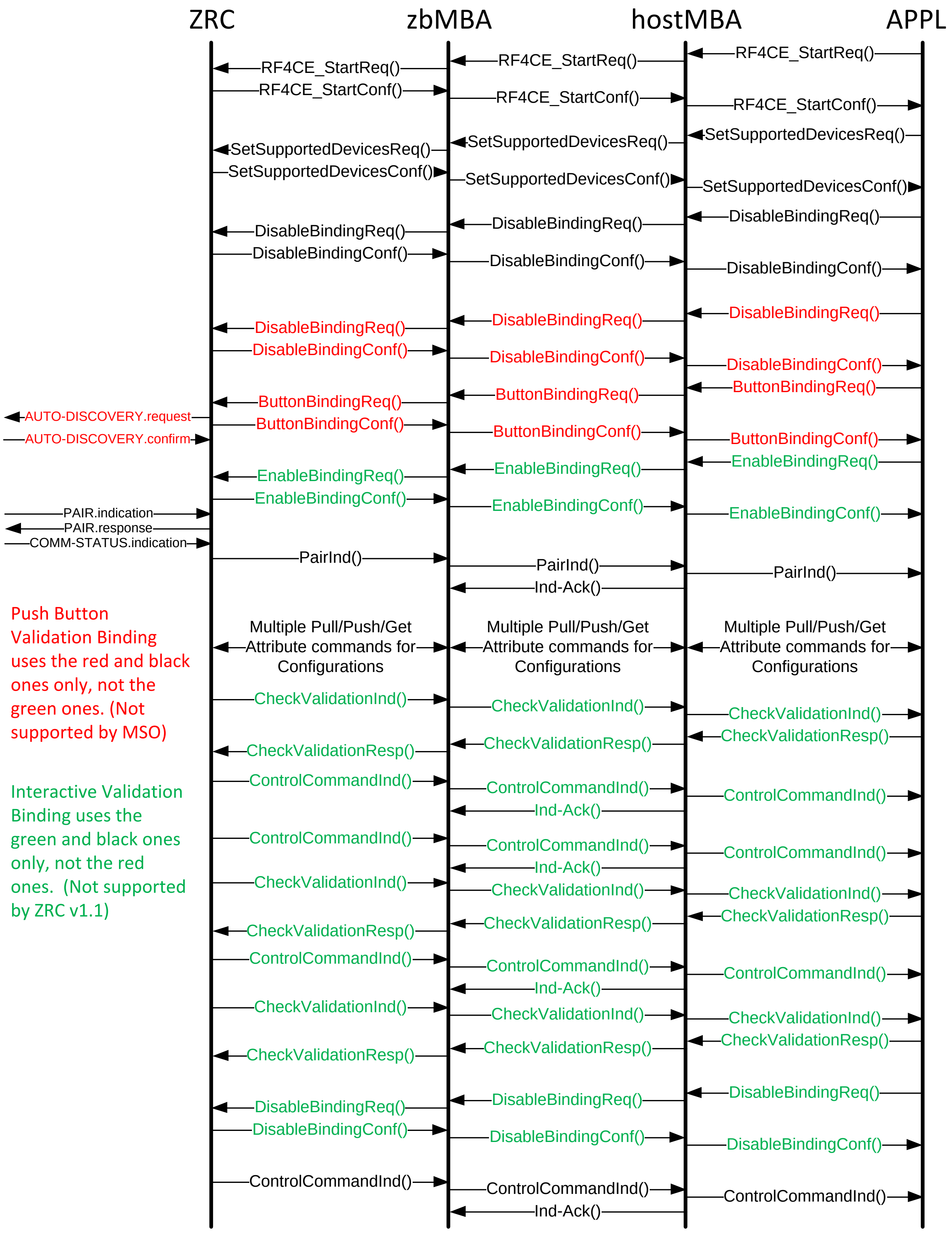


Figure 21: Binding Flow on Host Side

7.2. ZigBee-PRO Home Automation Profile

7.2.1. Form Network

To form a ZigBee-PRO Home Automation network, application software needs to call the function, **ZBPRO_ZDO_StartNetworkReq()**. This function will scan all given channels for the specified energy level and perform network discovery to find a clear channel with less noise. Then it will form a ZigBee-PRO HA network with a unique PAN ID which doesn't conflict with the neighbor networks. The device will act as the Coordinator of the ZigBee-PRO HA network after this function has been called.

7.2.2. Permit Joining

The newly formed Home Automation network should permit joining process so that the other end devices can join this network. There are two ways to achieve this. The first one is to call **ZBPRO_NWK_PermitJoiningReq().** The other one is to set the **ZBPRO_APS_IB_PERMIT_JOIN_DURATION_ID** attribute with the period to allow joining. These two ways are also used to prohibit some devices from joining this network.

7.2.3. Device finding and binding

The set-top-box will have multiple end point devices implemented by BroadBee. Home automation application software should call the function, **ZBPRO_APS_EndpointRegisterReq()** to bind a simple descriptor with each endpoint. BroadBee also provides the EZ-Mode commissioning procedure to configure these end point devices on the set-top-box to operate with other home automation devices. The application software should call **ZBPRO_ZHA_EzModeReq()** to do EZ-Mode network finding and binding in a limited commissioning time. When a home automation end device is also doing commissioning process that can be triggered by an interactive method such as a special button or key pressed on the device, the device can be commissioned successfully by the BroadBee system if the supported profile and in/out clusters match between two devices. Then the application software can communicate with the target home automation end device.

7.2.4. Device Control

After the system has known the network address, endpoint, profile ID and cluster IDs of the target device through the EZ-Mode commissioning, application software can control the target device by sending the corresponding cluster commands. For the Home Automation application, it can send the cluster commands defined in reference [2] and [4] to control the any device that has been certified from ZigBee Home Automation.

For incidences of an on/off end device that supports the on/off cluster of HA profile, the application software can call **ZBPRO_ZCL_OnOffCmdOnReq()**, **ZBPRO_ZCL_OnOffCmdOffReq ()**, or **ZBPRO_ZCL_OnOffCmdToggleReq()** function to turn on, turn off, or toggle the target home automation device.

7.3. Standard Use Cases

Application development requires having a reference on basic workflow sequences. Following use cases show the appropriate workflow sequences for building the most basic ZigBee networks and communicating in them by means of ZHA and ZRC profile commands.

7.3.1. Form ZHA Network and send On/Off Cluster Command to a joined device

This use case is created for the verification of the client and server connectivity action sequence.

It describes the interactions between two devices (Coordinator and Router) where Router joins ZigBee-PRO Home Automation network formed by Coordinator. Coordinator changes the state of Router by sending OnOff cluster action command from Actors.

7.3.1.1. Actors
ZigBee bulb as a network router (Router, R), 3rd party device
ZigBee network coordinator device (Coordinator, C), Broadcom device.

7.3.1.2. List of primitives
ZBPRO_ZDO_StartNetworkReq() [5.1.21.1.1.]
ZBPRO_APS_SetReq() [5.1.21.1.4.]
ZBPRO_ZDO_MatchDescReq() [5.1.24.1.1.1.1.]
ZBPRO_APS_EndpointRegisterReq() [5.1.5.1.1.]
ZBPRO_APS_BindReq() [5.1.22.1.1.1.]
ZBPRO_ZCL_OnOffCmdOnReq() [5.1.8.1.1.]

7.3.1.3. Action Sequence
Action sequence for this use case is quite simple, but it should be performed in a strict order. Since there are only two actors in the sequence, the device that acts in the particular step is indicated in parentheses before each step Brief description. (C for Coordinator, and R for Router).

1. (C) Application on Coordinator side calls **ZBPRO_APS_EndpointRegisterReq()** function to add information about OnOff cluster to device stack. Make sure to indicate all required parameters:
    `deviceId` = *0x07 (COMBINED_INTERFACE)*
    `profileId` = *0x0104 (ZHA)*
    `endpoint` = *0x01*
    `deviceVersion` = *0x00*
    `inClusterAmount` = *0x03*
    `outClusterAmount` = *0x0B*
    `useInternalHandler` = *0x01*
    `inClusterList` = *[0(Basic), 3(Identify), 1281(IasAce)]*

`outClusterList` *= [3(Identify), 4(Groups), 5(Scenes), 6(OnOff), 8(LevelControl), 257(DoorLock), 258(WindowCovering), 768(ColorControl), 1280(IasZone), 1282(IasWd)]*

Coordinator device responds to Applicatioin through the **ZBPRO_APS_EndpointRegisterReq()** callback function with corresponding status.

2. (C) Application software on the Coordinator side calls **ZBPRO_APS_SetReq()** function to set following parameters on Coordinator device:

    `MAC_EXTENDED_ADDRESS =` *0xAAaaAAaaAAaaAAaa (example)*

    `ZBPRO_NWK_DEVICE_TYPE =` *ZBPRO_DEVICE_TYPE_COORDINATOR*

    `ZBPRO_APS_CHANNEL_MASK =` *(1 << 15) (example)*

    `ZBPRO_APS_TRUST_CENTER_ADDRESS =` *MAC_EXTENDED_ADDRESS*

    `ZBPRO_APS_PERMIT_JOIN_DURATION =` *0xFF (allow)*

    Coordinator device responds through the **ZBPRO_APS_SetReq()** callback function with corresponding status.
3. (C) Application software on the Coordinator side calls **ZBPRO_NWK_SetKeyReq()** to set network key.
4. (C) Application software on the Coordinator side calls **ZBPRO_APS_SetReq()** function to set following parameters on Coordinator device:

    `ZBPRO_NWK_ACTIVE_KEY_SEQ_NUMBER` *= 0 (example)*

    Coordinator device responds to Applicatioin through the **ZBPRO_APS_SetReq()** callback primitive with corresponding status.
5. (C) Application software on the Coordinator side should call the **ZBPRO_ZDO_StartNetworkReq()** function to form the ZHA network. Coordinator device responds to Applicatioin through the **ZBPRO_ZDO_StartNetworkReq()** callback primitive with corresponding status.
6. (R) Light bulb enters the network with association frame sequence. Coordinator sends Transport key frame OTA. Stack on Coordinator side returns New child event (with **ZBPRO_NEW_CHILD_EID() id)** indication to Application to mark the new child device entering the network. This event is for the test only, and could be ignored by Application software.
7. (C) Application software on the Coordinator side should perform Service discovery procedure by calling **ZBPRO_ZDO_MatchDescReq()** function with following parameters:

    `zdpDstAddress.address` *= 0xFFFD*

    `zdpDstAddress.addressMode` *= 0x02*

    `nwkAddrOfInterest` *= 0xFFFD*

    `profileId` *= 0x0104*

    `numInClusters` *= 0x03*

    `numOutClusters` *= 0x00*

    `inOutClusterList` *= [6, 8, 768]: — list includes input Clusters Ids from the beginning and output Clusters Ids after that (OnOff cluster). Where inClusterList = [6, 8, 768] and outClusterList = [].*

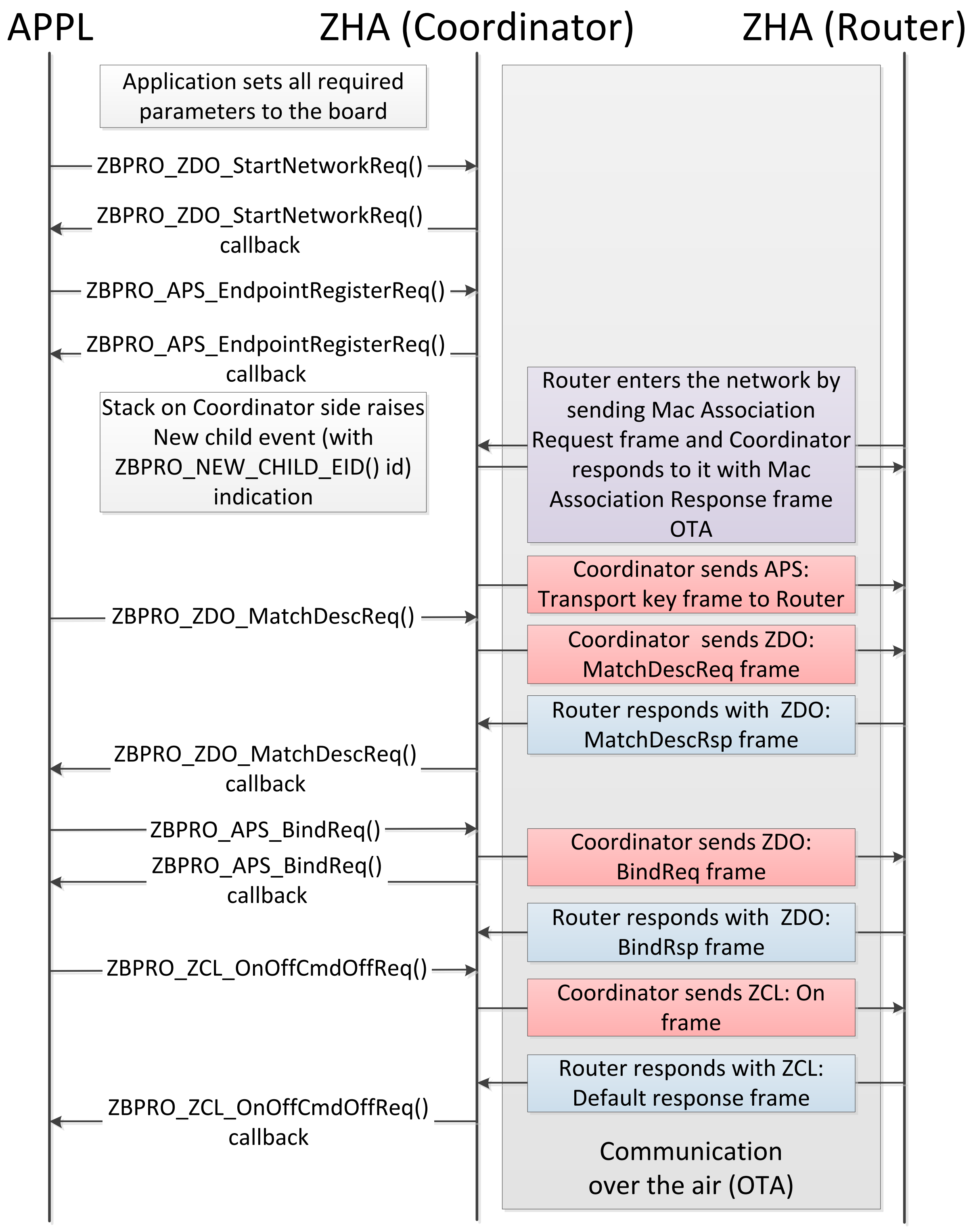
    `respWaitTimeout` *= b'\x00\x00\x00\x00': — response waiting timeout, in milliseconds. Zero means 'Use default ZDO timeout'.*

8. (C) Upon receiving match descriptor response frame OTA from Router, Coordinator device responds to Applicatioin through the **ZBPRO_ZDO_MatchDescReq()** callback function with following parameters:
   `status`: — *request status*
   `responseList`: — *list of received responses as sequence of ZBPRO_ZDO_MatchDescRespListItem_t elements. Contains network addresses of responded Router devices and their matching endpoints.*
9. (C) Application on Coordinator side calls **ZBPRO_APS_BindReq()** referring to OnOff cluster ID with following parameters.
   `srcAddress:` — *coordinator IEEE address*
   `dstAddr:` — *router NWK address*
   NOTE that after Match_desc we have only network address of the end device
   `clusterId:` — *ID of the On/Off cluster*
   `srcEndpoint:` — *coordinator endpoint for the binding entry*
   `dstEndpoint:` — *router endpoint for the binding entry*

   APS Binding allows to preset the local binding table with address, endpoint and cluster information of desired addresse. Application fills “`dstAddr`” with Router address from step 8.
10. (C) Upon receiving binding response frame OTA from Router, Coordinator device responds to Applicatioin through the **ZBPRO_APS_BindReq()** callback function with the “status” parameter:
11. (C) Application on Coordinator side calls **ZBPRO_ZCL_OnOffCmdOnReq()** to send an “On” action command to Router with following parameters:
   `disableDefaultResp` = *0x00*
   `localEndpoint` = *0x01*
   `overallStatus` = *0x00*
   `remoteApsAddress` = *0x00*
   `remoteEndpoint` = *0x0C*
   `respWaitTimeout` = *0x00*
12. (C) Upon receiving ZCL OnOff response frame OTA from Router, Coordinator evice responds to Applicatioin through the **ZBPRO_ZCL_OnOffCmdOnReq()** callback function with “`overallStatus`” parameter which is the status of the On/Off command transaction.

7.3.1.4. Sequence Diagram

Please note that APS Bind request in the below scheme does not require over the Air response, having the callback called instantly without waiting for response from the application.
Please note that the stack responses with the default response frame vs OnOff response frame on receiving the control command frames.


APPL
ZHA (Coordinator)
ZHA (Router)
Application sets all required parameters to the board
ZBPRO_ZDO_StartNetworkReq()
ZBPRO_ZDO_StartNetworkReq() callback
ZBPRO_APS_EndpointRegisterReq()
ZBPRO_APS_EndpointRegisterReq() callback
Stack on Coordinator side raises New child event (with ZBPRO_NEW_CHILD_EID() id) indication
Router enters the network by sending Mac Association Request frame and Coordinator responds to it with Mac Association Response frame OTA
Coordinator sends APS: Transport key frame to Router
ZBPRO_ZDO_MatchDescReq()
Coordinator sends ZDO: MatchDescReq frame
Router responds with ZDO: MatchDescRsp frame
ZBPRO_ZDO_MatchDescReq() callback
ZBPRO_APS_BindReq()
ZBPRO_APS_BindReq() callback
Coordinator sends ZDO: BindReq frame
Router responds with ZDO: BindRsp frame
ZBPRO_ZCL_OnOffCmdOffReq()
Coordinator sends ZCL: On frame
Router responds with ZCL: Default response frame
ZBPRO_ZCL_OnOffCmdOffReq() callback
Communication over the air (OTA)

7.3.2. Form ZHA Network with EZMode binding and send On/Off Cluster Command

This use case serves for the verification of the client and server connectivity action sequence.

It describes the interactions between two devices (Coordinator and Router) forming ZigBee-PRO Home Automation network and changing the state of Test harness by sending OnOff cluster action commands from DUT. Binding process utilizes the ZHA EZmode binding procedure.

7.3.2.1. Actors

ZigBee bulb as a network router (Router, R), 3rd party device
ZigBee network coordinator device (Coordinator, C)

7.3.2.2. List of primitives

ZBPRO_APS_SetReq() [5.1.21.1.4.]
ZBPRO_NWK_SetKeyReq() [4.2.1.1.1.1.]
ZBPRO_ZHA_EzModeReq() [5.1.21.1.2.]
ZBPRO_ZCL_OnOffCmdOffReq() [5.1.8.1.1.]

7.3.2.3. Action Sequence

Action sequence for this use case is quite simple, but it should be performed in a strict order. Since there are only two actors in the sequence, the device that acts in the particular step is indicated in parentheses before each step Brief description. (C for Coordinator, and R for Router).

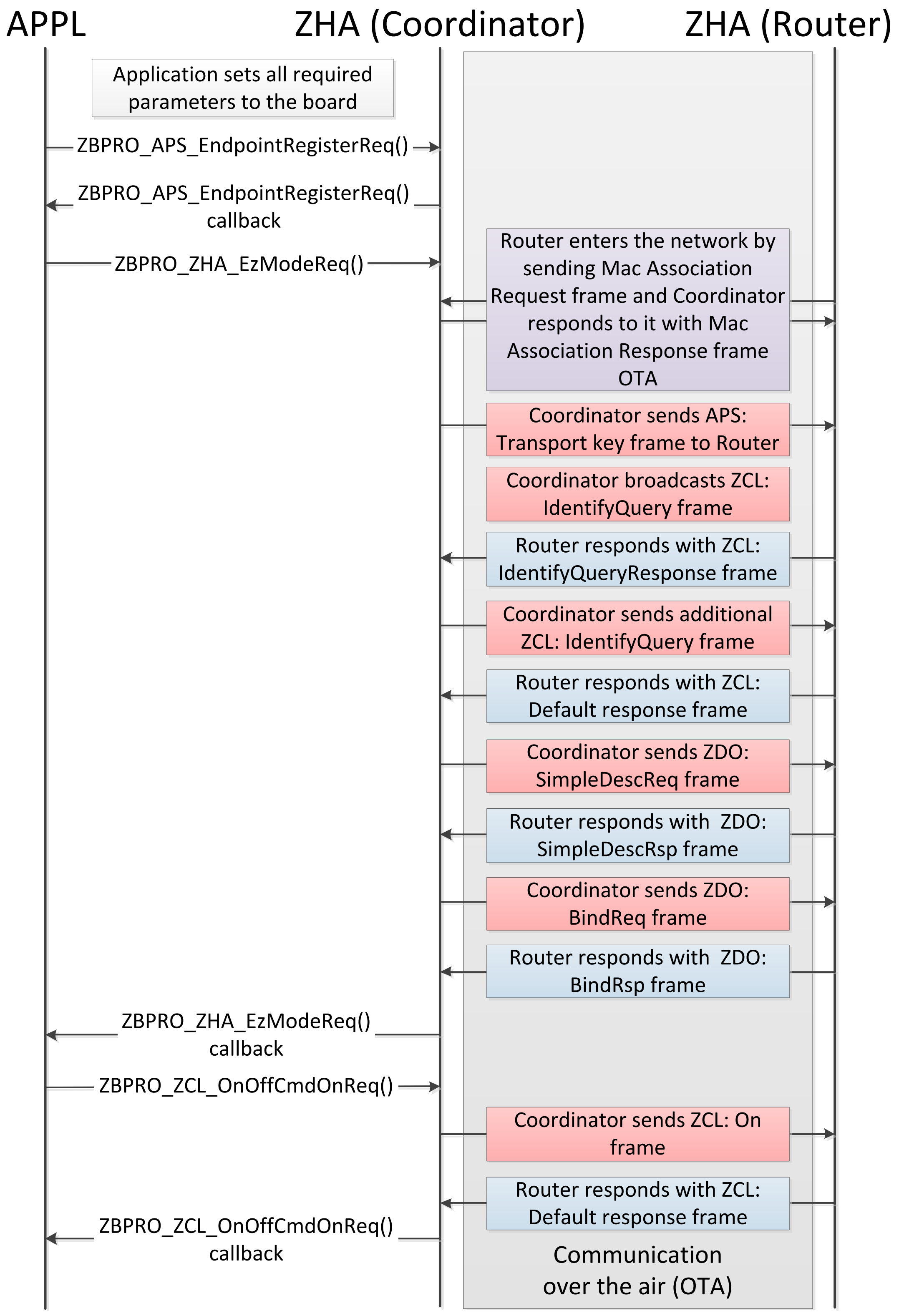
1. (C) Application software on the Coordinator side calls **ZBPRO_APS_SetReq()** function to set following parameters on Coordinator device:
   `MAC_EXTENDED_ADDRESS` = *0xAAaaAAaaAAaaAAaa (example)*
   `ZBPRO_APS_CHANNEL_MASK` *= (1 << 15) (example)*
   `ZBPRO_NWK_DEVICE_TYPE` = *ZBPRO_DEVICE_TYPE_COORDINATOR*
   `ZBPRO_APS_TRUST_CENTER_ADDRESS` = *MAC_EXTENDED_ADDRESS*
   `ZBPRO_APS_PERMIT_JOIN_DURATION` *= 0xFF(allow)*
2. (C) Application software on the Coordinator side calls **ZBPRO_NWK_SetKeyReq()** to set network key.
3. (C) Application on Coordinator side calls **ZBPRO_APS_EndpointRegisterReq()** function to add information about OnOff cluster to the device stack. Make sure to indicate all required parameters:
   `deviceId` *= 0x07 (COMBINED_INTERFACE)*
   `profileId` *= 0x0104 (ZHA)*
   `endpoint` *= 0x01*
   `deviceVersion` *= 0x00*
   `inClusterAmount` *= 0x03*
   `outClusterAmount` *= 0x0B*
   `useInternalHandler` *= 0x01*
   `inClusterList` *= [0(Basic), 3(Identify), 1281(IasAce)]*

`outClusterList` *= [3(Identify), 4(Groups), 5(Scenes), 6(OnOff), 8(LevelControl), 257(DoorLock), 258(WindowCovering), 768(ColorControl), 1280(IasZone), 1282(IasWd)]*

Coordinator device responds through the **ZBPRO_APS_EndpointRegisterReq()** callback function with corresponding status.

4. (R) Bulb starts EZ mode procedure as Target.
5. (C) Application on Coordinator side calls **ZBPRO_ZHA_EzModeReq()** function as initiator to form ZHA network with following parameters:
   `roundTimeMs` *= 10000*
   `times` *= 1*
   `ezRole` *= 0*
   `factoryFresh` *= 0*
   `endpoint` *= 1 (example)*
6. (C) The stack on Coordinator side responds through the **ZBPRO_ZHA_EzModeReq()** callback function with "`status`" parameters which is the status of the transaction.
7. (C) Application Coordinator side calls **ZBPRO_ZCL_OnOffCmdOnReq()** to send "On" action command to Router. Make sure to indicate values for all following parameters:
   `disableDefaultResp` *= 0x00*
   `localEndpoint` *= 0x01*
   `overallStatus` *= 0x00*
   `remoteApsAddress` *= 0x00*
   `remoteEndpoint` *= 0x0C*
   `respWaitTimeout` *= 0x00*
8. (C) Upon receiving ZCL OnOff response from Router, the stack on Coordinator side responds through the **ZBPRO_ZCL_OnOffCmdOnReq()** callback function with "`overallStatus`" parameter which is the status of the transaction.

7.3.2.4. Sequence Diagram


APPL
ZHA (Coordinator)
ZHA (Router)
Application sets all required parameters to the board
ZBPRO_APS_EndpointRegisterReq()
ZBPRO_APS_EndpointRegisterReq() callback
ZBPRO_ZHA_EzModeReq()
Router enters the network by sending Mac Association Request frame and Coordinator responds to it with Mac Association Response frame OTA
Coordinator sends APS: Transport key frame to Router
Coordinator broadcasts ZCL: IdentifyQuery frame
Router responds with ZCL: IdentifyQueryResponse frame
Coordinator sends additional ZCL: IdentifyQuery frame
Router responds with ZCL: Default response frame
Coordinator sends ZDO: SimpleDescReq frame
Router responds with ZDO: SimpleDescRsp frame
Coordinator sends ZDO: BindReq frame
Router responds with ZDO: BindRsp frame
ZBPRO_ZHA_EzModeReq() callback
ZBPRO_ZCL_OnOffCmdOnReq()
Coordinator sends ZCL: On frame
Router responds with ZCL: Default response frame
ZBPRO_ZCL_OnOffCmdOnReq() callback
Communication over the air (OTA)

7.3.3. Form ZRC 2.0 network with Push button pairing and send Action command

This use case describes ZRC 2.0 binding sequence.

It describes the interactions between two devices (Controller and Target) forming ZRC network, binding two devices with push button stimulus and sending an action command from Controller to Target.

7.3.3.1. Actors
ZRC recipient device, TV (Target, T)
ZRC Originator (Controller, C)

7.3.3.2. List of primitives
RF4CE_NWK_SetReq() [4.1.1.1.2.18.]
RF4CE_MAC_SetReq() [6.2.1.3.]
RF4CE_ZRC2_SetAttributesReq() [4.1.1.1.2.8.]
RF4CE_StartReq() [4.1.1.1.2.14.]
RF4CE_ZRC2_BindReq() [4.1.1.1.2.17.]
RF4CE_ZRC2_EnableBindingReq() [4.1.1.1.4.5.]
RF4CE_ZRC2_SetPushButtonStimulusReq() [4.1.1.1.4.12.]
RF4CE_ZRC2_PairNtfyInd() [4.1.1.1.4.10.]
RF4CE_ZRC2_ControlCommandPressedReq() [4.1.1.1.2.11.]
RF4CE_ZRC2_ControlCommandReleasedReq() [4.1.1.1.2.12.]
RF4CE_ZRC2_ControlCommandInd() [4.1.1.1.4.8.]

7.3.3.3. Action Sequence
Action sequence for this use case involves two actors and utilizing several RF4CE primitives. This use case shows the interaction process between Controller (Originator) and Target TV device where controller changes Target TV state by sendind the Action command.

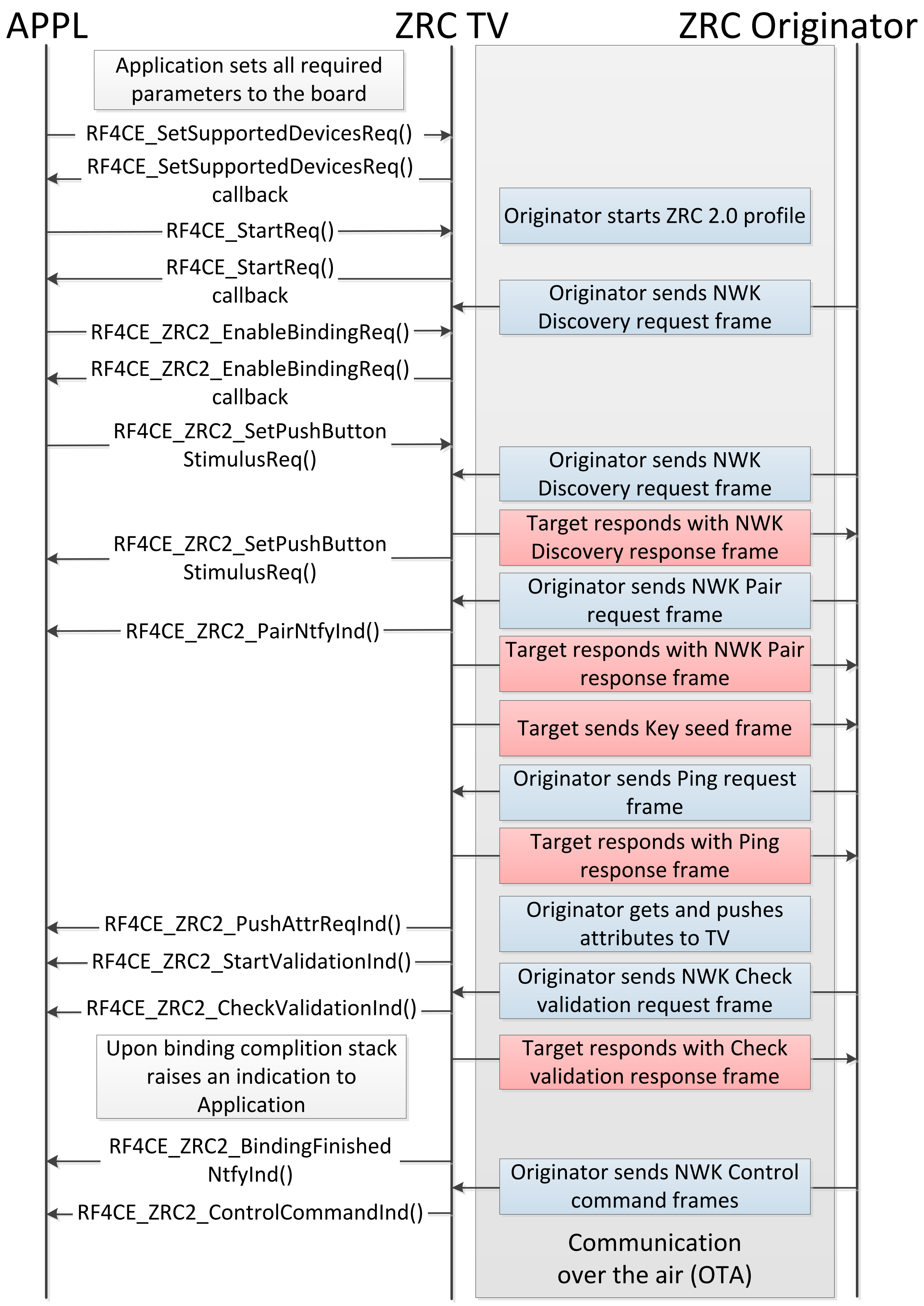
1. (T) Application software on the Target side calls **RF4CE_MAC_SetReq()** function to set following parameters on Target device:
   `MAC_EXTENDED_ADDRESS` = *0xAAaaAAaaAAaaAAaa (example)*
2. (T) Target stack responds through the **RF4CE_MAC_SetReq()** callback function with following parameters:
   `status:` — *result of the request to write the PIB attribute*
   `attribute:` — *identifier of the PIB attribute that was written*
3. (T) Application software on the Target side calls **RF4CE_NWK_SetReq()** function to set following parameters on Target device:
   `RF4CE_NWK_CHANNEL_NORMALIZATION_CAPABLE` = *0x01*
   `RF4CE_NWK_SECURITY_CAPABLE` = *0x01*
   `RF4CE_NWK_POWER_SOURCE` = *0x00*
4. (T) Target stack responds through the **RF4CE_NWK_SetReq()** callback function with following parameters:
   `status:` — *result of the request to write the NIB attribute*
   `attrId:` — *NIB attribute identification*

5. (T) Application software on the Target side calls **RF4CE_ZRC2_SetAttributesReq()** function to set following GDP attributes on Target device:
   `RF4CE_ZRC2_APL_ACTION_BANKS_SUPPORTED_TX` = *0xA5*
   `RF4CE_ZRC2_ACTION_CODES_SUPPORTED_TX` = *0xC1*
6. (T) Target stack responds through the **RF4CE_ZRC2_SetAttributesReq()** callback function with following parameters:
   `status:` — *result of the request to write the NIB attribute*
   `attrId:` — *NIB attribute identification*
7. (T) Application on Target side calls **RF4CE_SetSupportedDevicesReq()** function to set device type of the target with following parameters:
   `devices` = *RF4CE_TELEVISION*
   `numDevices` = *0x01*
8. (T) Target stack responds through the **RF4CE_SetSupportedDevicesReq()** callback function to Application with the status of operation.
9. (T) Application software initiates RF4CE profile on Target device by calling **RF4CE_StartReq()** function.
10. (T) Target stack responds through the **RF4CE_StartReq()** callback function.
11. (T) Application on Target side calls **RF4CE_ZRC2_EnableBindingReq()** to enable biding on TV.
12. (T) Target stack responds through the **RF4CE_ZRC2_EnableBindingReq()** callback function.
13. (T) Applcation on Target side calls **RF4CE_ZRC2_SetPushButtonStimulusReq()** function to initiate ZRC binding with Controller. Parameters:
   `autoDiscDuration` = *0x00*
14. (T) Target stack responds through the **RF4CE_ZRC2_SetPushButtonStimulusReq()** callback function with the parameter of `status.`
15. (C) Controller pushes some parameters to Target.
16. (T) Upon receiving pair request frame from Controller, target stack reports **RF4CE_ZRC2_PairNtfyInd()** indication with following partameters:
   `pairingRef:` — *pairing reference for Controller*
   `status:` — *pairing operation status*
17. (T) Upon receiving push frame OTA from Controller and setting required parameters, Target stack reports **RF4CE_ZRC2_PushAttrReqInd()** indication with following parameters:
   `pairingRef:` — *controller pairing reference*
   It also contains payload with Controller parameters.
18. (T) In order to validate the pairing, Trarget stack reports **RF4CE_ZRC2_StartValidationInd()** indication with following parameters:
   `isAutoValidated` *: — automatic validation parameter*
   `pairingRef`*: — pairing reference to Controller*
19. (T) Target stack reports **RF4CE_ZRC2_CheckValidationInd()** indication to Application right after receiving Check Validation Request OTA. Parameters are as follows:
   `pairingRef:` — *pairing reference of the Check Validation request*
   `isAutoValidated:` — *true if automatical validation took place*

20. (T) Upon completing the binding process, Target stack reports **RF4CE_ZRC2_BindingFinishedNtfyInd()** indication with following parameters:
    `pairingRef:` *— controller p[airing reference*
    `profileID:` *— identifier of the profile indicating the format of received data*
    `status:` *— binding status*
21. (C) Controller issues “command press” and “command release” frames functions to send Target the HDMI command and change Target state.
22. (T) Target stack reports to Application through the **RF4CE_ZRC2_ControlCommandInd()** notification containing Controller pairing reference number and payload with actionData.

7.3.3.4. Sequence Diagram

Please note that the stack raises Set Push Button Stimulus request callback only after receiving the Network Discovery request frame from the Originator.
Please note that the stack raises Pairing indication only after receiving the Network Pair request frame from the Originator.


APPL
ZRC TV
ZRC Originator
Application sets all required parameters to the board
RF4CE_SetSupportedDevicesReq()
RF4CE_SetSupportedDevicesReq() callback
RF4CE_StartReq()
RF4CE_StartReq() callback
RF4CE_ZRC2_EnableBindingReq()
RF4CE_ZRC2_EnableBindingReq() callback
RF4CE_ZRC2_SetPushButton StimulusReq()
RF4CE_ZRC2_SetPushButton StimulusReq()
RF4CE_ZRC2_PairNtfyInd()
RF4CE_ZRC2_PushAttrReqInd()
RF4CE_ZRC2_StartValidationInd()
RF4CE_ZRC2_CheckValidationInd()
Upon binding complition stack raises an indication to Application
RF4CE_ZRC2_BindingFinished NtfyInd()
RF4CE_ZRC2_ControlCommandInd()
Originator starts ZRC 2.0 profile
Originator sends NWK Discovery request frame
Originator sends NWK Discovery request frame
Target responds with NWK Discovery response frame
Originator sends NWK Pair request frame
Target responds with NWK Pair response frame
Target sends Key seed frame
Originator sends Ping request frame
Target responds with Ping response frame
Originator gets and pushes attributes to TV
Originator sends NWK Check validation request frame
Target responds with Check validation response frame
Originator sends NWK Control command frames
Communication over the air (OTA)

7.3.4. Form ZRC 2.0 network with two devices using interactive validation pairing and send Action command

This use case describes ZRC 2.0 binding with interactive validation sequence.

It describes the interactions between two devices (Controller and Target) forming ZRC network, binding two devices with initeractive validation and sending an action command from Controller to Target.

7.3.4.1. Actors
ZRC target device, TV (Target, T)
ZRC Originator (Controller, C)

7.3.4.2. List of primitives
RF4CE_NWK_SetReq() [4.1.1.1.2.18.]
RF4CE_MAC_SetReq() [6.2.1.3.]
RF4CE_ZRC2_SetAttributesReq() [4.1.1.1.2.8.]
RF4CE_StartReq() [4.1.1.1.2.14.]
RF4CE_ZRC2_BindReq() [4.1.1.1.2.17.]
RF4CE_ZRC2_EnableBindingReq() [4.1.1.1.4.5.]
RF4CE_ZRC2_PairNtfyInd() [4.1.1.1.4.10.]
RF4CE_ZRC2_StartValidationInd() [4.1.1.1.4.9.]
RF4CE_ZRC2_CheckValidationResp() [4.1.1.1.2.10.]
RF4CE_ZRC2_BindingFinishedNtfyInd() [4.1.1.1.4.11.]
RF4CE_ZRC2_ControlCommandPressedReq() [4.1.1.1.2.11.]
RF4CE_ZRC2_ControlCommandReleasedReq() [4.1.1.1.2.12.]
RF4CE_ZRC2_ControlCommandInd() [4.1.1.1.4.8.]

7.3.4.3. Action Sequence
Action sequence for this use case involves two actors and utilizing several RF4CE primitives. This use case shows the interaction process between Controller (Originator) and Target TV device where controller changes Target TV state by sendind the Action command.

1. (C) Application software on the Controller side calls **RF4CE_MAC_SetReq()** function to set following parameters on Controller device:
   `MAC_EXTENDED_ADDRESS` = *0xAAaaAAaaAAaaAAaa (example)*
2. (C) Controller stack responds through the **RF4CE_MAC_SetReq()** callback function with following parameters:
   `status:` *— result of the request to write the PIB attribute*
   `attribute:` *— identifier of the PIB attribute that was written*
3. (C) Application software on the Controller side calls **RF4CE_NWK_SetReq()** function to set following parameters on Controller device:
   `RF4CE_NWK_CHANNEL_NORMALIZATION_CAPABLE` = *0x01*
   `RF4CE_NWK_SECURITY_CAPABLE` = *0x01*
   `RF4CE_NWK_POWER_SOURCE` = *0x00*
4. (C) Controller stack responds through the **RF4CE_NWK_SetReq()** callback function with following parameters:

`status:` — *result of the request to write the NIB attribute*
`attrId:` — *NIB attribute identification*

5. (C) Application software on the Controller side calls **RF4CE_ZRC2_SetAttributesReq()** function to set following GDP attributes on Controller device:
   `RF4CE_ZRC2_APL_ACTION_BANKS_SUPPORTED_TX` = *0xA5*
   `RF4CE_ZRC2_ACTION_CODES_SUPPORTED_TX` = *0xC1*
   `RF4CE_GDP_APL_AUTO_CHECK_VALIDATION_PERIOD` = *0x87*
6. (C) Controller stack responds through the **RF4CE_ZRC2_SetAttributesReq()** callback function with following parameters:
   `status:` — *result of the request to write the NIB attribute*
   `attrId:` — *NIB attribute identification*
7. (C) Application software initiates RF4CE profile on Controller device by calling **RF4CE_StartReq()** function. Controller stack responds through the **RF4CE_StartReq()** callback function to application with operation status.
8. (T) Target device enables binding to be able to pair with Controller
9. (C) Application software on Controller side calls **RF4CE_ZRC2_BindReq()** function to perform RF4CE binding. Controller stack responses through a **RF4CE_ZRC2_BindReq()** callback function with following parameters expected:
   `status:` — *binding status*
   `pairingRef:` — *pairing reference on binding.*
10. (C) Controller stack reports **RF4CE_ZRC2_PairNtfyInd()** notification to indicate the pairing process result. Indication contains following parameters:
    `pairingRef:` — *controller pairing reference*
    `status:` — *device pairing status*
11. (C) Controller stack reports **RF4CE_ZRC2_GetAttrRespInd()** notification to Application upon getting and setting some target attributes. Parameters are as follows:
    `pairingRef:` — *pairing reference. If it is equal to RF4CE_NWK_INVALID_PAIRING_REF, then local attributes are retrieved*
    `payload:` — *payload must contain an array of aligned RF4CE_ZRC2_SetAttributeId_t records*
12. (C) Controller stack reports **RF4CE_ZRC2_StartValidationInd()** notification with following parameters:
    `pairingRef:` — *pairing reference of the Check Validation request*
    `isAutoValidated:` — *true if automatical validation took place*
13. (C) Application software on Controller side calls a series of **RF4CE_ZRC2_ControlCommandPressedReq()** and **RF4CE_ZRC2_ControlCommandReleasedReq()** with following parameters:
    `pairingRef:` — *target pairing reference.*
    `payload:` — *supplied payload consisting of one or more buttons of type RF4CE_ZRC2_Action_t* to perform interactive validation procedure by sending a required sequence of control commands.

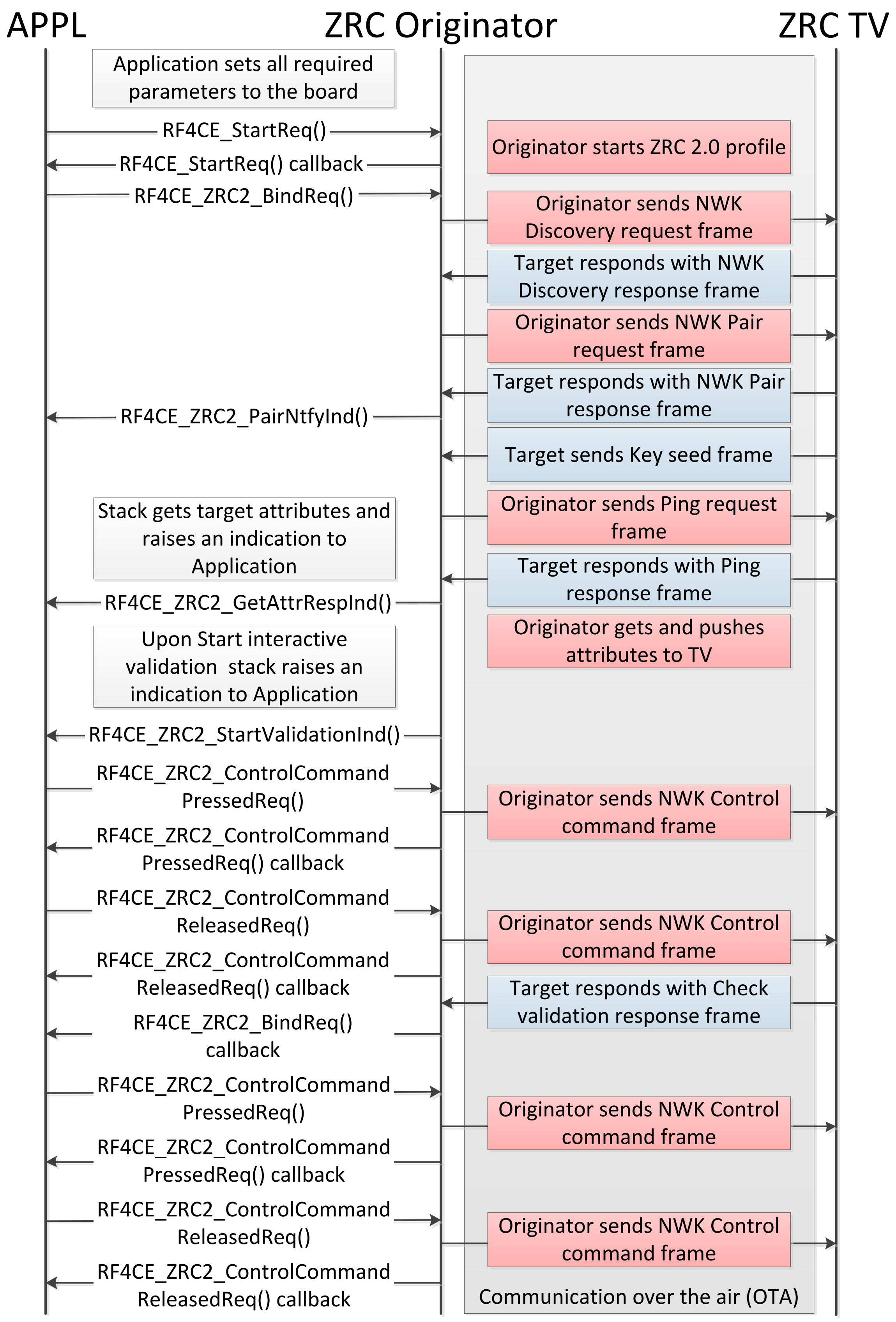
    (C) Controller stack responses through the **RF4CE_ZRC2_ControlCommandPressedReq()** and

RF4CE_ZRC2_ControlCommandReleasedReq() callback functions to Application software.

14. (T) Target responds on pairing validation with a frame.
15. (C) Controller side raises **RF4CE_ZRC2_BindReq()** callback with following parameters:
    `status`: — *binding status. See RF4CE_ZRC2_BindStatus_t*
    `pairingRef`: — *pairing reference on binding*
    `profileId`: — *actual profile*
16. (C) Application on Controller side calls **RF4CE_ZRC2_ControlCommandPressedReq()** function to send Target the HDMI command with "button pressed" parameter. Upon sending the request the application receives the callback with following parameters:
    `pairingRef:` — the *target pairing reference.*
    `payload:` — *supplied payload consisting of one or more RF4CE_ZRC2_Action_t and/or RF4CE_ZRC2_ActionVendor_t structures.*
17. (C) Application on Controller side calls **RF4CE_ZRC2_ControlCommandReleasedReq()** function to send Target the HDMI command with "button released" parameter. Upon sending the request the application receives the callback with following parameters:
    `pairingRef:` — *target pairing reference*
    `payload:` — *supplied payload consisting of one or more RF4CE_ZRC2_Action_t and/or RF4CE_ZRC2_ActionVendor_t structures*.

7.3.4.4. Sequence Diagram

Please note that the stack raises Binding request callback only after getting the check (binding) validation response frame from the end device.


APPL
ZRC Originator
ZRC TV
Application sets all required parameters to the board
RF4CE_StartReq()
RF4CE_StartReq() callback
RF4CE_ZRC2_BindReq()
Originator starts ZRC 2.0 profile
Originator sends NWK Discovery request frame
Target responds with NWK Discovery response frame
Originator sends NWK Pair request frame
Target responds with NWK Pair response frame
RF4CE_ZRC2_PairNtfyInd()
Target sends Key seed frame
Stack gets target attributes and raises an indication to Application
Originator sends Ping request frame
Target responds with Ping response frame
RF4CE_ZRC2_GetAttrRespInd()
Originator gets and pushes attributes to TV
Upon Start interactive validation stack raises an indication to Application
RF4CE_ZRC2_StartValidationInd()
RF4CE_ZRC2_ControlCommand PressedReq()
Originator sends NWK Control command frame
RF4CE_ZRC2_ControlCommand PressedReq() callback
RF4CE_ZRC2_ControlCommand ReleasedReq()
Originator sends NWK Control command frame
RF4CE_ZRC2_ControlCommand ReleasedReq() callback
Target responds with Check validation response frame
RF4CE_ZRC2_BindReq() callback
RF4CE_ZRC2_ControlCommand PressedReq()
Originator sends NWK Control command frame
RF4CE_ZRC2_ControlCommand PressedReq() callback
RF4CE_ZRC2_ControlCommand ReleasedReq()
Originator sends NWK Control command frame
RF4CE_ZRC2_ControlCommand ReleasedReq() callback
Communication over the air (OTA)

7.3.5. Form ZRC 1.1 network with push button pairing sequence and send Action command

This use case describes the interactions between two devices (Controller and Target) forming ZRC network, pairing two devices and sending control command from Controller to Target.

7.3.5.1. Actors

ZRC Target device, TV (Target, T)
ZRC Originator (Controller, C)

7.3.5.2. List of primitives

RF4CE_NWK_SetReq() [4.1.1.1.2.18.]
RF4CE_StartReq() [4.1.1.1.2.14.]
RF4CE_ZRC1_ControllerBindReq() [4.1.1.1.3.1.]
RF4CE_ZRC1_TargetBindReq() [4.1.1.1.4.1.]
RF4CE_PairInd() [4.1.1.1.2.19.]
RF4CE_ZRC1_ControlCommandPressedReq() [4.1.1.1.2.4.]
RF4CE_ZRC1_ControlCommandReleasedReq() [4.1.1.1.2.12.]
RF4CE_ZRC1_ControlCommandInd() [4.1.1.1.4.2.]
RF4CE_SetSupportedDevicesReq() [4.1.1.1.2.16.]

7.3.5.3. Action Sequence

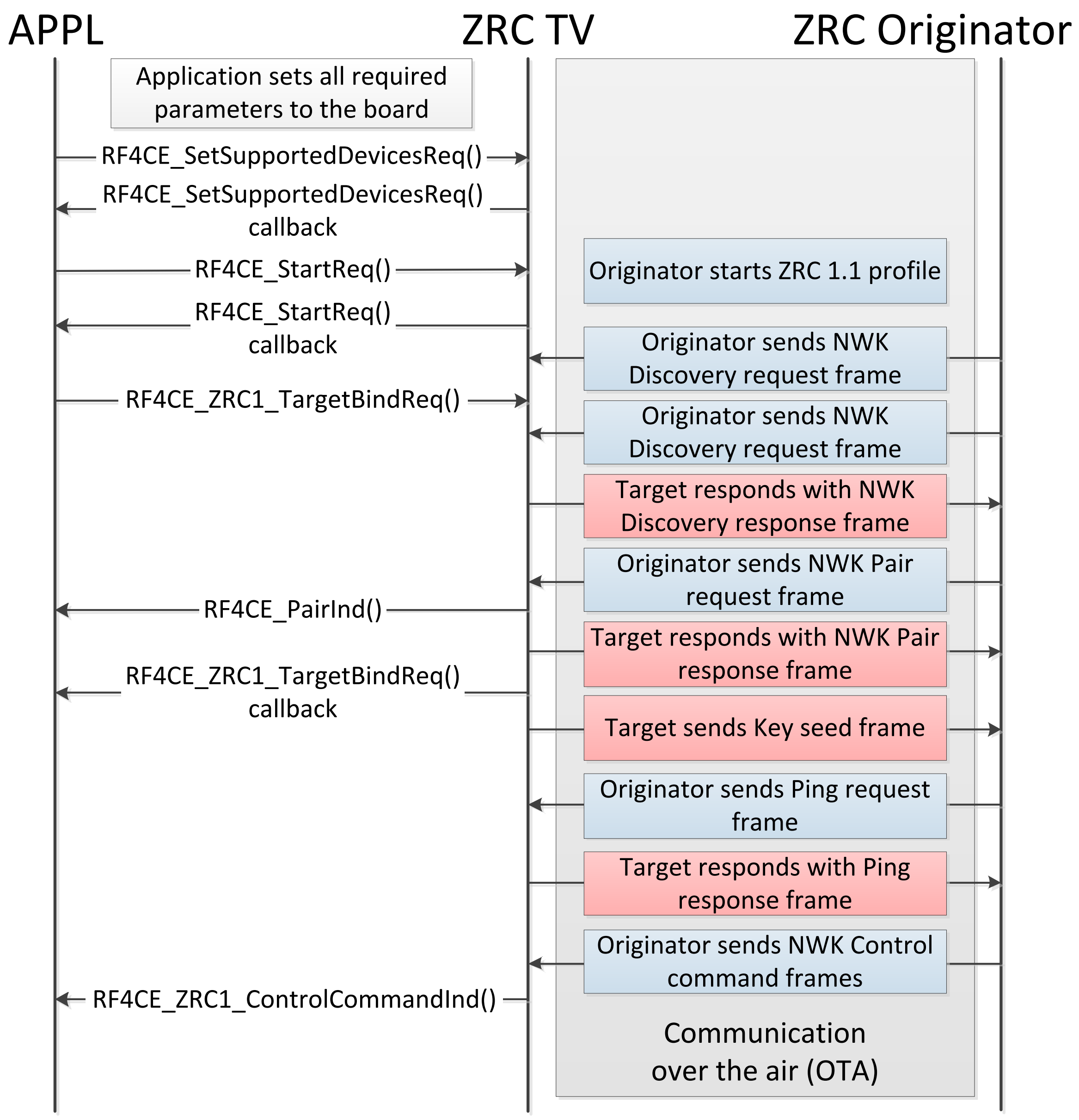
Action sequence for this use case involves two actors and utilizing several RF4CE primitives. This use case shows the interaction process between Controller (Originator) and Target TV device where controller changes Target TV state by sendind the Action command.

1. (T) Application software on the Target side calls **RF4CE_MAC_SetReq()** function to set following parameters on Target device:
   `MAC_EXTENDED_ADDRESS` = *0xAAaaAAaaAAaaAAaa (example)*
2. (T) Target stack response through the **RF4CE_MAC_SetReq()** callback function with following parameters:
   `status:` — *result of the request to write the PIB attribute*
   `attribute:` — *identifier of the PIB attribute that was written*
3. (T) Application software on the Target side calls **RF4CE_NWK_SetReq()** function to set following parameters on Target device:
   `RF4CE_NWK_CHANNEL_NORMALIZATION_CAPABLE` = *0x01*
   `RF4CE_NWK_SECURITY_CAPABLE` = *0x01*
   `RF4CE_NWK_POWER_SOURCE` = *0x00*
   `RF4CE_NWK_DISCOVERY_LQI_THRESHOLD` = *0x62*
4. (T) Target stack responds through the **RF4CE_NWK_SetReq()** callback function with following parameters:
   `status:` — *result of the request to write the NIB attribute*
   `attrId:` — *NIB attribute identification*
5. (T) Application on Target side calls **RF4CE_SetSupportedDevicesReq()** function to set device type of the target with following parameters:

`devices` = *RF4CE_TELEVISION*
`numDevices` = *0x01*

6. (T) Target stack responds through the **RF4CE_SetSupportedDevicesReq()** callback function to Application with following parameter:
   `status:` — *status of the set operation*
7. (T) Application software initiates RF4CE profile on Target device by calling **RF4CE_StartReq()** function. Target stack responds through the **RF4CE_StartReq()** callback function to application with operation status.
8. (T) Application software on Target side calls **RF4CE_ZRC1_TargetBindReq()** function to start pairing procedure.
9. (C) Controller performs the Discovery procedure and starts pairing by issuing pairing request frame.
10. (T) Target stack reports **RF4CE_PairInd()** indication with following parameters:
    `status`: — *status of pairing to controller*
    `pairingRef`: — *controller pairing reference*
11. (T) Target stack responds through the **RF4CE_ZRC1_TargetBindReq()** callback function with following parameters:
    `pairingRef:` — *pairing reference to Controller*
    `status:` — *status of the binding operation*
12. (C) Controller issues “command pressed” and “command released” command frames OTA to send the HDMI command with “button pressed” and “button released” parameters to Target.
13. (T) Target stack reports to application software with **RF4CE_ZRC1_ControlCommandInd()** notification containing actionData and payload with following paratemers:
    `pairingRef`: — *controller pairing reference*
    `flags`: — *RF4CE_ZRC1_USER_CONTROL_PRESSED or RF4CE_ZRC1_USER_CONTROL_REPEATED or RF4CE_ZRC1_USER_CONTROL_RELEASED.*
    `commandCode`: — *sent command code*

7.3.5.4. Sequence Diagram


APPL
ZRC TV
ZRC Originator
Application sets all required parameters to the board
RF4CE_SetSupportedDevicesReq()
RF4CE_SetSupportedDevicesReq() callback
RF4CE_StartReq()
RF4CE_StartReq() callback
RF4CE_ZRC1_TargetBindReq()
RF4CE_PairInd()
RF4CE_ZRC1_TargetBindReq() callback
RF4CE_ZRC1_ControlCommandInd()
Originator starts ZRC 1.1 profile
Originator sends NWK Discovery request frame
Originator sends NWK Discovery request frame
Target responds with NWK Discovery response frame
Originator sends NWK Pair request frame
Target responds with NWK Pair response frame
Target sends Key seed frame
Originator sends Ping request frame
Target responds with Ping response frame
Originator sends NWK Control command frames
Communication over the air (OTA)