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

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.

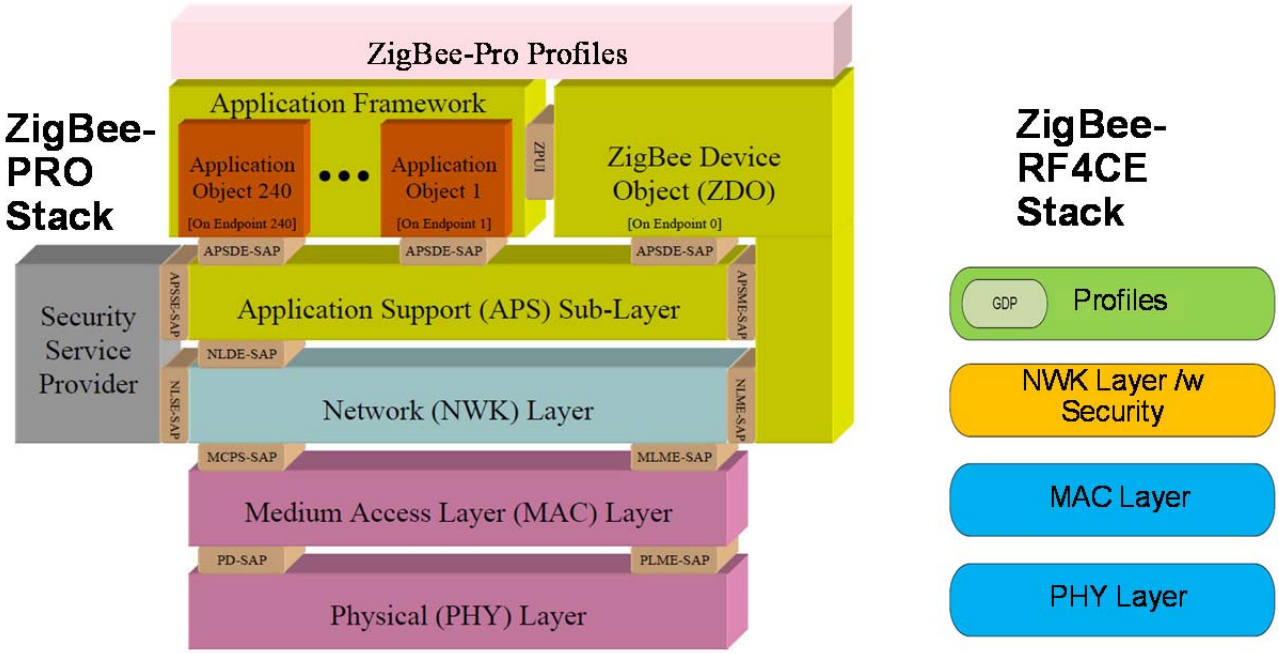

Figure 1: ZigBee-PRO and ZigBee-RF4CE Stacks

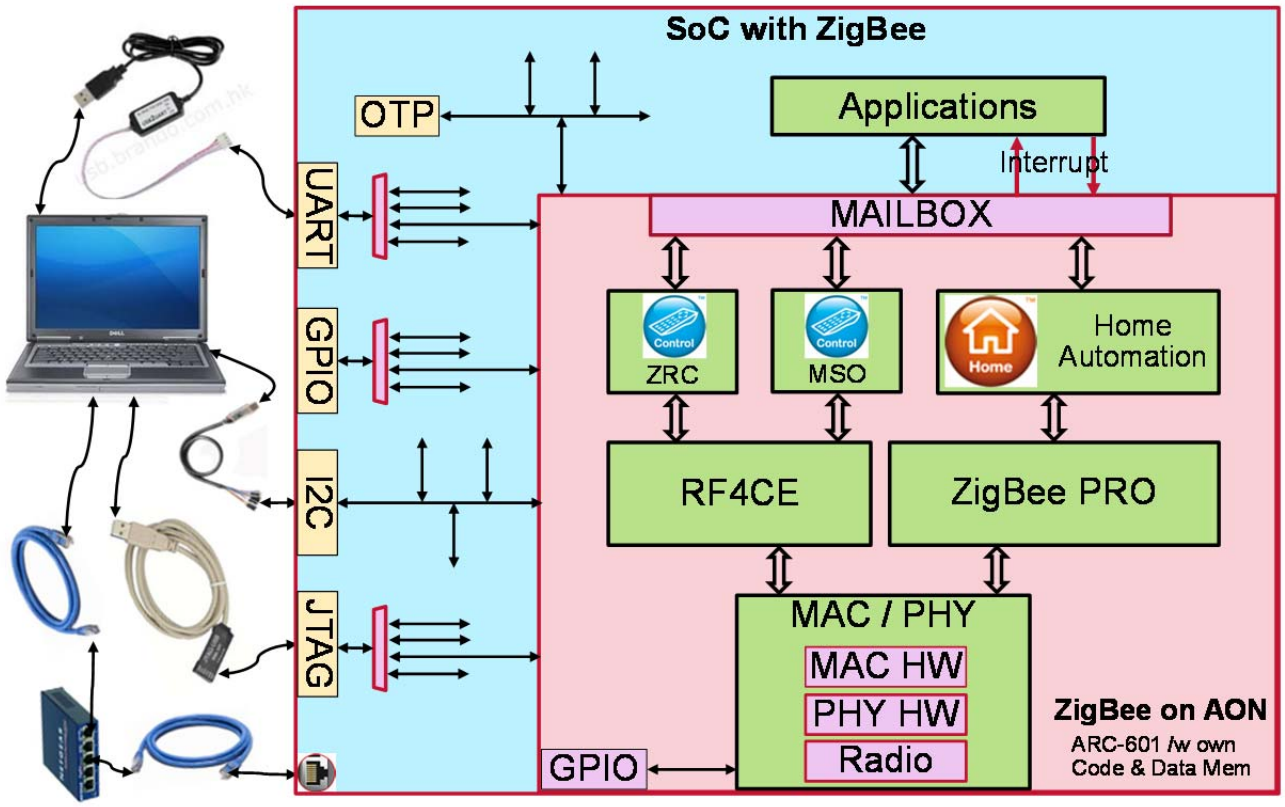

Figure 2: ZigBee Block Diagram in a SoC

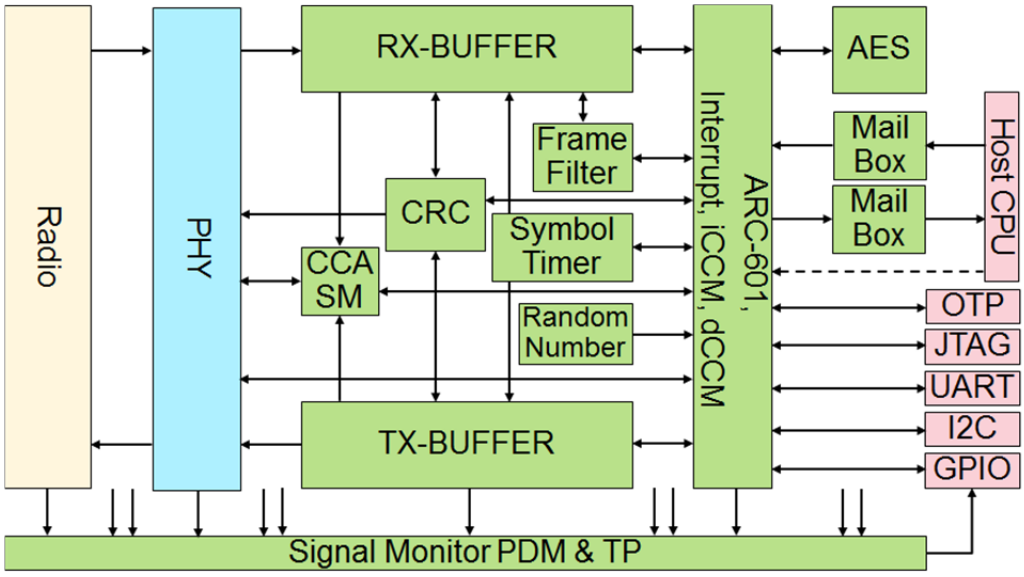

Figure 3: ZigBee Hardware Block Diagram

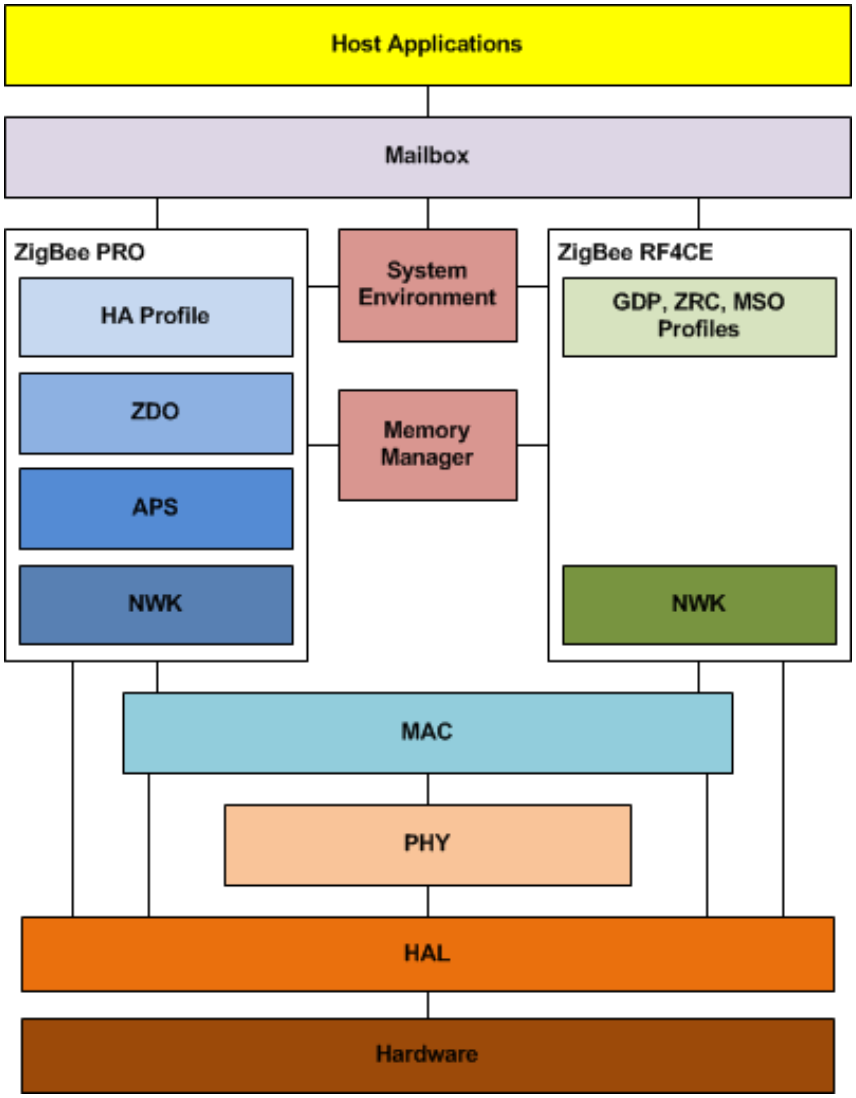

Figure 4: BroadBee Software Block Diagram

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.

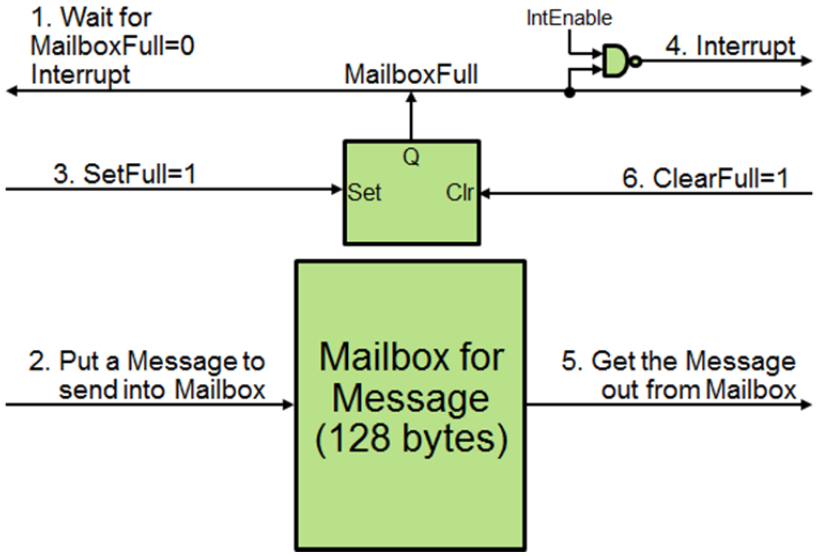

Figure 5: Mailbox Access Protocol

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.

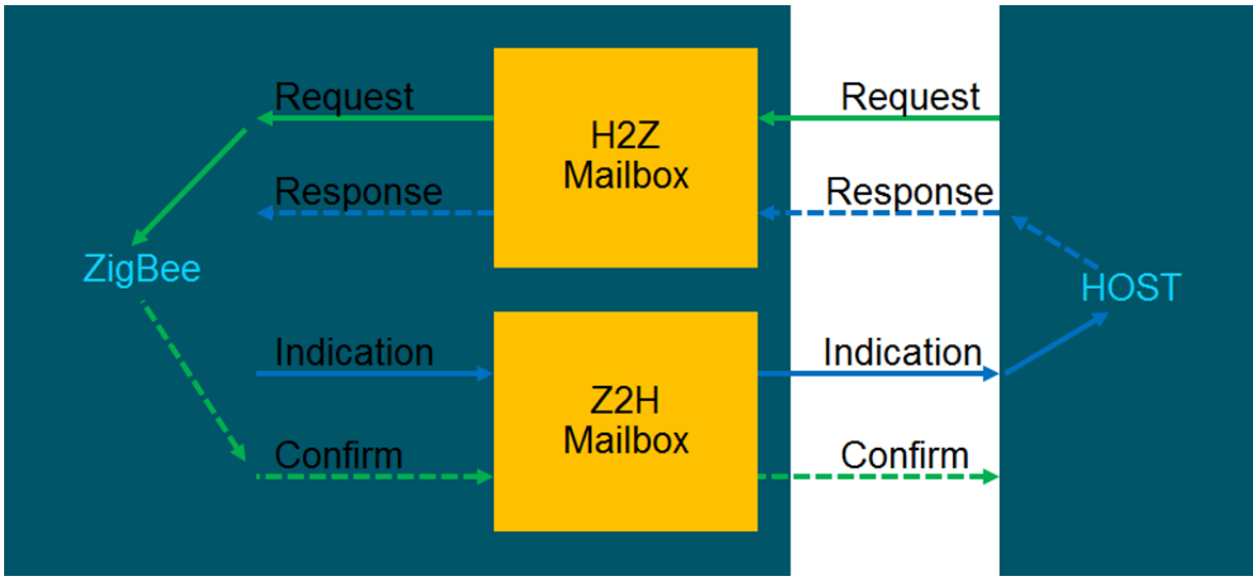

Figure 6: Service Primitives through Mailbox

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.

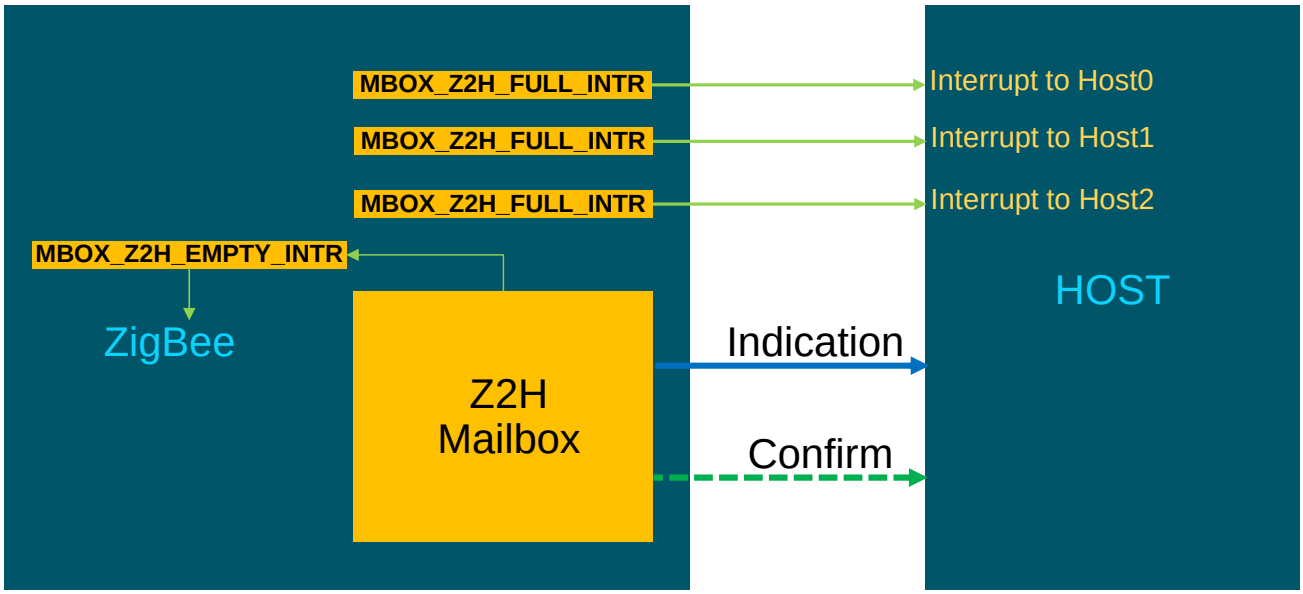

Figure 7: Message from ZigBee to Host

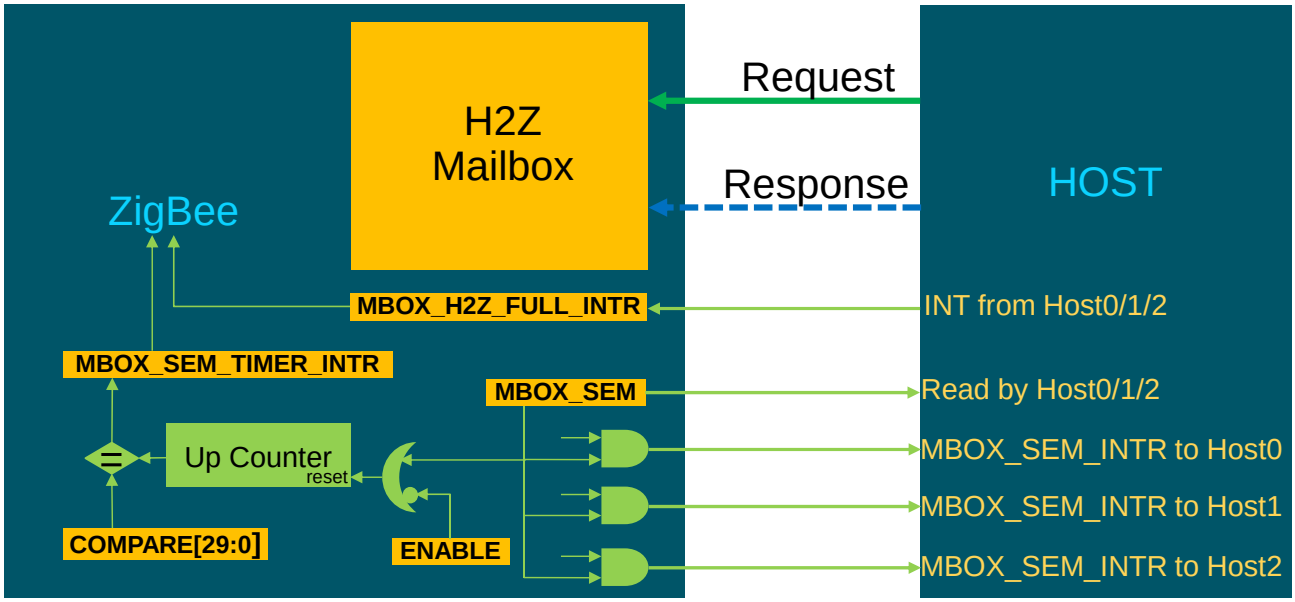

Figure 8: Message from Host to ZigBee

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.

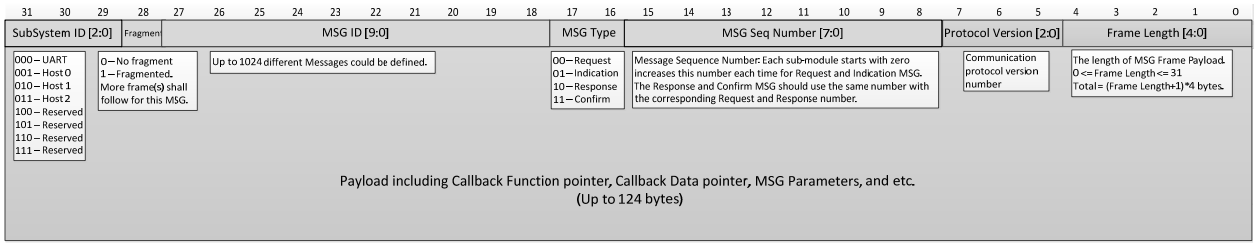

Figure 9: Message Format

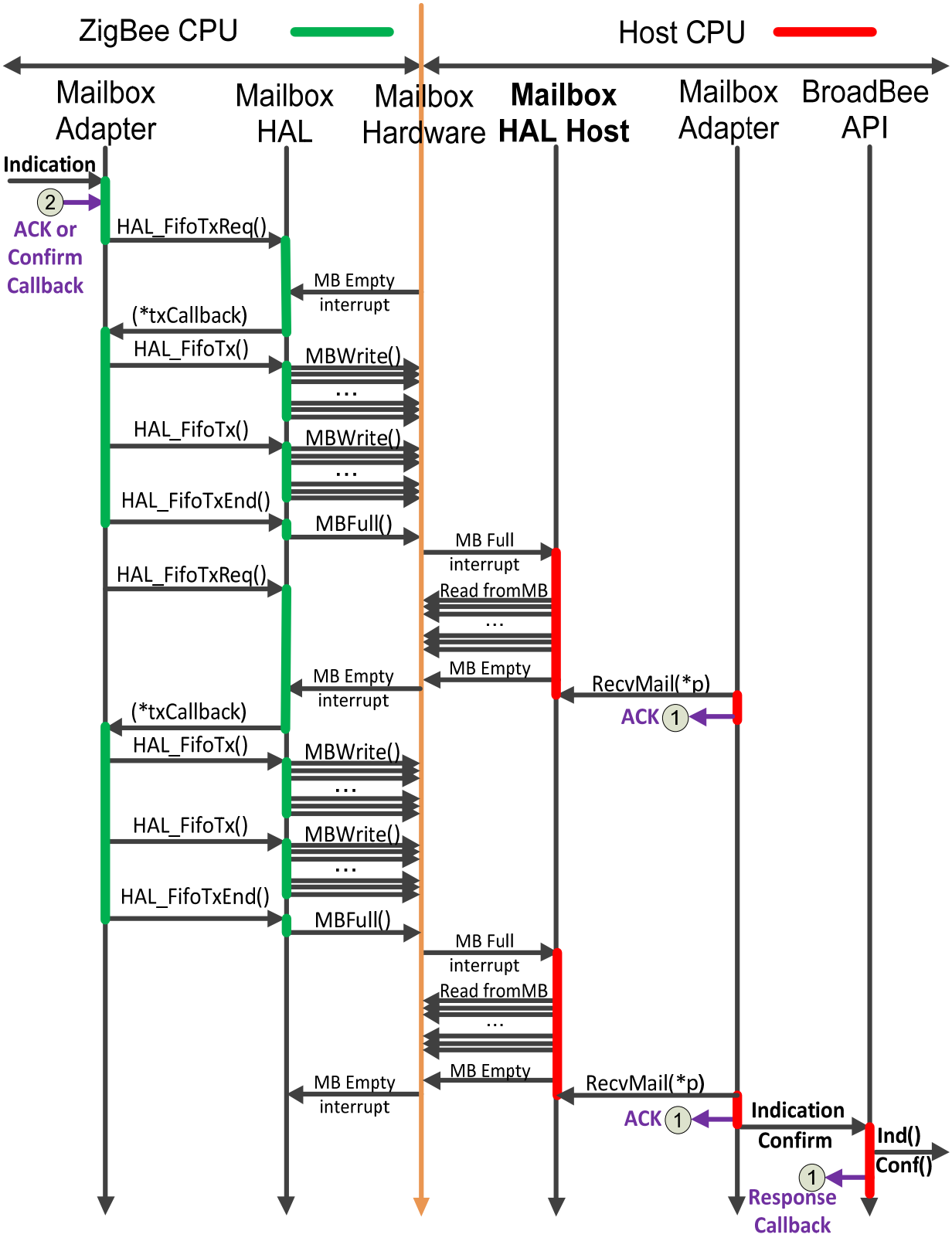

Figure 10: Example of a message from ZigBee to Host

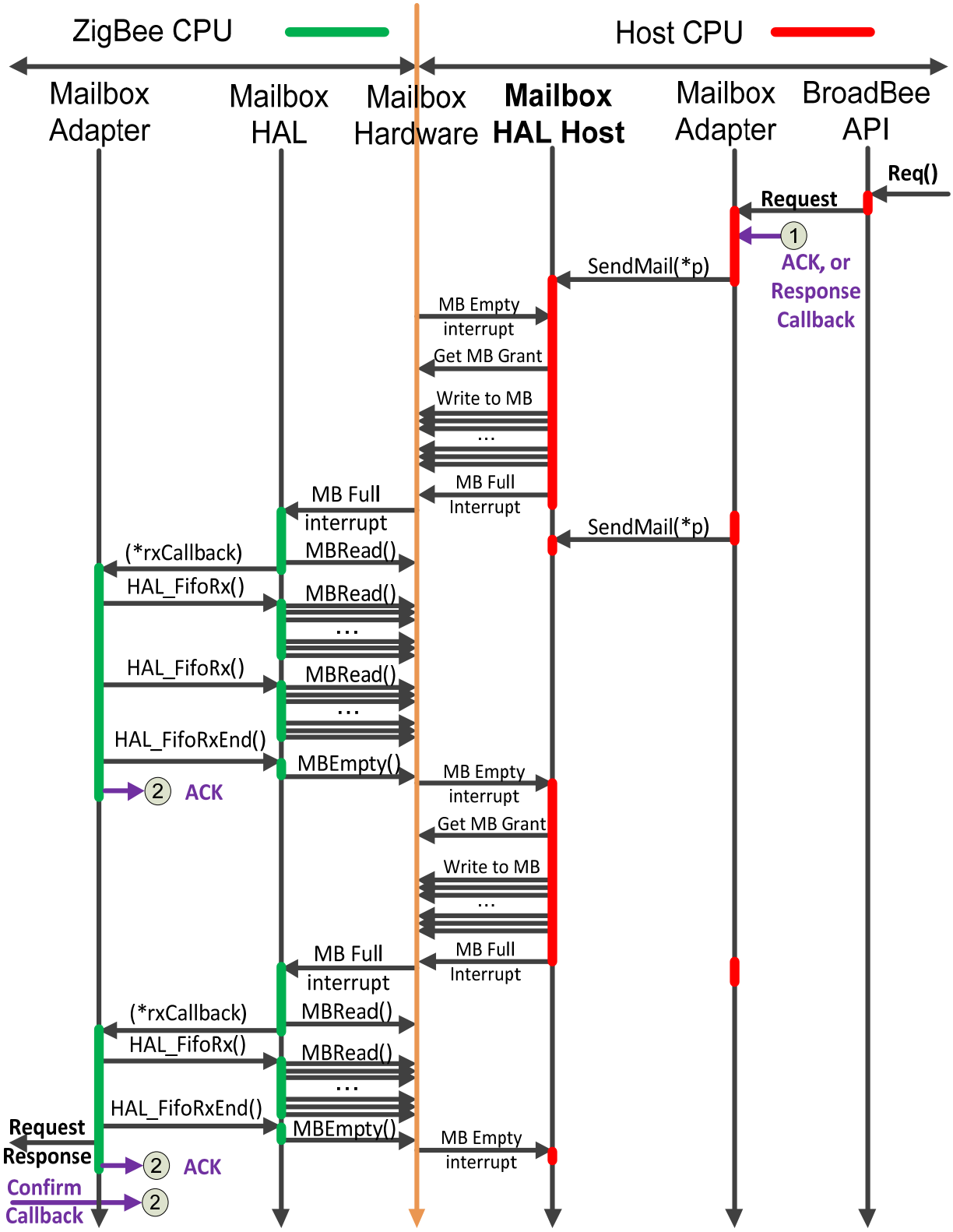

Figure 11: Example of a message from Host to ZigBee

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.

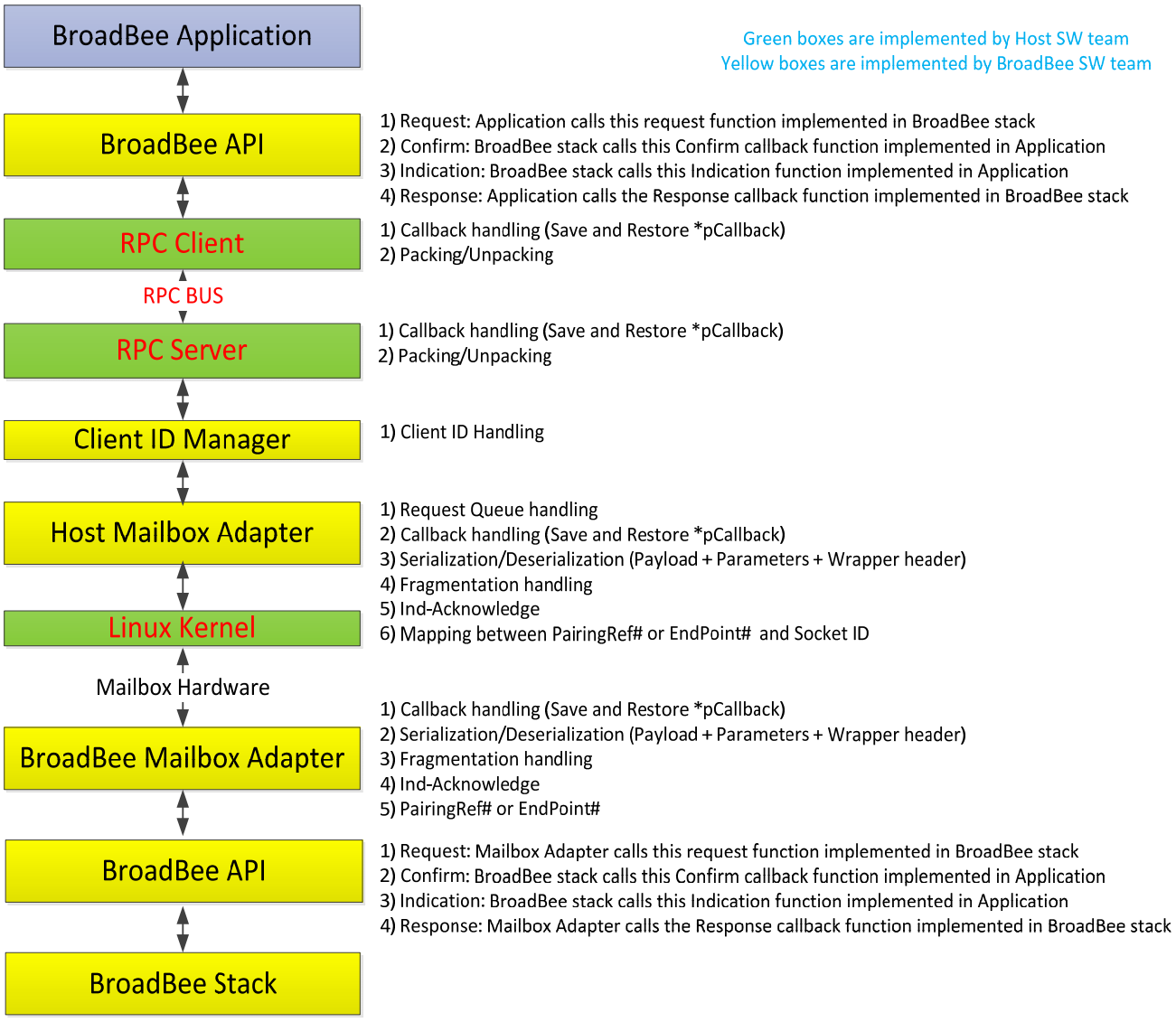

Figure 12: Communication Flow across MailBox

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);
```

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);
```

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);
```

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);
```

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 function call. 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 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. Enumerations

4.1.1. RF4CE_ZRC1_AttributesID_t

The RF4CE ZRC1 Attributes identifiers

```
typedef enum _RF4CE_ZRC1_AttributesID_t
{
    /* ZRC 1. The interval in ms at which user command repeat frames
       will be transmitted. */
    RF4CE_ZRC1_APL_KEY_REPEAT_INTERVAL = 0x80,
    /* ZRC 1. The duration that a recipient of a user control
       repeated command frame waits before terminating a repeated
       operation. */
    RF4CE_ZRC1_APL_KEY_REPEAT_WAIT_TIME = 0x81,
    /* ZRC 1. The value of the KeyExTransferCount parameter passed
       to the pair request primitive during the push button pairing
       procedure. */
    RF4CE_ZRC1_APL_KEY_EXCHANGE_TRANSFER_COUNT = 0x82,
    /* ZRC 1. Command discovery value. */
    RF4CE_ZRC1_APL_COMMAND_DISCOVERY = 0x83,
} RF4CE_ZRC1_AttributesID_t;
```

4.1.2. RF4CE_ZRC1_BindStatus_t

RF4CE ZRC 1.1 Bind result status.

```
typedef enum _RF4CE_ZRC1_BindStatus_t
{
    RF4CE_ZRC1_BOUND = 0,
    RF4CE_ZRC1_BIND_ERROR_DISCOVERY,
    RF4CE_ZRC1_BIND_ERROR_PAIRING,
    RF4CE_ZRC1_BIND_ERROR_NO_PAIRING_ENTRY,
    RF4CE_ZRC1_BIND_ERROR_TIMEOUT
} RF4CE_ZRC1_BindStatus_t;
```

4.1.3. RF4CE_ZRC_AttributeStatus_t

RF4CE ZRC Attributes operation status code.

```
typedef enum _RF4CE_ZRC_AttributeStatus_t
{
    /* Operation completed successfully. */
    RF4CE_ZRC_ATTR_STATUS_SUCCESS = 0,
    /* The supplied attribute ID is invalid. */
    RF4CE_ZRC_ATTR_STATUS_UNSUPPORTED_ATTRIBUTE,
    /* The request is invalid for that attribute ID. */
    RF4CE_ZRC_ATTR_STATUS_ILLEGAL_REQUEST,
    /* Attribute's index is out of range. For arrays only. */
    RF4CE_ZRC_ATTR_STATUS_INVALID_ENTRY
} RF4CE_ZRC_AttributeStatus_t;
```

4.1.4. RF4CE_ZRC1_CommandDiscoveryStatus_t

RF4CE ZRC 1.1 Command Discovery Request status.

```
typedef enum _RF4CE_ZRC1_CommandDiscoveryStatus_t
{
    RF4CE_ZRC1_COMMAND_DISCOVERY_SUCCESS = 0,
    RF4CE_ZRC1_COMMAND_DISCOVERY_SEND,
    RF4CE_ZRC1_COMMAND_DISCOVERY_RECEIVED,
    RF4CE_ZRC1_COMMAND_DISCOVERY_TIMEOUT,
    RF4CE_ZRC1_COMMAND_DISCOVERY_INVALID_REQUEST
} RF4CE_ZRC1_CommandDiscoveryStatus_t;
```

4.1.5. RF4CE_ZRC_ControlCommandConfStatus_t

RF4CE ZRC Control Command Status

```
typedef enum _RF4CE_ZRC_ControlCommandConfStatus_t
{
    RF4CE_ZRC_CC_SUCCESS = 0,
    RF4CE_ZRC_CC_INVALID_PARAMETER,
    RF4CE_ZRC_CC_INVALID_REQUEST,
    RF4CE_ZRC_CC_ERROR_SENDING
} RF4CE_ZRC_ControlCommandConfStatus_t;
```

4.2. API

4.2.1. 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.

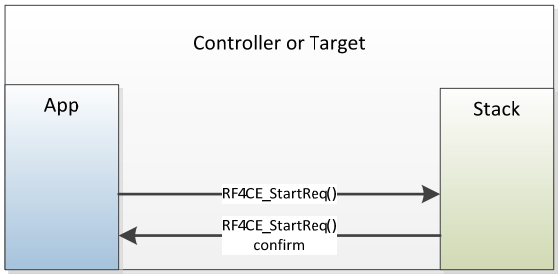

Figure 13: RF4CE ZRC profile start

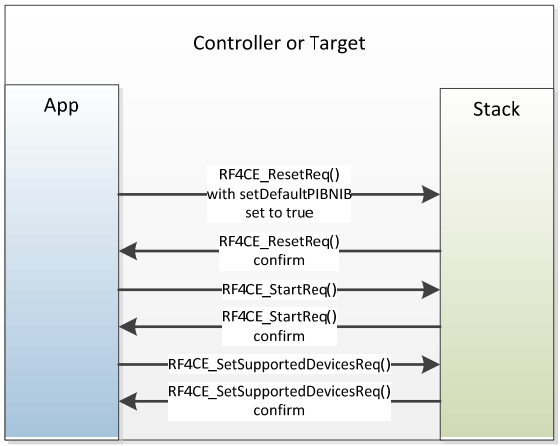

Figure 14: RF4CE ZRC profile start to factory default settings

4.2.2. 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 issued 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.

4.2.2.1. 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.

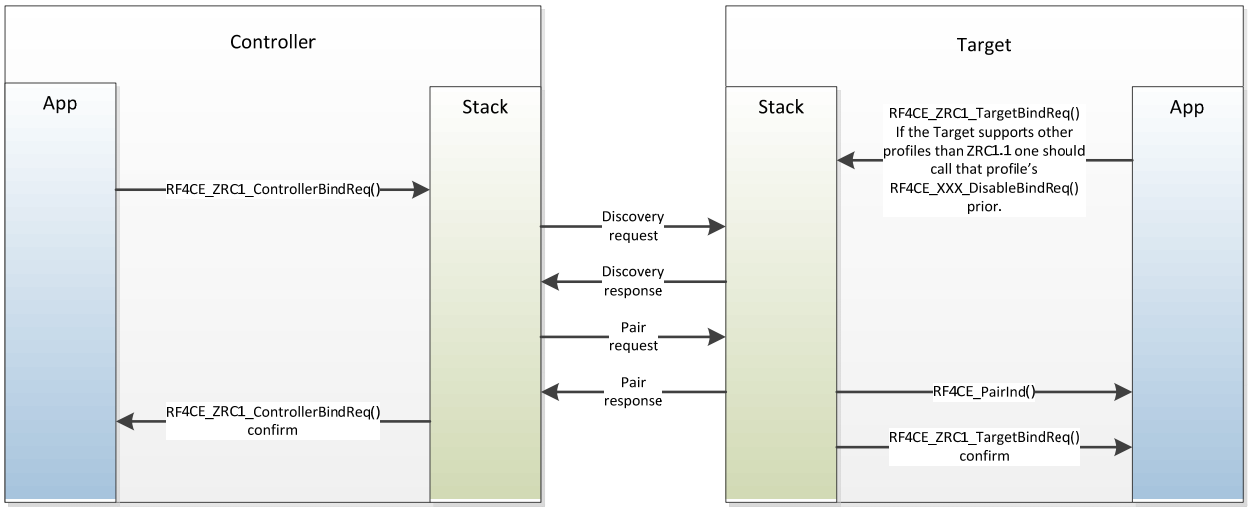

Figure 15: RF4CE ZRC 1.1 Binding Sequence

4.2.3. 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.

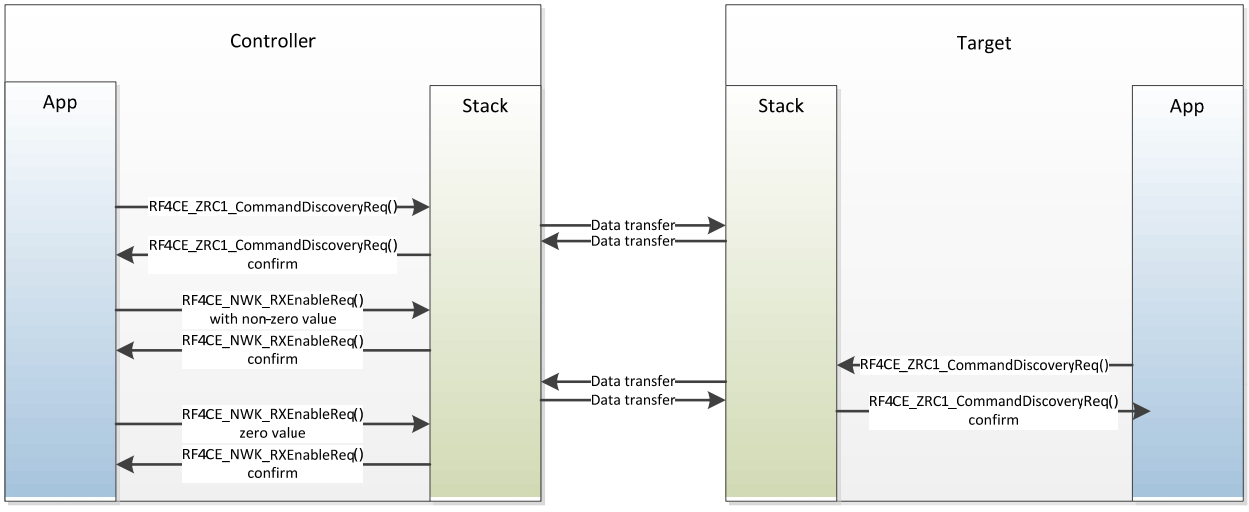

Figure 16: RF4CE ZRC 1.1 Command Discovery function

4.2.4. 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.

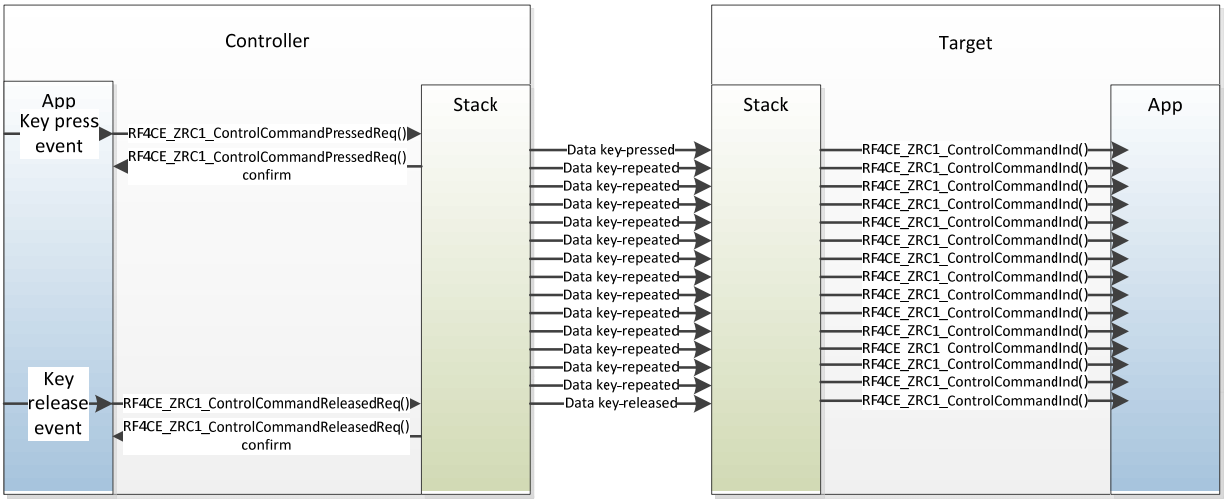

Figure 17: RF4CE ZRC 1.1 Control Command function

4.2.5. 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.

4.2.6. 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.

4.2.7. 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.

4.2.8. Heartbeat handling

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

4.2.9. Types

4.2.9.1. ZRC 1.1 Types

4.2.9.1.1. RF4CE_ZRC1_AttributesID_t

The RF4CE ZRC1 Attributes identifiers

```
typedef enum _RF4CE_ZRC1_AttributesID_t
{
    /* ZRC 1. The interval in ms at which user command repeat frames
       will be transmitted. */
    RF4CE_ZRC1_APL_KEY_REPEAT_INTERVAL = 0x80,
    /* ZRC 1. The duration that a recipient of a user control
       repeated command frame waits before terminating a repeated
       operation. */
    RF4CE_ZRC1_APL_KEY_REPEAT_WAIT_TIME = 0x81,
    /* ZRC 1. The value of the KeyExTransferCount parameter passed
       to the pair request primitive during the push button pairing
       procedure. */
    RF4CE_ZRC1_APL_KEY_EXCHANGE_TRANSFER_COUNT = 0x82,
    /* ZRC 1. Command discovery value. */
    RF4CE_ZRC1_APL_COMMAND_DISCOVERY = 0x83,
} RF4CE_ZRC1_AttributesID_t;
```

4.2.9.1.2. RF4CE_ZRC1_BindStatus_t

RF4CE ZRC 1.1 Bind result status.

```
typedef enum _RF4CE_ZRC1_BindStatus_t
{
    RF4CE_ZRC1_BOUND = 0,
    RF4CE_ZRC1_BIND_ERROR_DISCOVERY,
    RF4CE_ZRC1_BIND_ERROR_PAIRING,
    RF4CE_ZRC1_BIND_ERROR_NO_PAIRING_ENTRY,
    RF4CE_ZRC1_BIND_ERROR_TIMEOUT
} RF4CE_ZRC1_BindStatus_t;
```

4.2.9.1.3. RF4CE_ZRC_AttributeStatus_t

RF4CE ZRC Attributes operation status code.

```
typedef enum _RF4CE_ZRC_AttributeStatus_t
{
    /* Operation completed successfully. */
```

```
    RF4CE_ZRC_ATTR_STATUS_SUCCESS = 0,
    /* The supplied attribute ID is invalid. */
    RF4CE_ZRC_ATTR_STATUS_UNSUPPORTED_ATTRIBUTE,
    /* The request is invalid for that attribute ID. */
    RF4CE_ZRC_ATTR_STATUS_ILLEGAL_REQUEST,
    /* Attribute's index is out of range. For arrays only. */
    RF4CE_ZRC_ATTR_STATUS_INVALID_ENTRY
} RF4CE_ZRC_AttributeStatus_t;
```

4.2.9.1.4. RF4CE_ZRC1_CommandDiscoveryStatus_t

RF4CE ZRC 1.1 Command Discovery Request status.

```
typedef enum _RF4CE_ZRC1_CommandDiscoveryStatus_t
{
    RF4CE_ZRC1_COMMAND_DISCOVERY_SUCCESS = 0,
    RF4CE_ZRC1_COMMAND_DISCOVERY_SEND,
    RF4CE_ZRC1_COMMAND_DISCOVERY_RECEIVED,
    RF4CE_ZRC1_COMMAND_DISCOVERY_TIMEOUT
} RF4CE_ZRC1_CommandDiscoveryStatus_t;
```

4.2.9.1.5. RF4CE_ZRC_ControlCommandConfStatus_t

RF4CE ZRC Control Command Status

```
typedef enum _RF4CE_ZRC_ControlCommandConfStatus_t
{
    RF4CE_ZRC_CC_SUCCESS = 0,
    RF4CE_ZRC_CC_INVALID_PARAMETER,
    RF4CE_ZRC_CC_ERROR_SENDING
} RF4CE_ZRC_ControlCommandConfStatus_t;
```

4.2.9.1.6. RF4CE_ZRC1_Attribute_t

RF4CE ZRC 1.1 Vendor Specific status

```
typedef RF4CE_NLDE_DATA_Status_t RF4CE_ZRC1_VendorSpecificStatus_t;
```

4.2.9.1.7. RF4CE_ZRC1_VendorSpecificStatus_t

RF4CE ZRC 1.1 Attribute DATA union

```
typedef union _RF4CE_ZRC1_Attribute_t
{
    /* ZRC 1. The interval in ms at which user command repeat frames
    will be transmitted. */
    uint32_t aplZRC1KeyRepeatInterval;
    /* ZRC 1. The duration that a recipient of a user control
    repeated command frame waits before terminating a repeated
    operation. */
    uint32_t aplZRC1KeyRepeatWaitTime;
    /* ZRC 1. The value of the KeyExTransferCount parameter passed
    to the pair request primitive during the push button pairing
```

```
    procedure. */
    uint8_t aplZRC1KeyExchangeTransferCount;
    /* ZRC 1. Command discovery bitmap. */
    uint8_t aplZRC1CommandDiscovery[32];
} RF4CE_ZRC1_Attribute_t;
```

4.2.9.1.8. RF4CE_ZRC1_GetAttributeDescr_t

RF4CE ZRC 1.1 Attribute Get request descriptor declaration

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

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

typedef struct _RF4CE_ZRC1_GetAttributeDescr_t
    RF4CE_ZRC1_GetAttributeDescr_t;

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

struct _RF4CE_ZRC1_GetAttributeDescr_t
{
#ifndef _HOST_
    SYS_QueueElement_t queueElement;
    uint8_t requestType;
#endif /* _HOST_ */
    /* Request parameters */
    RF4CE_ZRC1_GetAttributeReqParams_t params;
    /* Callback on request completion. */
    RF4CE_ZRC1_GetAttributeCallback_t callback;
};
```

4.2.9.1.9. RF4CE_ZRC1_SetAttributeDescr_t

RF4CE ZRC 1.1 Attribute Set request descriptor declaration

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

typedef struct _RF4CE_ZRC1_SetAttributeConfParams_t
{
```

```
    uint8_t status;
} RF4CE_ZRC1_SetAttributeConfParams_t;

typedef struct _RF4CE_ZRC1_SetAttributeDescr_t
    RF4CE_ZRC1_SetAttributeDescr_t;

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

struct _RF4CE_ZRC1_SetAttributeDescr_t
{
#ifndef _HOST_
    /* Service field. */
    SYS_QueueElement_t queueElement;
    uint8_t requestType;
#endif /* _HOST_ */
    /* Request parameters */
    RF4CE_ZRC1_SetAttributeReqParams_t params;
    /* Callback on request completion. */
    RF4CE_ZRC1_SetAttributeCallback_t callback;
};
```

4.2.9.1.10. RF4CE_ZRC1_BindReqDescr_t

RF4CE ZRC 1.1 Bind request declaration

```
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;

typedef struct _RF4CE_ZRC1_BindReqDescr_t RF4CE_ZRC1_BindReqDescr_t;

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

struct _RF4CE_ZRC1_BindReqDescr_t
{
#ifndef _HOST_
    /* Service field. */
    SYS_QueueElement_t queueElement;
    uint8_t requestType;
#endif /* _HOST_ */
    /* Callback on request completion. */
    RF4CE_ZRC1_BindCallback_t callback;
};
```

4.2.9.1.11. RF4CE_ZRC1_CommandDiscoveryRequestFrame_t

RF4CE ZRC 1.1 Command Discovery Request Frame

```
typedef struct PACKED _RF4CE_ZRC1_CommandDiscoveryRequestFrame_t
{
    /* Command code. */
    uint8_t frameControl;
    /* Must be 0. */
    uint8_t reserved;
} RF4CE_ZRC1_CommandDiscoveryRequestFrame_t;
```

4.2.9.1.12. RF4CE_ZRC1_CommandDiscoveryResponseFrame_t

RF4CE ZRC 1.1 Command Discovery Response Frame

```
typedef struct PACKED _RF4CE_ZRC1_CommandDiscoveryResponseFrame_t
{
    /* Command code. */
    uint8_t frameControl;
    /* Must be 0. */
    uint8_t reserved;
    /* The data returned */
    uint8_t bitmap[32];
} RF4CE_ZRC1_CommandDiscoveryResponseFrame_t;
```

4.2.9.1.13. RF4CE_ZRC1_CommandDiscoveryReqDescr_t

RF4CE ZRC 1.1 Command Discovery request descriptor declaration

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

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

typedef struct _RF4CE_ZRC1_CommandDiscoveryReqDescr_t
    RF4CE_ZRC1_CommandDiscoveryReqDescr_t;

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

struct _RF4CE_ZRC1_CommandDiscoveryReqDescr_t
```

```
{
#ifndef _HOST_
    /* Service field. */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Request parameters */
    RF4CE_ZRC1_CommandDiscoveryReqParams_t params;
    /* Callback on request completion. */
    RF4CE_ZRC1_CommandDiscoveryCallback_t callback;
};
```

4.2.9.1.14. RF4CE_ZRC1_ControlCommandReqDescr_t

RF4CE ZRC 1.1 control command request descriptor declaration

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

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

typedef struct _RF4CE_ZRC1_ControlCommandReqDescr_t
    RF4CE_ZRC1_ControlCommandReqDescr_t;

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

struct _RF4CE_ZRC1_ControlCommandReqDescr_t
{
#ifndef _HOST_
    /* Service field. */
    SYS_QueueElement_t queueElement;
    /* Type of operation. */
    uint8_t typeOfOp;
#endif /* _HOST_ */
    /* Request parameters. */
    RF4CE_ZRC1_ControlCommandReqParams_t params;
    /* Request confirmation callback. */
    RF4CE_ZRC1_ControlCommandCallback_t callback;
};
```

4.2.9.1.15. RF4CE_ZRC1_VendorSpecificReqDescr_t

RF4CE ZRC 1.1 Vendor Specific request declaration

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

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

typedef struct _RF4CE_ZRC1_VendorSpecificReqDescr_t
    RF4CE_ZRC1_VendorSpecificReqDescr_t;

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

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

4.2.9.1.16. RF4CE_ZRC1_ControlCommandIndParams_t

RF4CE ZRC 1.1 Control Command Indication parameters structure

```
typedef struct _RF4CE_ZRC1_ControlCommandIndParams_t
{
    /* Pairing reference. */
    uint8_t pairingRef;
    /* RF4CE_ZRC1_USER_CONTROL_PRESSED or
    RF4CE_ZRC1_USER_CONTROL_REPEATED or
    RF4CE_ZRC1_USER_CONTROL_RELEASED. */
    uint8_t flags;
    /* Command code */
```

```
    uint8_t commandCode;
    /* Possible accompanying payload */
    SYS_DataPointer_t payload;
} RF4CE_ZRC1_ControlCommandIndParams_t;
```

4.2.9.1.17. RF4CE_ZRC1_VendorSpecificIndParams_t

RF4CE ZRC 1.1 Vendor Specific Indication parameters structure

```
typedef RF4CE_ZRC1_VendorSpecificReqParams_t
    RF4CE_ZRC1_VendorSpecificIndParams_t;
```

4.2.9.2. ZRC 2.0 Types

4.2.9.2.1. RF4CE_ZRC2_AttributesID_t

The RF4CE ZRC2 Attributes identifiers

```
typedef enum _RF4CE_ZRC2GDP2_AttributesID_t
{
    /* Specifies the version of the generic device profile
       specification to which the node was designed. */
    RF4CE_GDP2_APL_VERSION = 0x80,
    /* Specifies node's GDP capabilities */
    RF4CE_GDP2_APL_CAPABILITIES                    = 0x81,
    /* Specifies the value of the KeyExchangeTransferCount parameter
       passed to the pair request primitive during the binding
       procedure. */
    RF4CE_GDP2_APL_KEY_EXCHANGE_TRANSFER_COUNT     = 0x82,
    /* Specifies the high level status of the power supply of the
       node. */
    RF4CE_GDP2_APL_POWER_STATUS                    = 0x83,
    /* Polling Methods supported. */
    RF4CE_GDP2_APL_POLL_CONSTRAINTS                = 0x84,
    /* Current node's polling configuration. */
    RF4CE_GDP2_APL_POLL_CONFIGURATION              = 0x85,
    /* Specifies the number of pairing candidates to be kept in a
       sorted list during the binding procedure. */
    RF4CE_GDP2_APL_MAX_PAIRING_CANDIDATES          = 0x86,
    /* Specifies the time period in ms between the regular check
       validation requests that the controller transmits in the
       validation procedure. */
    RF4CE_GDP2_APL_AUTO_CHECK_VALIDATION_PERIOD    = 0x87,
    /* Specifies the maximum time in milliseconds that a
       binding recipient can stay in the validation procedure. */
    RF4CE_GDP2_APL_RECIPIENT_VALIDATION_WAIT_TIME = 0x88,
    /* Specifies the maximum time in milliseconds that a
       binding initiator can stay in the validation procedure. */
    RF4CE_GDP2_APL_INITIATOR_VALIDATION_WAIT_TIME = 0x89,
    /* Specifies the maximum time in ms that a node can stay in the
       validation procedure without receiving the Check Validation
       Status response corresponding to its Check Validation
```

```
        Request; or the maximum time in ms that a target can stay in
        The validation procedure without receiving a Check Validation
        Request. */
      RF4CE_GDP2_APL_LINK_LOST_WAIT_TIME            = 0x8A,
      /* Indicates what Identify actions the node supports. */
      RF4CE_GDP2_APL_IDENTIFY_CAPABILITIES          = 0x8B,

      /* ZRC 2.0 attributes */
      /* Specifies the version of the ZRC profile specification to
         which the device was designed. */
      RF4CE_ZRC2_APL_PROFILE_VERSION                = 0xa0,
      /* Specifies the profile's capabilities. */
      RF4CE_ZRC2_APL_PROFILE_CAPABILITIES           = 0xa1,
      /* Specifies the interval in ms at which Action command frames
         will be transmitted. */
      RF4CE_ZRC2_APL_ACTION_REPEAT_TRIGGER_INTERVAL  = 0xa2,
      /* Specifies the duration in ms that a recipient of an Action
         command frame waits before terminating the operation. */
      RF4CE_ZRC2_APL_ACTION_REPEAT_WAIT_TIME        = 0xa3,
      /* Indicates what Action banks are supported. */
      RF4CE_ZRC2_APL_ACTION_BANKS_SUPPORTED         = 0xa4,
      /* A vector of supported vendor IDs. */
      RF4CE_ZRC2_APL_IRDB_VENDOR_SUPPORT            = 0xa5,
      /* Action banks version. */
      RF4CE_ZRC2_APL_ACTION_BANKS_VERSION           = 0xa6,
      /* Specifies which Actions codes are supported for the
         corresponding Action bank. For RX. */
      RF4CE_ZRC2_APL_ACTION_CODES_SUPPORTED_RX      = 0xc0,
      /* Specifies which Actions codes are supported for the
         corresponding Action bank. For TX. */
      RF4CE_ZRC2_APL_ACTION_CODES_SUPPORTED_TX      = 0xc1,
      /* A vector of Action mappings that corresponds to the
         aplMappableActions attribute pushed by the controller. */
      RF4CE_ZRC2_APL_MAPPABLE_ACTIONS               = 0xc2,
      /* A one dimensional array of entries, where each entry
         corresponds to a description of the RF and IR code associated
         with a given action. */
      RF4CE_ZRC2_APL_ACTION_MAPPINGS                = 0xc3,
      /* A two dimensional array of entries, indexed by the instance
         ID and the HA Attribute ID. */
      RF4CE_ZRC2_APL_HOME_AUTOMATION                = 0xc4,
      /* A one dimensional array of entries, each is 256-bits array.*/
      RF4CE_ZRC2_APL_HOME_AUTOMATION_SUPPORTED      = 0xc5
  } RF4CE_ZRC2GDP2_AttributesID_t;
```

4.2.9.2.2. RF4CE_ZRC2_BindStatus_t

RF4CE ZRC 2.0 GDP 2.0 Bind result status

```
  typedef enum _RF4CE_ZRC2_BindStatus_t
  {
      RF4CE_ZRC2_BOUND = 0,
```

```
    RF4CE_ZRC2_BIND_ERROR_DISCOVERY,
    RF4CE_ZRC2_BIND_ERROR_PAIRING,
    RF4CE_ZRC2_BIND_ERROR_CONFIGURATION,
    RF4CE_ZRC2_BIND_ERROR_VALIDATION,
    RF4CE_ZRC2_BIND_ERROR_TIMEOUT,
    RF4CE_ZRC2_BIND_ERROR_NO_PAIRING_ENTRY
} RF4CE_ZRC2_BindStatus_t;
```

4.2.9.2.3. RF4CE_ZRC2_BindDuplicateClassID_t

RF4CE ZRC 2.0 GDP 2.0 Bind duplicate class IDs

```
typedef enum _RF4CE_ZRC2_BindDuplicateClassID_t
{
    RF4CE_ZRC2_DC_USE_AS_IS = 0,
    RF4CE_ZRC2_DC_RECLASSIFY = 1,
    RF4CE_ZRC2_DC_ABORT = 2,
    RF4CE_ZRC2_DC_RESERVED = 3
} RF4CE_ZRC2_BindDuplicateClassID_t;
```

4.2.9.2.4. RF4CE_ZRC2_ValidationStatus_t

RF4CE ZRC 2.0 GDP 2.0 Validation status

```
typedef enum _RF4CE_ZRC2_ValidationStatus_t
{
    RF4CE_ZRC2_VALIDATION_SUCCESS = 0,
    RF4CE_ZRC2_VALIDATION_PENDING = 1,
    RF4CE_ZRC2_VALIDATION_TIMEOUT = 2,
    RF4CE_ZRC2_VALIDATION_FAILURE = 3
} RF4CE_ZRC2_ValidationStatus_t;
```

4.2.9.2.5. RF4CE_ZRC2GDP2_Status_t

RF4CE ZRC 2.0 GDP 2.0 status values

```
typedef enum _RF4CE_ZRC2GDP2_Status_t
{
    RF4CE_ZRC2GDP2_SUCCESS = 0,
    RF4CE_ZRC2GDP2_UNSUPPORTED_REQUEST = 1,
    RF4CE_ZRC2GDP2_INVALID_PARAMETER = 2,
    RF4CE_ZRC2GDP2_CONFIGURATION_FAILURE = 3
} RF4CE_ZRC2GDP2_Status_t;
```

4.2.9.2.6. RF4CE_ZRC2GDP2_GetAttributeStatus_t

RF4CE ZRC 2.0 GDP 2.0 Get Attribute status values

```
typedef enum _RF4CE_ZRC2GDP2_GetAttributeStatus_t
{
    RF4CE_ZRC2GDP2_GET_ATTRIBUTE_SUCCESS = 0,
    RF4CE_ZRC2GDP2_GET_ATTRIBUTE_UNSUPPORTED_ATTRIBUTE = 1,
    RF4CE_ZRC2GDP2_GET_ATTRIBUTE_ILLEGAL_REQUEST = 2,
```

```
    RF4CE_ZRC2GDP2_GET_ATTRIBUTE_INVALID_ENTRY = 3
} RF4CE_ZRC2GDP2_GetAttributeStatus_t;
```

4.2.9.2.7. RF4CE_ZRC2_ClientNotifictionSubTypes_t

RF4CE ZRC 2.0 Client Notify Sub Type values

```
typedef enum _RF4CE_ZRC2_ClientNotifictionSubTypes_t
{
    RF4CE_ZRC2_CLIENT_NOTIFY_IDENTIFY              = 0x00,
    RF4CE_ZRC2_REQUEST_ACTION_MAPPING_NEGOTIATION  = 0x40,
    RF4CE_ZRC2_REQUEST_HOME_AUTOMATION_PULL        = 0x41
} RF4CE_ZRC2_ClientNotifictionSubTypes_t;
```

4.2.9.2.8. RF4CE_ZRC2_HeartbeatStatus_t

RF4CE ZRC 2.0 GDP 2.0 Heartbeat status

```
typedef RF4CE_NLDE_DATA_Status_t RF4CE_ZRC2_HeartbeatStatus_t;
```

4.2.9.2.9. RF4CE_ZRC2_ClientNotificationStatus_t

RF4CE ZRC 2.0 GDP 2.0 Client Notification status

```
typedef RF4CE_NLDE_DATA_Status_t
    RF4CE_ZRC2_ClientNotificationStatus_t
```

4.2.9.2.10. RF4CE_ZRC2_ControlCommandConfStatus_t

RF4CE ZRC 2.0 Control Command Status

```
typedef RF4CE_NLDE_DATA_Status_t
    RF4CE_ZRC2_ControlCommandConfStatus_t;
```

4.2.9.2.11. RF4CE_ZRC2_BindConfParams_t

RF4CE ZRC 2.0 GDP 2.0 Bind confirmation parameters

```
typedef struct _RF4CE_ZRC2_BindConfParams_t
{
    /* The status of binding. See RF4CE_ZRC2_BindStatus_t. */
    uint8_t status;
    /* The pairing reference on successful binding. */
    uint8_t pairingRef;
} RF4CE_ZRC2_BindConfParams_t;
```

4.2.9.2.12. RF4CE_ZRC2_BindReqDescr_t

RF4CE ZRC 2.0 GDP 2.0 Bind request

```
typedef struct _RF4CE_ZRC2_BindReqDescr_t RF4CE_ZRC2_BindReqDescr_t;

typedef void (*RF4CE_ZRC2_BindCallback_t)
```

```
    (RF4CE_ZRC2_BindReqDescr_t *req,
    RF4CE_ZRC2_BindConfParams_t *conf);

struct _RF4CE_ZRC2_BindReqDescr_t
{
#ifndef _HOST_
    /* Service field. */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Callback on request completion. */
    RF4CE_ZRC2_BindCallback_t callback;
};
```

4.2.9.2.13. RF4CE_ZRC2_ProxyBindReqParams_t

RF4CE ZRC 2.0 GDP 2.0 Proxy Bind request parameters

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

4.2.9.2.14. RF4CE_ZRC2_ProxyBindReqDescr_t

RF4CE ZRC 2.0 GDP 2.0 Bind request

```
typedef struct _RF4CE_ZRC2_ProxyBindReqDescr_t
    RF4CE_ZRC2_ProxyBindReqDescr_t;

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

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

4.2.9.2.15. RF4CE_ZRC2_BindingConfParams_t

RF4CE ZRC 2.0 GDP 2.0 Binding Common confirmation parameters

```
typedef struct _RF4CE_ZRC2_BindingConfParams_t
{
    /* One of the RF4CE_ZRC2GDP2_Status_t values */
```

```
    uint8_t status;
} RF4CE_ZRC2_BindingConfParams_t;
```

4.2.9.2.16. RF4CE_ZRC2_ButtonBindingReqDescr_t

RF4CE ZRC 2.0 GDP 2.0 Button Enabled Binding request descriptor

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

typedef struct _RF4CE_ZRC2_ButtonBindingReqDescr_t
    RF4CE_ZRC2_ButtonBindingReqDescr_t;

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

struct _RF4CE_ZRC2_ButtonBindingReqDescr_t
{
#ifndef _HOST_
    /* Service field. */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Request parameters. */
    RF4CE_ZRC2_ButtonBindingReqParams_t params;
    /* Callback on request completion. */
    RF4CE_ZRC2_ButtonBindingCallback_t callback;
};
```

4.2.9.2.17. RF4CE_ZRC2_BindingReqDescr_t

RF4CE ZRC 2.0 GDP 2.0 Binding request descriptor

```
typedef struct _RF4CE_ZRC2_BindingReqDescr_t
    RF4CE_ZRC2_BindingReqDescr_t;

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

struct _RF4CE_ZRC2_BindingReqDescr_t
{
#ifndef _HOST_
    struct
    {
        SYS_QueueElement_t queueElement;
        bool enable;
    } service;
```

```
#endif /* _HOST_ */
    RF4CE_ZRC2_BindingCallback_t callback;
};
```

4.2.9.2.18. RF4CE_ZRC2_CheckValidationIndParams_t

RF4CE ZRC 2.0 GDP 2.0 CheckValidation indication parameters

```
typedef struct _RF4CE_ZRC2_CheckValidationIndParams_t
{
    /* Pairing reference of the Check Validation request */
    uint8_t pairingRef;
} RF4CE_ZRC2_CheckValidationIndParams_t;
```

4.2.9.2.19. RF4CE_ZRC2_CheckValidationRespDescr_t

RF4CE ZRC 2.0 GDP 2.0 CheckValidation Response descriptor

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

typedef struct _RF4CE_ZRC2_CheckValidationRespDescr_t
{
#ifndef _HOST_
    /* Service field */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Response parameters */
    RF4CE_ZRC2_CheckValidationRespParams_t params;
} RF4CE_ZRC2_CheckValidationRespDescr_t;
```

4.2.9.2.20. RF4CE_ZRC2_AttributeId_t

RF4CE ZRC 2.0 Attributes Id type

```
typedef struct PACKED _RF4CE_ZRC2_AttributeId_t
{
    /* The attribute ID - one of the RF4CE_ZRC2GDP2_AttributesID_t
    values */
    uint8_t attributeId;
    /* The attribute index for arrayed attributes */
    union
    {
        struct
        {
            uint8_t lo;
            uint8_t hi;
```

```
        } indexByte;
        uint16_t index;
    } index;
} RF4CE_ZRC2_AttributeId_t;
```

4.2.9.2.21. RF4CE_ZRC2_GetAttributesReqDescr_t

RF4CE ZRC 2.0 Attributes Get request descriptor

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

typedef struct _RF4CE_ZRC2_GetAttributesReqDescr_t
    RF4CE_ZRC2_GetAttributesReqDescr_t;

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

struct _RF4CE_ZRC2_GetAttributesReqDescr_t
{
#ifndef _HOST_
    /* Service field */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Request parameters */
    RF4CE_ZRC2_GetAttributesReqParams_t params;
    /* Request callback */
    RF4CE_ZRC2_GetAttributesReqCallback_t callback;
};
```

4.2.9.2.22. RF4CE_ZRC2_GetAttributesConfParams_t

RF4CE ZRC 2.0 Attributes Get requests confirmation parameters

```
typedef struct _RF4CE_ZRC2_GetAttributesConfId_t
{
    /* The attribute id */
    RF4CE_ZRC2_AttributeId_t id;
    /* One of the RF4CE_ZRC2GDP2_GetAttributeStatus_t values */
    uint8_t status;
    /* The attribute value size - real size of the value field */
    uint8_t size;
    /* The attribute value. Real size is indicated by size field */
    uint8_t value[1];
```

```
} RF4CE_ZRC2_GetAttributesConfId_t;

typedef struct _RF4CE_ZRC2_GetAttributesConfParams_t
{
    /* The number of the attributes in the payload */
    uint8_t numAttributes;
    /* The payload must contain an array of
    RF4CE_ZRC2_GetAttributesConfId_t records */
    SYS_DataPointer_t payload;
} RF4CE_ZRC2_GetAttributesConfParams_t;
```

4.2.9.2.23. RF4CE_ZRC2_SetAttributesReqParams_t

RF4CE ZRC 2.0 Attributes Set request parameters

```
typedef struct PACKED _RF4CE_ZRC2_SetAttributeId_t
{
    /* The attribute id */
    RF4CE_ZRC2_AttributeId_t id;
    /* The attribute value size - real size of the value field */
    uint8_t size;
    /* The attribute value. Real size is shown by the size field */
    uint8_t value[1];
} RF4CE_ZRC2_SetAttributeId_t;

typedef struct _RF4CE_ZRC2_SetAttributesReqParams_t
{
    /* Pairing reference. If equal to RF4CE_NWK_INVALID_PAIRING_REF
    then local attributes are retrieved */
    uint8_t pairingRef;
    /* The number of the attributes in the request */
    uint8_t numAttributes;
    /* The payload must contain an array of
    RF4CE_ZRC2_SetAttributeId_t records */
    SYS_DataPointer_t payload;
} RF4CE_ZRC2_SetAttributesReqParams_t;
```

4.2.9.2.24. RF4CE_ZRC2_SetAttributesReqDescr_t

RF4CE ZRC 2.0 Attributes Set request descriptor

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

typedef struct _RF4CE_ZRC2_SetAttributesReqDescr_t
    RF4CE_ZRC2_SetAttributesReqDescr_t;

typedef void (*RF4CE_ZRC2_SetAttributesReqCallback_t)
```

```
    (RF4CE_ZRC2_SetAttributesReqDescr_t *req,
    RF4CE_ZRC2_SetAttributesConfParams_t *conf);

struct _RF4CE_ZRC2_SetAttributesReqDescr_t
{
#ifndef _HOST_
    /* Service field */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Request parameters */
    RF4CE_ZRC2_SetAttributesReqParams_t params;
    /* Request callback */
    RF4CE_ZRC2_SetAttributesReqCallback_t callback;
};
```

4.2.9.2.25. RF4CE_ZRC2_KeyExchangeReqDescr_t

RF4CE ZRC 2.0 GDP 2.0 Key Exchange request structure

```
typedef struct _RF4CE_ZRC2_KeyExchangeReqParams_t
{
    /* Pairing reference */
    uint8_t pairingRef;
    /* One or more of the RF4CE_ZRC2GDP2_KEY_EXCHANGE_FLAG_xxx
    values */
    uint8_t flags;
    /* Vendor specific parameter */
    uint8_t vendorSpecific;
    /* The shared secret used for Vendor specific request */
    SYS_DataPointer_t payload;
} RF4CE_ZRC2_KeyExchangeReqParams_t;

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

typedef struct _RF4CE_ZRC2_KeyExchangeReqDescr_t
    RF4CE_ZRC2_KeyExchangeReqDescr_t;

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

struct _RF4CE_ZRC2_KeyExchangeReqDescr_t
{
#ifndef _HOST_
    /* Service field */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
```

```
    /* Filled by application request structure */
    RF4CE_ZRC2_KeyExchangeReqParams_t params;
    /* The request confirmation callback */
    RF4CE_ZRC2_KeyExchangeCallback_t callback;
};
```

4.2.9.2.26. RF4CE_ZRC2_HeartbeatReqDescr_t

RF4CE ZRC 2.0 GDP 2.0 Heartbeat request descriptor

```
typedef struct _RF4CE_ZRC2_HeartbeatConfParams_t
{
    /* One of the RF4CE_ZRC2_HeartbeatStatus_t values */
    uint8_t status;
} RF4CE_ZRC2_HeartbeatConfParams_t;

typedef struct _RF4CE_ZRC2_HeartbeatReqParams_t
{
    /* Pairing reference */
    uint8_t pairingRef;
    uint8_t trigger;
} RF4CE_ZRC2_HeartbeatReqParams_t, RF4CE_ZRC2_HeartbeatIndParams_t;

typedef struct _RF4CE_ZRC2_HeartbeatReqDescr_t
    RF4CE_ZRC2_HeartbeatReqDescr_t;

typedef void (*RF4CE_ZRC2_HeartbeatCallback_t)
    (RF4CE_ZRC2_HeartbeatReqDescr_t *req,
    RF4CE_ZRC2_HeartbeatConfParams_t *conf);

struct _RF4CE_ZRC2_HeartbeatReqDescr_t
{
#ifndef _HOST_
    /* Service field */
    struct
    {
        SYS_QueueElement_t queueElement;
    } service;
#endif /* _HOST_ */
    /* Filled by application request structure */
    RF4CE_ZRC2_HeartbeatReqParams_t params;
    /* The request confirmation callback */
    RF4CE_ZRC2_HeartbeatCallback_t callback;
};
```

4.2.9.2.27. RF4CE_ZRC2_ClientNotificationReqDescr_t

RF4CE ZRC 2.0 GDP 2.0 Controller Notification request

```
typedef struct _RF4CE_ZRC2_ClientNotificationReqParams_t
{
    /* Pairing reference */
```

```
    uint8_t pairingRef;
    /* The client notification sub type value. One of the
    RF4CE_ZRC2_ClientNotifictionSubTypes_t values */
    uint8_t subType;
    /* The client notification variable payload */
    SYS_DataPointer_t payload;
} RF4CE_ZRC2_ClientNotificationReqParams_t,
  RF4CE_ZRC2_ClientNotificationIndParams_t;

typedef struct _RF4CE_ZRC2_ClientNotificationConfParams_t
{
    /* One of the RF4CE_ZRC2_ClientNotificationStatus_t values */
    uint8_t status;
} RF4CE_ZRC2_ClientNotificationConfParams_t;

typedef struct _RF4CE_ZRC2_ClientNotificationReqDescr_t
    RF4CE_ZRC2_ClientNotificationReqDescr_t;

typedef void (*RF4CE_ZRC2_ClientNotificationCallback_t)
    (RF4CE_ZRC2_ClientNotificationReqDescr_t *req,
    RF4CE_ZRC2_ClientNotificationConfParams_t *conf);

struct _RF4CE_ZRC2_ClientNotificationReqDescr_t
{
#ifndef _HOST_
    /* Service field */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Filled by application request structure */
    RF4CE_ZRC2_ClientNotificationReqParams_t params;
    /* The request confirmation callback */
    RF4CE_ZRC2_ClientNotificationCallback_t callback;
};
```

4.2.9.2.28. RF4CE_ZRC2_ControlCommandReqDescr_t

RF4CE ZRC 2.0 control command request descriptor

```
typedef struct _RF4CE_ZRC2_ActionVendor_t
{
    /* Use RF4CE_ZRC2_ACTION_GET_xxx/RF4CE_ZRC2_ACTION_SET_xxx to
    access. RF4CE_ZRC2_ACTION_RESERVED, RF4CE_ZRC2_ACTION_START and
    RF4CE_ZRC2_ACTION_REPEAT don't have any value and will be
    discarded anyway */
    uint8_t actionControl;
    /* Action bank. */
    uint8_t bank;
    /* Action code. */
    uint8_t code;
    /* Actual length of the actionPayload field. If this field is 0
    then next record in the array starts right instead of
    actionPayload filed */
```

```
    uint8_t payloadLength;
    /* Action Vendor ID. */
    uint16_t vendorId;
    /* Supplied payload. Actual length is set by the payloadLength
    field */
    uint8_t actionPayload[1];
} RF4CE_ZRC2_ActionVendor_t;

typedef struct _RF4CE_ZRC2_Action_t
{
    /* Use RF4CE_ZRC2_ACTION_GET_xxx/RF4CE_ZRC2_ACTION_SET_xxx to
    access. RF4CE_ZRC2_ACTION_RESERVED, RF4CE_ZRC2_ACTION_START and
    RF4CE_ZRC2_ACTION_REPEAT don't have any value and will be
    discarded anyway */
    uint8_t actionControl;
    /* Action bank. */
    uint8_t bank;
    /* Action code. */
    uint8_t code;
    /* Actual length of the actionPayload field. If this field is 0
    then next record in the array starts right instead of
    actionPayload filed */
    uint8_t payloadLength;
    /* Supplied payload. Actual length is set by the payloadLength
    field */
    uint8_t actionPayload[1];
} RF4CE_ZRC2_Action_t;

typedef struct _RF4CE_ZRC2_ControlCommandReqParams_t
{
    /* Pairing reference. */
    uint8_t pairingRef;
    /* Payload consisting of one or more RF4CE_ZRC2_Action_t and/or
    RF4CE_ZRC2_ActionVendor_t structures. */
    SYS_DataPointer_t payload;
} RF4CE_ZRC2_ControlCommandReqParams_t,
RF4CE_ZRC2_ControlCommandIndParams_t;

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

typedef struct _RF4CE_ZRC2_ControlCommandReqDescr_t
    RF4CE_ZRC2_ControlCommandReqDescr_t;

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

struct _RF4CE_ZRC2_ControlCommandReqDescr_t
```

```
{
#ifndef _HOST_
    /* Service field. */
    SYS_QueueElement_t queueElement;
#endif /* _HOST_ */
    /* Request parameters. */
    RF4CE_ZRC2_ControlCommandReqParams_t params;
    /* Request confirmation callback. */
    RF4CE_ZRC2_ControlCommandCallback_t callback;
};
```

4.2.9.2.29. RF4CE_ZRC2_StartValidationIndParams_t

RF4CE ZRC 2.0 GDP 2.0 Start Validation indication parameters

```
typedef struct _RF4CE_ZRC2_StartValidationIndParams_t
{
    uint8_t pairingRef;
} RF4CE_ZRC2_StartValidationIndParams_t;
```

4.2.9.3. Common Types

4.2.9.3.1. RF4CE_PairingReferenceIndParams_t

RF4CE Pairing Reference Indication parameters

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

4.2.9.3.2. RF4CE_PairingReferenceProfileIdIndParams_t

RF4CE Pairing Reference Indication with profile ID parameters

```
typedef struct _RF4CE_PairingReferenceProfileIdIndParams_t
{
    /* The pairing reference */
    uint8_t pairingRef;
    /* The profile ID */
    uint8_t profileId;
} RF4CE_PairingReferenceProfileIdIndParams_t;
```

4.2.9.3.3. RF4CE_UnpairReqParams_t

The RF4CE UNPAIR request parameters type:

```
typedef struct _RF4CE_UnpairReqParams_t
{
    /* The pairing reference of the remote node. */
    uint8_t pairingRef;
```

```
} RF4CE_UnpairReqParams_t;
```

4.2.9.3.4. RF4CE_UnpairConfParams_t

The RF4CE UNPAIR request confirmation type:

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

4.2.9.3.5. RF4CE_UnpairReqDescr_t

The RF4CE UNPAIR request descriptor type:

```
typedef struct _RF4CE_UnpairReqDescr_t
{
    /* The request parameters. */
    RF4CE_UnpairReqParams_t params;
    /* The request confirmation callback. */
    void (*callback)(RF4CE_UnpairReqDescr_t *req,
    RF4CE_UnpairConfParams_t *conf);
} RF4CE_UnpairReqDescr_t;
```

4.2.9.3.6. RF4CE_PairingReferenceIndParams_t

The RF4CE GDP Pairing Reference indication parameters structure:

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

4.2.9.3.7. RF4CE_StartResetConfParams_t

The RF4CE Start/Reset profile confirmation parameters structure:

```
typedef struct _RF4CE_StartResetConfParams_t
{
    /* The result status of the operation. */
    uint8_t status;
} RF4CE_StartResetConfParams_t;
```

4.2.9.3.8. RF4CE_StartReqDescr_t

The RF4CE Start profile descriptor structure:

```
typedef struct _RF4CE_StartReqDescr_t
{
    /* Callback that is called upon finish of the Start operation */
    void (*callback)(RF4CE_StartReqDescr_t *req,
```

```
    RF4CE_StartResetConfParams_t *conf);
} RF4CE_StartReqDescr_t;
```

4.2.9.3.9. RF4CE_ResetReqParams_t

The RF4CE Reset profile request parameters structure:

```
typedef struct _RF4CE_ResetReqParams_t
{
    /* True if full reset is done with default attributes set. */
    Bool8_t setDefaultNIB;
} RF4CE_ResetReqParams_t;
```

4.2.9.3.10. RF4CE_ResetReqDescr_t

The RF4CE Reset profile descriptor structure:

```
typedef struct _RF4CE_ResetReqDescr_t
{
    /* Request parameters structure */
    RF4CE_ResetReqParams_t params;
    /* Callback that is called upon finish of the Start operation */
    void (*callback)(RF4CE_ResetReqDescr_t *req,
    RF4CE_StartResetConfParams_t *conf);
} RF4CE_ResetReqDescr_t;
```

4.2.9.3.11. RF4CE_RegisterFieldMask_t

RF4CE Register Field Mask

```
typedef enum _RF4CE_RegisterFieldMask_t{
    RF4CE_ORGVENDORID_MASK     = (1 << 0),
    RF4CE_ORGVENDORRSTRING_MASK= (1 << 1),
    RF4CE_ORGUSERSTRING_MASK   = (1 << 2),
    RF4CE_ORGDEVICETYPE_MASK   = (1 << 3),
    RF4CE_ORGPROFILEID_MASK    = (1 << 4),
    RF4CE_MACADDRESS_MASK      = (1 << 5),
    RF4CE_PAIRREF_MASK         = (1 << 6)
}RF4CE_RegisterFieldMask_t;
```

4.2.9.3.12. RF4CE_RegisterVirtualDeviceParams_t

RF4CE Register Virtual Device parameters structure:

```
typedef struct _RF4CE_RegisterVirtualDeviceParams_t
{
    uint16_t orgVendorId;
    uint8_t orgVendorString[7];
    uint8_t orgUserString[15];
    uint8_t orgDeviceType;
    uint16_t orgProfileID;
    uint8_t expectedMacAddress[8];
```

```
    uint8_t pairRef;
    RF4CE_RegisterFieldMask_t  fieldValidMask;    /* mask bits to
indicate which parameter is valid */
} RF4CE_RegisterVirtualDeviceParams_t;
```

4.2.9.3.13. RF4CE_RegisterVirtualDeviceConfParams_t

RF4CE Register Virtual device confirmation pamameters structure:

```
typedef struct _RF4CE_RegisterVirtualDeviceConfParams_t
{
    uint32_t  status;
} RF4CE_RegisterVirtualDeviceConfParams_t;
```

4.2.9.3.14. RF4CE_RegisterVirtualDeviceReqDescr_t

RF4CE register virtual device request descriptor structure:

```
typedef struct _RF4CE_RegisterVirtualDeviceReqDescr_t
{
    /* 32-bit data. */
    void (*callback)(RF4CE_RegisterVirtualDeviceReqDescr_t   *reqDescr,
RF4CE_RegisterVirtualDeviceConfParams_t *confParams);      /*!< Confirm
callback function.  */
    RF4CE_RegisterVirtualDeviceParams_t     params;        /*!< Request
parameters set.     */
} RF4CE_RegisterVirtualDeviceReqDescr_t;
```

4.2.10. Common API

4.2.10.1.1. RF4CE_ZRC1_GetAttributesReq()

Prototype:

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

Description:

Starts asynchronous ZRC 1.1 Get Attributes Request.

Parameters:

`request`: pointer to the request descriptor.

Return value:

None.

4.2.10.1.2. RF4CE_ZRC1_SetAttributesReq()

Prototype:

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

Description:

Starts asynchronous ZRC 1.1 Set Attributes Request.

Parameters:

`request`: pointer to the request descriptor.

Return value:

None.

4.2.10.1.3. RF4CE_ZRC1_CommandDiscoveryReq()

Prototype:

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

Description:

Starts sending ZRC 1.1 Command Discovery request.

Parameters:

`request`: pointer to the ZRC 1.1 Command Discovery request descriptor structure.

Return value:

None.

4.2.10.1.4. RF4CE_ZRC1_ControlCommandPressedReq()

Prototype:

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

Description:

Starts sending ZRC 1.1 Control Command.

Parameters:

`request`: pointer to the ZRC 1.1 Control Command request descriptor structure.

Return value:

None.

4.2.10.1.5. RF4CE_ZRC1_ControlCommandReleasedReq()

Prototype:

```
void RF4CE_ZRC1_ControlCommandReleasedReq(
    RF4CE_ZRC1_ControlCommandReqDescr_t *request);
```

Description:

Ends sending ZRC 1.1 Control Command.

Parameters:

`request`: pointer to the ZRC 1.1 Control Command request descriptor structure.

Return value:

None.

4.2.10.1.6. RF4CE_ZRC1_VendorSpecificReq()

Prototype:

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

Description:

Starts Vendor Specific sending.

Parameters:

`request`: pointer to the request.

Return value:

None.

4.2.10.1.7. RF4CE_ZRC2_GetAttributesReq()

Prototype:

```
void RF4CE_ZRC2_GetAttributesReq(
  RF4CE_ZRC2_GetAttributesReqDescr_t *request);
```

Description:

Starts asynchronous ZRC 2.0 Get Attributes Request

Parameters:

`request`: pointer to the get attributes request descriptor structure.

Return value:

None.

4.2.10.1.8. RF4CE_ZRC2_SetAttributesReq()

Prototype:

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

Description:

Starts asynchronous ZRC 2.0 Set Attributes Request.

Parameters:

`request`: pointer to the set attributes request descriptor structure.

Return value:

None.

4.2.10.1.9. RF4CE_ZRC2_KeyExchangeReq()

Prototype:

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

Description:

Starts RF4CE ZRC 2.0 Key Exchange procedure.

Parameters:

`request`: pointer to the Key Exchange request descriptor structure.

Return value:

None.

4.2.10.1.10. RF4CE_ZRC2_CheckValidationResp()

Prototype:

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

Description:

Starts ZRC 2.0 Check Validation Response.

Parameters:

`response`: pointer to the response structure.

Return value:

None.

4.2.10.1.11. RF4CE_ZRC2_HeartbeatReq()

Prototype:

```
void RF4CE_ZRC2_HeartbeatReq(
    RF4CE_ZRC2_HeartbeatReqDescr_t *request);
```

Description:

Starts transmission of the ZRC 2.0 Heartbeat request.

Parameters:

`request`: pointer to the request structure.

Return value:

None.

4.2.10.1.12. RF4CE_ZRC2_ControlCommandPressedReq()

Prototype:

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

Description:

Starts sending ZRC 2.0 Control Command Pressed action.

Parameters:

`request`: pointer to the ZRC 2.0 Control Command request descriptor.

Return value:

None.

4.2.10.1.13. RF4CE_ZRC2_ControlCommandReleasedReq()

Prototype:

```
void RF4CE_ZRC2_ControlCommandReleasedReq(
    RF4CE_ZRC2_ControlCommandReqDescr_t *request);
```

Description:

Starts sending ZRC 2.0 Control Command Released action.

Parameters:

`request`: pointer to the ZRC 2.0 Control Command request descriptor.

Return value:

None.

4.2.10.1.14. RF4CE_UnpairReq()

Prototype:

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

Description:

Performs unbinding with the remote node.

Parameters:

`request`: pointer to the unpair request descriptor structure.

Return value:

None.

4.2.10.1.15. RF4CE_StartReq()

Prototype:

```
void RF4CE_StartReq(RF4CE_StartReqDescr_t *request);
```

Description:

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

Parameters:

`request`: pointer to the Start request descriptor structure.

Return value:

None.

4.2.10.1.16. RF4CE_ResetReq()

Prototype:

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

Description:

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

Parameters:

`request`: pointer to the Reset request descriptor structure.

Return value:

None.

4.2.10.1.17. RF4CE_RegisterVirtualDevice

Prototype:

```
void RF4CE_RegisterVirtualDevice(RF4CE_RegisterVirtualDeviceReqDescr_t
* request);
```

Description:

Register application information to show its intrest of special pairing ref.

Parameters:

`request`: pointer to the RF4CE register virtual device request descriptor structure.

Return value:

None.

4.2.11. Controller only API

4.2.11.1.1. RF4CE_ZRC1_ControllerBindReq()

Prototype:

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

Description:

Starts Controller side Binding Procedure.

Parameters:

`request`: pointer to the request.

Return value:

None.

4.2.11.1.2. RF4CE_ZRC2_ClientNotificationInd()

Prototype:

```
void RF4CE_ZRC2_ClientNotificationInd(
  RF4CE_ZRC2_ClientNotificationIndParams_t *indication);
```

Description:

ZRC 2.0 Client notification request indication.

Parameters:

`indication`: pointer to the Client Notification indication structure.

Return value:

None.

4.2.11.1.3. RF4CE_ZRC2_BindReq()

Prototype:

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

Description:

Starts asynchronous Binding Procedure.

Parameters:

`request`: pointer to the bind request descriptor structure.

Return value:

None.

4.2.11.1.4. RF4CE_ZRC2_ProxyBindReq()

Prototype:

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

Description:

Starts asynchronous Proxy Binding Procedure.

Parameters:

`request`: pointer to the proxy bind request descriptor structure.

Return value:

None.

4.2.12. Target only API

4.2.12.1.1. RF4CE_ZRC1_TargetBindReq()

Prototype:

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

Description:

Starts Target side Binding Procedure.

Parameters:

`request`: pointer to the request.

Return value:

None.

4.2.12.1.2. RF4CE_ZRC1_ControlCommandInd()

Prototype:

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

Description:

Indication to the HOST on ZRC 1.1 Control Command.

Parameters:

`request`: pointer to the indication structure.

Return value:

None.

4.2.12.1.3. RF4CE_ZRC1_VendorSpecificInd()

Prototype:

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

Description:

Indication to the HOST on ZRC 1.1 Vendor Specific.

Parameters:

`request`: pointer to the indication structure.

Return value:

None.

4.2.12.1.4. RF4CE_ZRC2_HeartbeatInd()

Prototype:

```
void RF4CE_ZRC2_HeartbeatInd(
  RF4CE_ZRC2_HeartbeatIndParams_t *indication);
```

Description:

ZRC 2.0 Heartbeat request indication.

Parameters:

`indication`: pointer to the Pairing Reference indication structure.

Return value:

None.

4.2.12.1.5. RF4CE_ZRC2_CheckValidationInd()

Prototype:

```
void RF4CE_ZRC2_CheckValidationInd(
  RF4CE_ZRC2_CheckValidationIndParams_t *indication);
```

Description:

ZRC 2.0 Check Validation indication.

Parameters:

`indication`: pointer to the Check Validation indication structure.

Return value:

None.

4.2.12.1.6. RF4CE_ZRC2_EnableBindingReq()

Prototype:

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

Description:

Enables Binding Procedure on the Target.

Parameters:

`request`: pointer to the Binding Request descriptor structure.

Return value:

None.

4.2.12.1.7. RF4CE_ZRC2_DisableBindingReq()

Prototype:

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

Description:

Disables Binding Procedure on the Target.

Parameters:

`request:` pointer to the Binding Request descriptor structure.

Return value:

None.

4.2.12.1.8. RF4CE_ZRC2_ButtonBindingReq()

Prototype:

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

Description:

Starts Button Pressed Binding Procedure on the Target.

Parameters:

`request`: pointer to the Button Binding Request descriptor structure.

Return value:

None.

4.2.12.1.9. RF4CE_ZRC2_ClientNotificationReq()

Prototype:

```
void RF4CE_ZRC2_ClientNotificationReq(
  RF4CE_ZRC2_ClientNotificationReqDescr_t *request);
```

Description:

Starts ZRC 2.0 Client notification request.

Parameters:

`request`: pointer to the Client Notification request descriptor structure.

Return value:

None.

4.2.12.1.10. RF4CE_ZRC2_ControlCommandInd()

Prototype:

```
void RF4CE_ZRC2_ControlCommandInd(
  RF4CE_ZRC2_ControlCommandIndParams_t *indication);
```

Description:

ZRC 2.0 Control Command indication.

Parameters:

`indication`: pointer to the ZRC Control Command indication data structure.

Return value:

None.

4.2.12.1.11. RF4CE_ZRC2_StartValidationInd()

Prototype:

```
void RF4CE_ZRC2_StartValidationInd(
    RF4CE_ZRC2_StartValidationIndParams_t *indication);
```

Description:

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

Parameters:

`indication`: pointer to the indication structure.

Return value:

None.

4.2.12.1.12. RF4CE_CounterExpiredInd()

Prototype:

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

Description:

Indication to the HOST on Counter Expired error.

Parameters:

`indication`: pointer to the indication structure.

Return value:

None.

4.2.12.1.13. RF4CE_PairInd()

Prototype:

```
void RF4CE_PairInd(
  RF4CE_PairingReferenceIndParams_t *indication);
```

Description:

Indication to the HOST on Pair indication data.

Parameters:

`indication`: pointer to the indication structure.

Return value:

None.

4.2.12.1.14. RF4CE_UnpairInd()

Prototype:

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

Description:

Indication to the HOST on Unpair indication data.

Parameters:

`indication`: pointer to the indication structure.

Return value:

None.

5. 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.

5.1. Enumerations

5.1.1.1. 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;
```

5.1.1.2. 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;
```

5.1.1.3. 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;
```

5.1.1.4. 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;
```

5.1.1.5. 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;
```

5.2. API

5.2.1. Internal profile attributes

5.2.1.1. Types

5.2.1.1.1. RF4CE_MSO_GetProfileAttributeReqParams_t

The MSO Profile Attribute GET request parameters.

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

5.2.1.1.2. RF4CE_MSO_ProfileAttributesUnion_t

The MSO Profile Attributes union for all of the attributes.

```
typedef struct _RF4CE_MSO_ProfileAttributesUnion_t
{
    uint16_t aplKeyRepeatWaitTime;
    uint16_t aplValidationWaitTime;
    uint16_t aplValidationInitialWatchdogTime;
    uint16_t aplKeyRepeatInterval;
```

```
    uint32_t aplResponseWaitTime;
    uint8_t aplMaxPairingCandidates;
    uint16_t aplLinkLostWaitTime;
    uint16_t aplAutoCheckValidationPeriod;
    uint8_t aplKeyExchangeTransferCount;
    uint8_t aplUserString[RF4CE_NWK_USER_STRING_LENGTH];
} RF4CE_MSO_ProfileAttributesUnion_t;
```

5.2.1.1.3. RF4CE_MSO_GetProfileAttributeConfParams_t

The MSO Profile Attribute GET request confirmation parameters.

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

5.2.1.1.4. RF4CE_MSO_GetProfileAttributeReqDescr_t

The MSO Profile Attribute GET request descriptor.

```
typedef struct _RF4CE_MSO_GetProfileAttributeReqDescr_t
{
    /* Request parameters. */
    RF4CE_MSO_GetProfileAttributeReqParams_t params;
    /* Request callback. */
    void (*callback)(RF4CE_MSO_GetProfileAttributeReqDescr_t *req,
                     RF4CE_MSO_GetProfileAttributeConfParams_t *conf);
} RF4CE_MSO_GetProfileAttributeReqDescr_t;
```

5.2.1.1.5. RF4CE_MSO_SetProfileAttributeReqParams_t

The MSO Profile Attribute SET request parameters.

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

5.2.1.1.6. RF4CE_MSO_SetProfileAttributeConfParams_t

The MSO Profile Attribute SET request confirmation parameters.

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

5.2.1.1.7. RF4CE_MSO_SetProfileAttributeReqDescr_t

The MSO Profile Attribute SET request descriptor.

```
typedef struct _RF4CE_MSO_SetProfileAttributeReqDescr_t
{
    /* Request parameters. */
    RF4CE_MSO_SetProfileAttributeReqParams_t params;
    /* Request callback. */
    void (*callback)(RF4CE_MSO_SetProfileAttributeReqDescr_t *req,
                     RF4CE_MSO_SetProfileAttributeConfParams_t *conf);
} RF4CE_MSO_SetProfileAttributeReqDescr_t;
```

5.2.1.2. Functions

5.2.1.2.1. RF4CE_MSO_GetProfileAttributeReq()

Prototype:

```
void RF4CE_MSO_GetProfileAttributeReq(
                     RF4CE_MSO_GetProfileAttributeReqDescr_t *request);
```

Description:

Starts RF4CE MSO Get Profile Attribute.

Parameters:

`request`: pointer to the request descriptor structure.

Return value:

None.

5.2.1.2.2. RF4CE_MSO_SetProfileAttributeReq()

Prototype:

```
void RF4CE_MSO_SetProfileAttributeReq(
                     RF4CE_MSO_SetProfileAttributeReqDescr_t *request);
```

Description:

Starts RF4CE MSO Set Profile Attribute.
Parameters:

`request`: pointer to the request descriptor structure.
Return value:

None.

5.2.2. Remote Information Base Support

5.2.2.1. Types

5.2.2.1.1. RF4CE_MSO_SetRIBAttributeReqParams_t

RF4CE MSO RIB attributes Set Request Parameters type.

```
typedef struct _RF4CE_MSO_SetRIBAttributeReqParams_t
{
    /* Pairing reference for the request. */
    uint8_t pairingRef;
    /* Requested attribute ID. */
    uint8_t attributeID;
    /* Requested attribute Index. */
    uint8_t attributeIndex;
    /* The requested attribute value. */
    SYS_DataPointer_t payload;
} RF4CE_MSO_SetRIBAttributeReqParams_t;
```

5.2.2.1.2. RF4CE_MSO_SetRIBAttributeConfParams_t

RF4CE MSO RIB attributes Set Response (Confirmation) Parameters type.

```
typedef struct _RF4CE_MSO_SetRIBAttributeConfParams_t
{
    /* Requested attribute status: one of the
RF4CE_MSO_RIBAttributeStatus_t codes */
    uint8_t status;
    /* Requested attribute ID. */
    uint8_t attributeId;
    /* Requested attribute Index. */
    uint8_t attributeIndex;
} RF4CE_MSO_SetRIBAttributeConfParams_t;
```

5.2.2.1.3. RF4CE_MSO_SetRIBAttributeReqDescr_t

RF4CE MSO RIB attributes Set Request Descriptor type.

```
typedef struct _RF4CE_MSO_SetRIBAttributeReqDescr_t
{
    /* Request parameters. */
    RF4CE_MSO_SetRIBAttributeReqParams_t params;
    /* Request callback. */
    void (*callback) (RF4CE_MSO_SetRIBAttributeReqDescr_t *req,
                      RF4CE_MSO_SetRIBAttributeConfParams_t *conf);
} RF4CE_MSO_SetRIBAttributeReqDescr_t;
```

5.2.2.1.4. RF4CE_MSO_GetRIBAttributeReqParams_t

RF4CE MSO RIB attributes Get Request Parameters type.

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

5.2.2.1.5. RF4CE_MSO_GetRIBAttributeConfParams_t

RF4CE MSO RIB attributes Get Response (Confirmation) Parameters type.

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

5.2.2.1.6. RF4CE_MSO_GetRIBAttributeReqDescr_t

RF4CE MSO RIB attributes Get Request Descriptor type.

```
typedef struct _RF4CE_MSO_GetRIBAttributeReqDescr_t
{
    /* Request parameters. */
    RF4CE_MSO_GetRIBAttributeReqParams_t params;
    /* Request callback. */
```

```
      void (*callback)(RF4CE_MSO_GetRIBAttributeReqDescr_t *req,
                       RF4CE_MSO_GetRIBAttributeConfParams_t *conf);
  } RF4CE_MSO_GetRIBAttributeReqDescr_t;
```

5.2.2.2. HOST Side API

5.2.2.2.1. RF4CE_MSO_GetRIBInd()

Prototype:

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

Description:

Get RF4CE MSO RIB Attribute from the HOST.

Parameters:

`request`: pointer to the request descriptor structure.

Return value:

None.

5.2.2.2.2. RF4CE_MSO_SetRIBInd()

Prototype:

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

Description:

Set RF4CE MSO RIB Attribute to the HOST.

Parameters:

`request`: pointer to the request descriptor structure.

Return value:

None.

5.2.2.3. Functions

5.2.2.3.1. RF4CE_MSO_GetRIBAttributeReq()

Prototype:

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

Description:

Initializes the RF4CE MSO RIB Attribute GET Request.

Parameters:

`request`: pointer to the request descriptor structure.

Return value:

None.

5.2.2.3.2. RF4CE_MSO_SetRIBAttributeReq()

Prototype:

```
void RF4CE_MSO_SetRIBAttributeReq(
  RF4CE_MSO_SetRIBAttributeReqDescr_t *request);
```

Description:

Initializes the RF4CE MSO RIB Attribute SET Request.

Parameters:

`request`: pointer to the request descriptor structure.

Return value:

None.

5.2.3. Binding Support

5.2.3.1. Types

5.2.3.1.1. RF4CE_MSO_ValidationStatus_t

RF4CE MSO Validation status.

```
typedef enum _RF4CE_MSO_ValidationStatus_t
{
    /* The validation is successful. */
    RF4CE_MSO_VALIDATION_SUCCESS = 0,
    /* The validation is still in progress. */
    RF4CE_MSO_VALIDATION_PENDING = 0xC0,
    /* The validation timed out, and the binding procedure SHOULD
       continue with other devices in the list. */
    RF4CE_MSO_VALIDATION_TIMEOUT,
    /* The validation was terminated at the target side, as more
       than one controller tried to pair. */
    RF4CE_MSO_VALIDATION_COLLISION,
    /* The validation failed, and the binding procedure SHOULD
       continue with other devices in the list. */
    RF4CE_MSO_VALIDATION_FAILURE,
    /* The validation is aborted, and the binding procedure SHOULD
       continue with other devices in the list. */
    RF4CE_MSO_VALIDATION_ABORT,
    /* The validation is aborted, and the binding procedure SHOULD
       NOT continue with other devices in the list. */
    RF4CE_MSO_VALIDATION_FULL_ABORT
```

```
} RF4CE_MSO_ValidationStatus_t;
```

5.2.3.1.2. RF4CE_MSO_BindStatus_t

RF4CE MSO Bind result status.

```
typedef enum _RF4CE_MSO_BindStatus_t
{
    RF4CE_MSO_BOUND = 0,
    RF4CE_MSO_BIND_ERROR_DISCOVERY,
    RF4CE_MSO_BIND_ERROR_PAIRING,
    RF4CE_MSO_BIND_ERROR_VALIDATION,
    RF4CE_MSO_BIND_ERROR_TIMEOUT
} RF4CE_MSO_BindStatus_t;
```

5.2.3.1.3. RF4CE_MSO_BindConfParams_t

RF4CE MSO Bind confirmation structure.

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

5.2.3.1.4. RF4CE_MSO_BindReqDescr_t

RF4CE MSO Bind request.

```
typedef struct _RF4CE_MSO_BindReqDescr_t
{
    /* Request callback */
    void (*callback)(RF4CE_MSO_BindReqDescr_t *req,
                     RF4CE_MSO_BindConfParams_t *conf);
} RF4CE_MSO_BindReqDescr_t;
```

5.2.3.1.5. RF4CE_MSO_CheckValidationRespConfParams_t

RF4CE MSO Validate confirmation parameters.

```
typedef struct _RF4CE_MSO_CheckValidationRespConfParams_t
{
    uint8_t status;
} RF4CE_MSO_CheckValidationRespConfParams_t;
```

5.2.3.1.6. RF4CE_MSO_CheckValidationReqParams_t

RF4CE MSO Check Validation request parameters.

```
typedef struct _RF4CE_MSO_CheckValidationReqParams_t
{
    uint8_t pairingRef;
    uint8_t flags;
} RF4CE_MSO_CheckValidationReqParams_t;
```

5.2.3.1.7. RF4CE_MSO_CheckValidationRespDescr_t

RF4CE MSO Validate request.

```
typedef struct _RF4CE_MSO_CheckValidationRespDescr_t
{
    RF4CE_MSO_CheckValidationReqParams_t params;
    void (*callback)(RF4CE_MSO_CheckValidationRespDescr_t *req,
                     RF4CE_MSO_CheckValidationRespConfParams_t *conf);
} RF4CE_MSO_CheckValidationRespDescr_t;
```

5.2.3.1.8. RF4CE_MSO_CheckValidationConfParams_t

RF4CE MSO Check Validation confirmation parameters.

```
typedef struct _RF4CE_MSO_CheckValidationConfParams_t
{
    uint8_t status;
} RF4CE_MSO_CheckValidationConfParams_t;
```

5.2.3.1.9. RF4CE_MSO_CheckValidationReqDescr_t

RF4CE MSO Check Validation request.

```
typedef struct _RF4CE_MSO_CheckValidationReqDescr_t
{
    RF4CE_MSO_CheckValidationReqParams_t params;
    void (*callback)(RF4CE_MSO_CheckValidationReqDescr_t *req,
                     RF4CE_MSO_CheckValidationConfParams_t *conf);
} RF4CE_MSO_CheckValidationReqDescr_t;
```

5.2.3.2. HOST Side API

5.2.3.2.1. RF4CE_MSO_CheckValidationInd()

Prototype:

```
void RF4CE_MSO_CheckValidationInd(
  RF4CE_MSO_CheckValidationRespDescr_t *indication);
```

Description:

Reflects the FIRST Check Validation Request to the HOST.

Parameters:

`indication`: pointer to the indication structure.

Return value:

None.

None.

5.2.3.3. Controller only API

5.2.3.3.1. RF4CE_MSO_BindReq()

Prototype:

```
void RF4CE_MSO_BindReq(RF4CE_MSO_BindReqDescr_t *request);
```

Description:

Starts binding procedure.

Parameters:

`request`: pointer to the request structure.

Return value:

None.

5.2.3.3.2. RF4CE_MSO_CheckValidationReq()

Prototype:

```
void RF4CE_MSO_CheckValidationReq(
  RF4CE_MSO_CheckValidationReqDescr_t *request);
```

Description:

Issues the Check Validation request.

Parameters:

`request`: pointer to the request structure.

Return value:

None.

5.2.4. User Control Command Support

5.2.4.1. Types

5.2.4.1.1. RF4CE_MSO_UserControlReqType_t

RF4CE MSO User Control request types.

```
typedef enum _RF4CE_MSO_UserControlReqType_t
{
    RF4CE_MSO_USER_CONTROL_PRESSED = MSO_CONTROL_PRESSED,
    RF4CE_MSO_USER_CONTROL_REPEATED = MSO_CONTROL_REPEATED,
    RF4CE_MSO_USER_CONTROL_RELEASED = MSO_CONTROL_RELEASED
} RF4CE_MSO_UserControlReqType_t;
```

5.2.4.1.2. RF4CE_MSO_UserControlReqParams_t

RF4CE MSO User Control request parameters.

```
typedef struct _RF4CE_MSO_UserControlReqParams_t
{
    uint8_t pairingRef;
    RF4CE_MSO_UserControlReqType_t requestType;
    SYS_DataPointer_t payload;
} RF4CE_MSO_UserControlReqParams_t;
```

5.2.4.1.3. RF4CE_MSO_UserControlConfParams_t

RF4CE MSO User Control confirmation parameters.

```
typedef struct _RF4CE_MSO_UserControlConfParams_t
{
    Bool8_t status;
} RF4CE_MSO_UserControlConfParams_t;
```

5.2.4.1.4. RF4CE_MSO_UserControlReqDescr_t

RF4CE MSO User Control request descriptor.

```
typedef struct _RF4CE_MSO_UserControlReqDescr_t
{
    RF4CE_MSO_UserControlReqParams_t params;
    void (*callback)(RF4CE_MSO_UserControlReqDescr_t *req,
                     RF4CE_MSO_UserControlConfParams_t *conf);
} RF4CE_MSO_UserControlReqDescr_t;
```

5.2.4.1.5. RF4CE_MSO_UserControlIndParams_t

RF4CE MSO User Control indication parameters.

```
typedef struct _RF4CE_MSO_UserControlIndParams_t
{
    uint8_t pairingRef;
    SYS_DataPointer_t payload;
} RF4CE_MSO_UserControlIndParams_t;
```

5.2.4.2. HOST Side API

5.2.4.2.1.1. RF4CE_MSO_UserControlInd()

Prototype:

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

Description:

MSO User Control command indication to HOST.

Parameters:

`indication`: pointer to the indication structure.

Return value:

None.

5.2.4.3. Controller only API

5.2.4.3.1.1. RF4CE_MSO_UserControlReq()

Prototype:

```
void RF4CE_MSO_UserControlReq(
  RF4CE_MSO_UserControlReqDescr_t *request);
```

Description:

Initiates MSO User Control request.

Parameters:

`request`: pointer to the request structure.

Return value:

None.

6. API for 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.

6.1. 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

6.2. 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

6.3. Common Data Types

6.3.1. APS_AddrMode_t

Address mode indicates which type of address shall be used. Refer to [1] section 2.2.5.1.1.2.

```
typedef enum
{
  /** Indirect addressing (using binding) */
  APS_INDIRECT_MODE = 0x00,
  /** Group addressing mode */
  APS_GROUP_ADDRESS_MODE = 0x01,
  /** Unicast addressing mode, with a 16-bit network (short) address
  */
  APS_SHORT_ADDRESS_MODE = 0x02,
  /** Unicast addressing mode, with a 64-bit IEEE (extended) address.
  (not supported) */
  APS_EXT_ADDRESS_MODE = 0x03
} APS_AddrMode_t;
```

6.3.2. APS_Address_t

Can be set to short, extended or group addresses depending on the address mode.

```
typedef union _APS_Address_t
{
```

```
    /** 16-bit network (short) address */
    uint16_t shortAddr;
    /** 64-bit IEEE (extended) address. */
    uint64_t extAddr;
  } APS_Address_t;
```

6.3.3. APS_EndpointId_t

Contains an 8-bit endpoint address.

```
typedef uint8_t APS_EndpointId_t;
```

6.3.4. HA_ProfileId_t

A profile identifier is 16-bits long and specifies the application profile being used. Refer to [4] section 2.5.1.1.

```
typedef uint16_t HA_ProfileId_t;
```

6.3.5. HA_ClusterId_t

A cluster identifier is 16-bits in length and specifies the set of related commands and attributes within a standard application profile. Refer to [4] section 2.5.1.3.

```
typedef uint16_t HA_ClusterId_t;
```

6.3.6. HA_ManufacturerSpecCode_t

The manufacturer code is 16-bits in length and specifies the ZigBee assigned manufacturer code for proprietary extensions to a profile. Refer to [4] section 2.3.1.2.

```
typedef uint16_t HA_ManufacturerSpecCode_t;
```

6.3.7. HA_AddressingInfo_t

Contains all necessary addressing information for data/command delivery.

```
typedef struct _HA_AddressingInfo_t
{
  APS_AddrMode_t  addrMode;
  APS_Address_t  addr;
  APS_EndpointId_t  endpoint;
  HA_ProfileId_t  profile;
  HA_ClusterId_t  cluster;
```

```
  HA_ManufacturerSpecCode_t  mSpecCode;
  bool   disableDefResp;
} HA_AddressingInfo_t;
```

6.3.8. HA_IndicationAddressingInfo_t

Type contains source addressing information and destination endpoint.

```
typedef struct _HA_IndicationAddressingInfo_t
{
  HA_AddressingInfo_t   srcAddrInfo;
  APS_EndpointId_t  dstEndpoint;
} HA_IndicationAddressingInfo_t;
```

6.3.9. HA_RequestAddressingInfo_t

Type contains destination addressing information and source endpoint.

```
typedef struct _HA_RequestAddressingInfo_t
{
  HA_AddressingInfo_t   dstAddrInfo;
  APS_EndpointId_t  srcEndpoint;
} HA_RequestAddressingInfo_t;
```

6.3.10. HA_AttributeId_t

Attribute identifier is 16-bits in length and is used to specify attribute within a cluster.

```
typedef uint16_t HA_AttributeId_t;
```

6.3.11. HA_AttributeData_t

This data type is used for describing complete attribute state.

```
typedef struct HA_AttributeData_t
{
  HA_AttributeId_t  attrId;
  HA_AttrDataType_t  type;
  union
  {
    bool                    dataBoolean;
    uint8_t                 dataUint8;
    uint8_t                 dataBitmap8;
    uint8_t                 dataEnum8;
    int8_t                  dataInt8;
    int16_t                 dataInt16;
```

```
    uint16_t                dataUint16;
    uint32_t                dataBitmap32;
    int32_t                 dataInt32;
    uint32_t                dataUint32;
    N_Security_Key_t        dataNetworkKey;
    N_Address_Extended_t    dataIEEE;
    uint8_t*                pOctetStr;
    char*                   dataCharStr;
    const char*             dataConstCharStr;
  };
} HA_AttributeData_t;
```

6.3.12. HA_Status_t

Enumeration contains possible result codes of command execution (from table 2.16 [4]).

```
typedef enum _HA_Status_t
{
  /** Operation was successful. */
    HA_SUCCESS_STATUS    = 0x00,
  /** Operation was unsuccessful. */
    HA_FAILED_STATUS    = 0x01,
    /** The sender of the command does not have authorization to carry
out this command. */
    HA_NOT_AUTHORIZED_STATUS  = 0x7e,
    /** A reserved field/subfield/bit contains a nonzero value. */
    HA_RESERVED_FIELD_NOT_ZERO_STATUS  = 0x7f,
    /** The command appears to contain the wrong fields, as detected
either by the      presence of one or more invalid field entries or by
there being missing fields.      Command not carried out. Implementer
has discretion as to whether to return      this error or
::HA_INVALID_FIELD_STATUS. */
    HA_MALFORMED_COMMAND_STATUS              = 0x80,
    /** The specified cluster command is not supported on the device.
Command not      carried out. */
    HA_UNSUP_CLUSTER_COMMAND_STATUS  = 0x81,
    /** The specified general ZCL command is not supported on the
device.*/
    HA_UNSUP_GENERAL_COMMAND_STATUS  = 0x82,
    /** A manufacturer-specific unicast, cluster specific command was
received with an unknown manufacturer code, or the manufacturer code
was recognized but      the command is not supported. */
    HA_UNSUP_MANUF_CLUSTER_COMMAND  = 0x83,
    /** A manufacturer-specific unicast, ZCL specific command was
received with an      unknown manufacturer code, or manufacturer code
was recognized but the command is not supported. */
    HA_UNSUP_MANUF_GENERAL_COMMAND_STATUS  = 0x84,
    /** At least one field of the command contains an incorrect value,
according to the specification the device is implemented to. */
    HA_INVALID_FIELD_STATUS                  = 0x85,
```

```
    /** The specified attribute does not exist on the device. */
    HA_UNSUPPORTED_ATTRIBUTE_STATUS          = 0x86,
    /** Out of range error, or set to a reserved value. Attribute keeps
its old value. */
    HA_INVALID_VALUE_STATUS                  = 0x87,
    /** Attempt to write a read-only attribute. */
    HA_READ_ONLY_STATUS                      = 0x88,
    /** An operation (e.g. an attempt to create an entry in a table)
failed due to an insufficient amount of free space available.*/
    HA_INSUFFICIENT_SPACE_STATUS             = 0x89,
    /** An attempt to create an entry in a table failed due to a
duplicate entry      already being present in the table. */
    HA_DUPLICATE_EXISTS_STATUS               = 0x8a,
    /** The requested information (e.g. table entry) be found. */
    HA_NOT_FOUND_STATUS                      = 0x8b,
    /** Periodic reports cannot be issued for this attribute. */
    HA_UNREPORTABLE_ATTRIBUTE_STATUS         = 0x8c,
    /** The data type given for an attribute is incorrect. Command not
carried out. */
    HA_INVALID_DATA_TYPE_STATUS              = 0x8d,
    /** The selector for an attribute is incorrect. */
    HA_INVALID_SELECTOR_STATUS               = 0x8e,
    /** A request has been made to read an attribute that the requester
is not      authorized to read. No action taken. */
    HA_WRITE_ONLY_STATUS                     = 0x8f,
    /** Setting the requested values would put the device in an
inconsistent state on startup. No action taken. */
    HA_INCONSISTENT_STARTUP_STATE_STATUS = 0x90,
    /** An attempt has been made to write an attribute that is present
but is defined using an out-of-band method and not over the air. */
    HA_DEFINED_OUT_OF_BAND_STATUS            = 0x91,

    /** An operation was unsuccessful due to a hardware failure. */
    HA_HARDWARE_FAILURE_STATUS               = 0xc0,
    /** An operation was unsuccessful due to a software failure. */
    HA_SOFTWARE_FAILURE_STATUS               = 0xc1,
    /** An error occurred during calibration. */
    HA_CALIBRATION_ERROR_STATUS              = 0xc2,
} HA_Status_t;
```

6.4. Management Services

6.4.1. Data Types

6.4.1.1. HA_RegisterEndpointReq_t

Type contains parameters of endpoint to be added and service data.

```
typedef struct _HA_RegisterEndpointReq_t
{
  SimpleDescriptor_t  simpleDescriptor;
  void (*HA_RegisterEndpointConf)(
                  HA_RegisterEndpointReq_t *origReq,
                  HA_Status_t status);
} HA_RegisterEndpointReq_t;
```

6.4.1.2. HA_UnregisterEndpointReq_t

Type contains parameters of endpoint to be removed.

```
typedef struct _HA_UnregisterEndpointReq_t
{
  APS_EndpointId_t       endpoint;
  void (*HA_UnregisterEndpointConf)(
                  HA_RegisterEndpointReq_t *origReq,
                  HA_Status_t status);
} HA_UnregisterEndpointReq_t;
```

6.4.1.3. HA_AttributeSubscriptionDescr_t

Type describes current attribute events description state.

```
typedef struct HA_AttributeSubscriptionDescr_t
{
  APS_EndpointId_t       endpoint;
  HA_ClusterId_t  cluster;
  HA_AttributeId_t  attr;
} HA_AttributeSubscriptionDescr_t;
```

6.4.1.4. HA_ChangeSubscriptionForAttributeEventsReq_t

Type contains parameters of attributes for which Read or/and Write access attempts will generate a notification to the application.

```
typedef struct _HA_ChangeSubscriptionForAttributeEventsReq_t
{
  uint8_t  descrListLength;
  HA_AttributeSubscriptionDescr_t      *descrList;
  void (*HA_ChangeSubscriptionForAttributeEventsConf)(
                  HA_SubscribeForAttributeEventsReq_t *origReq,
                  HA_Status_t status);
} HA_SubscribeForAttributeEventsReq_t;
```

6.4.2. Functions

6.4.2.1. HA_RegisterEndpointReq()

Prototype:

```
void HA_RegisterEndpointReq(HA_RegisterEndpointReq_t *req)
```

Description:

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

6.4.2.2. HA_UnregisterEndpointReq()

Prototype:

```
void HA_UnregisterEndpointReq(HA_UnregisterEndpointReq_t *req)
```

Description:

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

6.4.2.3. HA_ChangeSubscriptionForAttributeEventsReq()

Prototype:

```
void HA_ChangeSubscriptionForAttributeEventsReq(
                    HA_ChangeSubscriptionForAttributeEventsReq_t *req)
```

Description:

Changes current subscription for attribute Read/Write access events.

6.5. Foundation commands

6.5.1. Data types

6.5.1.1. HA_WriteAttributeRespData_t

Describes the result of Attribute Writing attempt.

```
typedef struct _HA_WriteAttributeRespData_t
{
  HA_Status_t   status;
  HA_AttributeId_t  attrId;
} HA_WriteAttributeRespData_t;
```

6.5.1.2. HA_WriteAttributesStructuredRespData_t

Describes the result of Structured Attribute writing operation (Refer to [4] section 2.4.17.1).

```
typedef struct _HA_WriteAttributesStructuredRespData_t
{
  HA_Status_t   status;
  HA_AttributeId_t  attrId;
  uint8_t  *selectorData;
} HA_WriteAttributesStructuredRespData_t;
```

6.5.1.3. HA_StructAttrDescr_t

Type contains descriptor of structured attribute to be used within Read Attributes Structured command (Refer to [4] sections 2.4.15.1.2, 2.4.15.1.3).

```
typedef struct _HA_StructAttrDescr_t
{
  HA_AttributeId_t  attrId;
  uint8_t  *selectorData;
} HA_StructAttrDescr_t;
```

6.5.1.4. HA_StructAttributeData_t

Type contains value to be written for structured attribute (Refer to [4] sections 2.4.16.1).

```
typedef struct _HA_StructAttributeData_t
{
  HA_AttributeId_t  attrId;
  uint8_t  *selectorData;
  HA_AttrDataType_t  type;
  uint8_t  *value;
} HA_StructAttributeData_t;
```

6.5.1.5. HA_ReadAttributeReq_t

Type describes parameters of Read Attribute command.

```
typedef struct _HA_ReadAttributeReq_t
{
  HA_RequestAddressingInfo_t   addrInfo;
  uint8_t  attrListLength;
  HA_AttributeId_t   *attrList;
  void (*HA_ReadAttributeConf)(
                  HA_ReadAttributeReq_t *origReq,
                  HA_Status_t status);
} HA_ReadAttributeReq_t;
```

6.5.1.6. HA_ReadAttributesStructuredReq_t

Type describes parameters of Read Attributes Structured command.

```
typedef struct _HA_ReadAttributesStructuredReq_t
{
  HA_RequestAddressingInfo_t   addrInfo;
  uint8_t  descrListLength;
  HA_StructAttrDescr_t   *descrList;
  void (*HA_ReadAttributesStructuredConf)(
                  HA_ReadAttributesStructuredReq_t *origReq,
                  HA_Status_t status);
} HA_ReadAttributesStructuredReq_t;
```

6.5.1.7. HA_ReadAttributeResp_t

Type describes parameters of Read Attribute Response command.

```
typedef struct _HA_ReadAttributeResp_t
{
  HA_RequestAddressingInfo_t  addrInfo;
  uint8_t  attrListLength;
  HA_AttributeData_t   *attrDataList;
} HA_ReadAttributeResp_t;
```

6.5.1.8. HA_AttributeAccessNtfy_t

Type describes parameters of attribute access notification.

```
typedef struct _HA_AttributeAccessNtfy_t
{
  HA_IndicationAddressingInfo_t  indAddrInfo;
  HA_RequestAddressingInfo_t  reqAddrInfo;
  uint8_t  attrListLength;
  HA_AttributeId_t   *attrList;
} HA_AttributeAccessNtfy_t;
```

6.5.1.9. HA_WriteAttributeReq_t

Type describes parameters of Write Attribute command.

```
typedef struct _HA_WriteAttributeReq_t
{
  HA_RequestAddressingInfo_t  addrInfo;
  uint8_t    attrListLength;
  HA_AttributeData_t  *attrList;
```

```
  void (*HA_WriteAttributeConf)(
                  HA_WriteAttributeReq_t *origReq,
                  HA_Status_t status);
} HA_WriteAttributeReq_t;
```

6.5.1.10. HA_WriteAttributesStructuredReq_t

Type describes parameters of Write Attributes Structured command.

```
typedef struct _HA_WriteAttributesStructuredReq_t
{
  HA_RequestAddressingInfo_t  addrInfo;
  uint8_t    attrListLength;
  HA_StructAttributeData_t  *attrList;
  void (*HA_WriteAttributesStructuredConf)(
                  HA_WriteAttributesStructuredReq_t *origReq,
                  HA_Status_t status);
} HA_WriteAttributesStructuredReq_t;
```

6.5.1.11. HA_WriteAttributeResp_t

Type describes parameters of Write Attribute Response command. The respPayload field may be only 1 byte in length in case if all attributes were written successfully (this byte shall have value - HA_SUCCESS_STATUS). Otherwise it contains an array of HA_WriteAttributeRespData_t structures.

```
typedef struct _HA_WriteAttributeResp_t
{
  HA_IndicationAddressingInfo_t  addrInfo;
  uint8_t  respLength;
  uint8_t  *respPayload;
} HA_WriteAttributeResp_t;
```

6.5.1.12. HA_WriteAttributesStructuredResp_t

Type describes parameters of Write Attributes Structured Response command. The respPayload field may be only 1 byte in length in case if all attributes were written successfully (this byte shall have value - HA_SUCCESS_STATUS). Otherwise it contains an array of HA_WriteAttributesStructuredRespData_t structures.

```
typedef struct _HA_WriteAttributesStructuredResp_t
{
  HA_IndicationAddressingInfo_t  addrInfo;
  uint8_t  respLength;
  uint8_t  *respPayload;
} HA_WriteAttributesStructuredResp_t;
```

6.5.1.13. HA_ReportTime_t

This type is intended to store time periods used by Configure Reporting command.

```
typedef uint16_t HA_ReportTime_t;
```

6.5.1.14. HA_ConfigureReportingReqElement_t

Type is used to form one record of Configure Reporting command (Refer to [4] section 2.4.7.1).

```
typedef struct _HA_ConfigureReportingReqElement_t
{
    uint8_t   direction;
    HA_AttributeId_t  attributeId;
    union
    {
        struct
        {
          HA_AttrDataType_t   attributeType;
          HA_ReportTime_t   minReportingInterval;
          HA_ReportTime_t   maxReportingInterval;
          uint8_t   *reportableChange;
        };
        HA_ReportTime_t   timeoutPeriod;
    };
} HA_ConfigureReportingReqElement_t;
```

6.5.1.15. HA_ConfigureReportingReq_t

Data type consists of the amount of the configuration records and the list of the configuration record (Refer to [4] section 2.4.7)

```
typedef struct _HA_ConfigureReportingReq_t
{
  HA_RequestAddressingInfo_t  addrInfo;
  uint8_t  elemListLength;
  HA_ConfigureReportingReqElement_t  *elemList;
  void (*HA_ConfigureReportingConf)(
                  HA_ConfigureReportingReq_t *origReq,
                  HA_Status_t status);
} HA_ConfigureReportingReq_t;
```

6.5.1.16. HA_ConfigureReportingStatus_t

This type contains the result of Configure Reporting command execution for separate attribute (Refer to [4] section 2.4.8.1).

```
typedef struct _HA_ConfigureReportingStatus_t
{
  /** Status of the configure reporting operation. */
  HA_Status_t            status;
  /** The direction field specifies whether values of the attribute are
be reported (0x00), or whether reports of the attribute are to be
received (0x01). */
    uint8_t             direction;
  /** Requested attribute's ID. */
    HA_AttributeId_t   attributeId;
} HA_ConfigureReportingStatus_t;
```

6.5.1.17. HA_ConfigureReportingResp_t

Type is intended to store parameters of the Configure Reporting Response command (Refer to [4] section 2.4.8).

```
typedef struct _HA_ConfigureReportingResp_t
{
  HA_IndicationAddressingInfo_t  addrInfo;
  uint8_t  statusListLength;
  HA_ConfigureReportingStatus_t  *statusList;
} HA_ConfigureReportingResp_t;
```

6.5.1.18. HA_ReadReportingConfigurationReqElem_t

Type contains payload element of Read Reporting Configuration command (Refer to [4] section 2.4.9.1)

```
typedef struct HA_ReadReportingConfigurationReqElem_t
{
  /** The direction field specifies whether values of the attribute are
be reported (0x00), or whether reports of the attribute are to be
received (0x01). */
    uint8_t             direction;
  /** Requested attribute identifier. */
    HA_AttributeId_t   attrId;
} HA_ReadReportingConfigurationReqElem_t;
```

6.5.1.19. HA_ReadReportingConfigurationReq_t

Type contains parameters for Read Reporting Configuration command.

```
typedef struct _HA_ReadReportingConfigurationReq_t
```

```
{
  HA_RequestAddressingInfo_t  addrInfo;
  uint8_t  elemListLength;
  HA_ReadReportingConfigurationReqElem_t   *elemList;
  void (*HA_ReadReportingConfigurationConf)(
                  HA_ReadReportingConfigurationReq_t *origReq,
                  HA_Status_t status);
} HA_ReadReportingConfigurationReq_t;
```

6.5.1.20. HA_ReadReportingConfigurationRespElement_t

Type is used to form one record of Read Reporting Configuration Response (Refer to [4] section 2.4.10.1).

```
typedef struct _HA_ReadReportingConfigurationRespElement_t
{
  HA_Status_t  status;
    uint8_t   direction;
    HA_AttributeId_t  attributeId;
    union
    {
        struct
        {
          HA_AttrDataType_t   attributeType;
          HA_ReportTime_t   minReportingInterval;
          HA_ReportTime_t   maxReportingInterval;
          uint8_t   *reportableChange;
        };
        HA_ReportTime_t   timeoutPeriod;
    };
} HA_ReadReportingConfigurationRespElement_t;
```

6.5.1.21. HA_ReadReportingConfigurationResp_t

Type is used to convey to the application level parameters of received Read Reporting Configuration Response command.

```
typedef struct _HA_ReadReportingConfigurationResp_t
{
  HA_IndicationAddressingInfo_t  addrInfo;
  uint8_t  elemListLength;
  HA_ReadReportingConfigurationRespElement_t   *elemList;
} HA_ReadReportingConfigurationResp_t;
```

6.5.1.22. HA_ReportAttributeInd_t

Type contains parameters of received Report Attributes command (Refer to [4] section 2.4.11).

```
typedef struct HA_ReportAttributeInd_t
{
  HA_IndicationAddressingInfo_t  addrInfo;
  uint8_t  attrListLength;
  HA_AttributeData_t   *attrList;
} HA_ReportAttributeInd_t;
```

6.5.1.23. HA_DefaultResp_t

Type contains parameters of received Default Response (Refer to [4] section 2.4.12).

```
typedef struct HA_DefaultResp_t
{
  HA_IndicationAddressingInfo_t  addrInfo;
  uint8_t  cmdId;
  HA_Status_t  status;
} HA_DefaultResp_t;
```

6.5.1.24. HA_DiscoverAttributesReq_t

Type is used to form Discover Attributes Request command (Refer to [4] section 2.4.13).

```
typedef struct HA_DiscoverAttributesReq_t
{
  HA_RequestAddressingInfo_t  addrInfo;
  HA_AttributeId_t  startAttrId;
  uint8_t  maxAttrIds;
  void (*HA_DiscoverAttributesConf)(
                  HA_DiscoverAttributesReq_t *origReq,
                  HA_Status_t status);
} HA_DiscoverAttributesReq_t;
```

6.5.1.25. HA_DiscoverAttributesRespElem_t

Type contains description of one separate attribute within Discover Attributes Response command.

```
typedef struct HA_DiscoverAttributesRespElem_t
{
  HA_AttributeId_t  attrId;
  HA_AttrDataType_t  type;
} HA_DiscoverAttributesRespElem_t;
```

6.5.1.26. HA_DiscoverAttributesResp_t

Type describes format of indication to the application level with parameters of Discover Attributes Response parameters.

```
typedef struct HA_DiscoverAttributesResp_t
{
  HA_IndicationAddressingInfo_t  addrInfo;
  uint8_t  discoveryComplete;
  uint8_t  elemListLength;
  HA_DiscoverAttributesRespElem_t *elemList;
} HA_DiscoverAttributesResp_t;
```

6.5.2. Functions

6.5.2.1. HA_ReadAttributeReq()

Prototype:

```
void HA_ReadAttributeReq(HA_ReadAttributeReq_t *req)
```

Description:

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

6.5.2.2. HA_ReadAttributesStructuredReq()

Prototype:

```
void HA_ReadAttributesStructuredReq(HA_ReadAttributesStructuredReq_t
*req)
```

Description:

Sends Read Attributes Structured request (Refer to [4] section 2.4.15).

6.5.2.3. HA_ReadAttributeRespInd()

Prototype:

```
void HA_ReadAttributeRespInd(HA_ReadAttributeResp_t *resp)
```

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.

6.5.2.4. HA_ReadAttributeNtfyInd()

Prototype:

```
void HA_ReadAttributeNtfyInd(HA_AttributeAccessNtfy_t *ntfy)
```

Description:
The indication about the fact that an attribute was read. This primitive is intended to notify the Host application about read access to HA attributes. If some attribute will be read by remote device, the HA_ReadAttributeNtfyInd() will be issued. It will contain a list of attribute identifiers and addressing information

6.5.2.5. HA_WriteAttributeReq()

Prototype:
```
void HA_WriteAttributeReq(HA_WriteAttributeReq_t *req)
```

Description:
Sends Write Attribute request (Refer to [4] section 2.4.3).

6.5.2.6. HA_WriteAttributeUndividedReq()

Prototype:
```
void HA_WriteAttributeUndividedReq(HA_WriteAttributeReq_t *req)
```

Description:
Sends Write Attribute Undivided request (Refer to [4] section 2.4.4).

6.5.2.7. HA_WriteAttributeNoReponseReq()

Prototype:
```
void HA_WriteAttributeNoResponseReq(HA_WriteAttributeReq_t *req)
```

Description:
Sends Write Attribute No Response request (Refer to [4] section 2.4.6).

6.5.2.8. HA_WriteAttributesStructuredReq()

Prototype:
```
void HA_WriteAttributesStructuredReq(HA_WriteAttributesStructuredReq_t
*req)
```

Description:
Sends Write Attributes Structured request (Refer to [4] section 2.4.16).

6.5.2.9. HA_WriteAttributeNtfyInd()

Prototype:
```
void HA_WriteAttributeNtfyInd(HA_AttributeAccessNtfy_t *ind)
```

Description:

The indication about the fact that an attribute was written. This primitive is intended to notify the Host application about write access to HA attributes. If some attribute will be written by remote device, HA_WriteAttributeNtfyInd() will be raised with following parameters:

- Addressing information describing the remote device.
- List of attributes which were written with their new values.

6.5.2.10. HA_WriteAttributeRespInd()

Prototype:

```
void HA_WriteAttributeRespInd(HA_WriteAttributeResp_t *resp)
```

Description:

Primitive is used to send an indication to the application level when Write Attribute Response command is received (Refer to [4] section 2.4.5). Should be implemented on application level.

6.5.2.11. HA_WriteAttributesStructuredRespInd()

Prototype:

```
void
HA_WriteAttributesStructuredRespInd(HA_WriteAttributesStructuredResp_t
*resp)
```

Description:

Primitive is used to send an indication to the application level when Write Attributes Structured Response command is received (Refer to [4] section 2.4.17). Should be implemented on application level.

6.5.2.12. HA_ConfigureReportingReq()

Prototype:

```
void HA_ConfigureReportingReq(HA_ConfigureReportingReq_t *req)
```

Description:

Sends Configure Reporting request (Refer to [4] section 2.4.7).

6.5.2.13. HA_ConfigureReportingRespInd()

Prototype:

```
void HA_ConfigureReportingRespInd(HA_ConfigureReportingResp_t *resp)
```

Description:
Primitive is used to send an indication to the application level when Configure Reporting Response command is received (Refer to [4] section 2.4.8). Should be implemented on application level.

6.5.2.14. HA_ReadReportingConfigurationReq()

Prototype:
```
void
HA_ReadReportingConfigurationReq(HA_ReadReportingConfigurationReq_t
*ind)
```

Description:
Sends read reporting configuration request (Refer to [4] section 2.4.9).

6.5.2.15. HA_ReadReportingConfigurationRespInd()

Prototype:
```
void
HA_ReadReportingConfigurationRespInd(HA_ReadReportingConfigurationResp_
t *resp)
```

Description:
Primitive is used to send an indication to the application level when Read Reporting Configuration Response command is received (Refer to [4] section 2.4.10). Should be implemented on application level.

6.5.2.16. HA_ReportAttributeInd()

Prototype:
```
void HA_ReportAttributeInd(HA_ReportAttributeInd_t *ind)
```

Description:
The indication about the fact that an attribute was reported.

6.5.2.17. HA_DefaultRespInd()

Prototype:
```
void HA_DefaultRespInd(HA_DefaultResp_t *resp)
```

Description:
Primitive is used to send an indication to the application level when Default Response command is received (Refer to [4] section 2.4.12). Should be implemented on application level.

6.5.2.18. HA_DiscoverAttributesReq()

Prototype:

```
void HA_DiscoverAttributesReq(HA_DiscoverAttributesReq_t *req)
```

Description:

Sends discover attributes request (Refer to [4] section 2.4.13).

6.5.2.19. HA_DiscoverAttributesRespInd()

Prototype:

```
void HA_DiscoverAttributesRespInd(HA_DiscoverAttributesResp_t *resp)
```

Description:

Primitive is used to send an indication to the application level when Discover Attributes Response command is received (Refer to [4] section 2.4.14). Should be implemented on application level.

6.6. Basic Cluster Server

There are no supported commands for this cluster.

6.7. On/Off Cluster Client

6.7.1. Data Types

6.7.1.1. HA_OnReq_t

Type contains parameters of the On command and addressing information.

```
typedef struct _HA_OnReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  void (*HA_OnConf)(HA_OnReq_t *origReq, HA_Status_t status);
} HA_OnReq_t;
```

6.7.1.2. HA_OffReq_t

Type contains parameters of the Off command and addressing information

```
typedef struct _HA_OffReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  void (*HA_OffConf)(HA_OffReq_t *origReq, HA_Status_t status);
} HA_OffReq_t;
```

6.7.1.3. HA_ToggleReq_t

Type contains parameters of the Toggle command and addressing information.

```
typedef struct _HA_ToggleReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  void (*HA_ToggleConf)(
                  HA_ToggleReq_t *origReq,
                  HA_Status_t status);
} HA_ToggleReq_t;
```

6.7.2. Functions

6.7.2.1. HA_OnOffClusterOnReq()

Prototype:

```
void HA_OnOffClusterOnReq(HA_OnReq_t *req)
```

Description:

Sends On command to the destination device.

6.7.2.2. HA_OnOffClusterOffReq()

Prototype:

```
void HA_OnOffClusterOffReq(HA_OffReq_t *req)
```

Description:

Sends Off command to the destination device.

6.7.2.3. HA_OnOffClusterToggleReq()

Prototype:

```
void HA_OnOffClusterToggleReq(HA_ToggleReq_t *req)
```

Description:

Sends Toggle command to the destination device.

6.8. Scenes Cluster Client

6.8.1. Data Types

6.8.1.1. HA_AddSceneReq_t

Type contains parameters of the Add Scene command and addressing information.

```
typedef struct _HA_AddSceneReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.4.1.1. */
  HA_AddScene_t   commandParameters;
  void (*HA_AddSceneConf)(
                  HA_AddSceneReq_t *origReq,
                  HA_Status_t status);
} HA_AddSceneReq_t;
```

6.8.1.2. HA_ViewSceneReq_t

Type contains parameters of the View Scene command and addressing information.

```
typedef struct _HA_ViewSceneReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.4.2.1. */
  HA_ViewScene_t   commandParameters;
  void (*HA_ViewSceneConf)(
                  HA_ViewSceneReq_t *origReq,
                  HA_Status_t status);
} HA_ViewSceneReq_t;
```

6.8.1.3. HA_RemoveSceneReq_t

Type contains parameters of the Remove Scene command and addressing information.

```
typedef struct _HA_RemoveSceneReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.4.3.1. */
  HA_RemoveScene_t   commandParameters;
  void (*HA_RemoveSceneConf)(
                  HA_RemoveSceneReq_t *origReq,
                  HA_Status_t status);
} HA_RemoveSceneReq_t;
```

6.8.1.4. HA_RemoveAllScenesReq_t

Type contains parameters of the Remove All Scene command and addressing information.

```
typedef struct _HA_RemoveAllScenesReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.4.4.1. */
  HA_RemoveAllScenes_t   commandParameters;
  void (*HA_RemoveAllScenesConf)(
                  HA_RemoveAllScenesReq_t *origReq,
                  HA_Status_t status);
} HA_RemoveAllScenesReq_t;
```

6.8.1.5. HA_StoreSceneReq_t

Type contains parameters of the Store Scene command and addressing information.

```
typedef struct _HA_StoreSceneReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.4.5.1. */
  HA_StoreScene_t   commandParameters;
  void (*HA_StoreSceneConf)(
                  HA_StoreSceneReq_t *origReq,
                  HA_Status_t status);
} HA_StoreSceneReq_t;
```

6.8.1.6. HA_RecallSceneReq_t

Type contains parameters of the Recall Scene command and addressing information.

```
typedef struct _HA_RecallSceneReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.4.6.1. */
  HA_RecallScene_t   commandParameters;
  void (*HA_RecallSceneConf)(
                  HA_RecallSceneReq_t *origReq,
                  HA_Status_t status);
} HA_RecallSceneReq_t;
```

6.8.1.7. HA_GetSceneMembershipReq_t

Type contains parameters of the Get Scene Membership command and addressing information.

```
typedef struct _HA_GetSceneMembershipReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.4.7.1. */
  HA_GetSceneMembership_t   commandParameters;
  void (*HA_GetSceneMembershipConf)(
                  HA_GetSceneMembershipReq_t *origReq,
                  HA_Status_t status);
} HA_GetSceneMembershipReq_t;
```

6.8.1.8. HA_AddSceneResponseInd_t

Type contains parameters of the Add Scene Response command and addressing information.

```
typedef struct _HA_AddSceneResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.5.1.1. */
  HA_AddSceneResponse_t   commandParameters;
} HA_AddSceneResponseInd_t;
```

6.8.1.9. HA_ViewSceneResponseInd_t

Type contains parameters of the View Scene Response command and addressing information.

```
typedef struct _HA_ViewSceneResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.5.2.1. */
  HA_ViewSceneResponse_t   commandParameters;
} HA_ViewSceneResponseInd_t;
```

6.8.1.10. HA_RemoveSceneResponseInd_t

Type contains parameters of the Remove Scene Response command and addressing information.

```
typedef struct _HA_RemoveSceneResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.5.3.1. */
  HA_RemoveSceneResponse_t   commandParameters;
```

```
} HA_RemoveSceneResponseInd_t;
```

6.8.1.11. HA_RemoveAllScenesResponseInd_t

Type contains parameters of the Remove All Scenes Response command and addressing information.

```
typedef struct _HA_RemoveAllScenesResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.5.4.1. */
  HA_RemoveAllScenesResponse_t   commandParameters;
} HA_RemoveAllScenesResponseInd_t;
```

6.8.1.12. HA_StoreSceneResponseInd_t

Type contains parameters of the Store Scene Response command and addressing information.

```
typedef struct _HA_StoreSceneResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.5.5.1. */
  HA_StoreSceneResponse_t   commandParameters;
} HA_StoreSceneResponseInd_t;
```

6.8.1.13. HA_GetSceneMembershipResponseInd_t

Type contains parameters of the Get Scene Membership Response command and addressing information.

```
typedef struct _HA_GetSceneMembershipResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.7.2.5.6.1. */
  HA_GetSceneMembershipResponse_t   commandParameters;
} HA_GetSceneMembershipResponseInd_t;
```

6.8.2. Functions

6.8.2.1. HA_ScenesClusterAddSceneReq()

Prototype:

```
void HA_ScenesClusterAddSceneReq(HA_AddSceneReq_t *req)
```

Description:
Sends Add Scene command to the destination device.

6.8.2.2. HA_ScenesClusterViewSceneReq()

Prototype:
```
void HA_ScenesClusterViewSceneReq(HA_ViewSceneReq_t *req)
```

Description:
Sends View Scene command to the destination device.

6.8.2.3. HA_ScenesClusterRemoveSceneReq()

Prototype:
```
void HA_ScenesClusterRemoveSceneReq(HA_RemoveSceneReq_t *req)
```

Description:
Sends Remove Scene command to the destination device.

6.8.2.4. HA_ScenesClusterRemoveAllScenesReq()

Prototype:
```
void HA_ScenesClusterRemoveAllScenesReq(HA_RemoveAllScenesReq_t *req)
```

Description:
Sends Remove All Scenes command to the destination device.

6.8.2.5. HA_ScenesClusterStoreSceneReq()

Prototype:
```
void HA_ScenesClusterStoreSceneReq(HA_StoreSceneReq_t *req)
```

Description:
Sends Store Scene command to the destination device.

6.8.2.6. HA_ScenesClusterRecallSceneReq()

Prototype:
```
void HA_ScenesClusterRecallSceneReq(HA_RecallSceneReq_t *req)
```

Description:
Sends Recall Scene command to the destination device.

6.8.2.7. HA_ScenesClusterGetGroupMembershipSceneReq()

Prototype:

```
void HA_ScenesClusterGetSceneMembershipReq(HA_GetSceneMembershipReq_t
*req)
```

Description:

Sends Get Scene Membership command to the destination device.

6.8.2.8. HA_ScenesClusterAddSceneResponseInd()

Prototype:

```
void HA_ScenesClusterAddSceneResponseInd(HA_AddSceneResponseInd_t *ind)
```

Description:

Indicates parameters of received Add Scene Response command. Should be defined on the application level.

6.8.2.9. HA_ScenesClusterViewSceneResponseInd()

Prototype:

```
void HA_ScenesClusterViewSceneResponseInd(
                    HA_ViewSceneResponseInd_t *ind)
```

Description:

Indicates parameters of received View Scene Response command. Should be defined on the application level.

6.8.2.10. HA_ScenesClusterRemoveSceneResponseInd()

Prototype:

```
void HA_ScenesClusterRemoveSceneResponseInd(
                    HA_RemoveSceneResponseInd_t *ind)
```

Description:

Indicates parameters of received Remove Scene Response command. Should be defined on the application level.

6.8.2.11. HA_ScenesClusterRemoveAllScenesResponseInd()

Prototype:

```
void HA_ScenesClusterRemoveAllScenesResponseInd(
                    HA_RemoveAllScenesResponseInd_t *ind)
```

Description:
Indicates parameters of received Remove All Scenes Response command. Should be defined on the application level.

6.8.2.12. HA_ScenesClusterStoreSceneResponseInd()

Prototype:
```
void HA_ScenesClusterStoreSceneResponseInd(
                    HA_StoreSceneResponseInd_t *ind)
```

Description:
Indicates parameters of received Store Scene Response command. Should be defined on the application level.

6.8.2.13. HA_ScenesClusterGetSceneMembershipResponseInd()

Prototype:
```
void HA_ScenesClusterGetSceneMembershipResponseInd(
                    HA_GetSceneMembershipResponseInd_t *ind)
```

Description:
Indicates parameters of received Get Scene Membership Response command. Should be defined on the application level.

6.9. Identify Cluster Client

6.9.1. Data Types

6.9.1.1. HA_IdentifyReq_t

Type contains parameters of the Identify command and addressing information.

```
typedef struct _HA_IdentifyReq_t
{
  HA_AddressingInfo_t   dstAddrInfo;
  APS_EndpointId_t      srcEndpoint;
  /** Command parameters as described in [4] section 3.5.2.3.1.1. */
  HA_Identify_t   commandParameters;
  void (*HA_IdentifyConf)(
                    HA_IdentifyReq_t *origReq,
                    HA_Status_t status);
} HA_IdentifyReq_t;
```

6.9.1.2. HA_IdentifyQueryReq_t

Type contains parameters of the Identify Query command and addressing information.

```
typedef struct _HA_IdentifyQueryReq_t
{
  HA_AddressingInfo_t   dstAddrInfo;
  APS_EndpointId_t      srcEndpoint;
  /** There are no special parameters for this command */
  void (*HA_IdentifyQueryConf)(
                    HA_IdentifyQueuryReq_t *origReq,
                    HA_Status_t status);
} HA_IdentifyQueryReq_t;
```

6.9.1.3. HA_IdentifyQueryResponseInd_t

Type contains parameters of the Identify Query Response command and addressing information.

```
typedef struct _HA_IdentifyQueryResponseInd_t
{
  HA_AddressingInfo_t   srcAddrInfo;
  APS_EndpointId_t      dstEndpoint;
  /** Command parameters as described in [4] section 3.5.2.4.1.1. */
  HA_IdentifyQueryResponse_t   commandParameters;
} HA_IdentifyQueryResponseInd_t;
```

6.9.2. Functions

6.9.2.1. HA_IdentifyClusterIdentifyReq()

Prototype:

```
void HA_IdentifyClusterIdentifyReq(HA_IdentifyReq_t *req)
```

Description:

Sends Identify command to the destination device.

6.9.2.2. HA_IdentifyClusterIdentifyQueryReq()

Prototype:

```
void HA_IdentifyClusterIdentifyQueryReq(HA_IdentifyQueryReq_t *req)
```

Description:

Sends Identify Query command to the destination device.

6.9.2.3. HA_IdentifyClusterIdentifyQueryResponseInd()

Prototype:

```
void HA_IdentifyClusterIdentifyQueryResponseInd(
                    HA_IdentifyQueryResponseInd_t *ind)
```

Description:

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

6.10. Identify Cluster Server

6.10.1. Data Types

6.10.1.1. HA_IdentifyQueryResponseReq_t

Type contains parameters of the Identify Query Response command and addressing information.

```
typedef struct _HA_IdentifyQueryResponseReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.5.2.4.1.1. */
  HA_IdentifyQueryResponse_t   commandParameters;
  void (*HA_IdentifyQueryResponseConf)(
                    HA_IdentifyQueuryResponseReq_t *origReq,
                    HA_Status_t status);
} HA_IdentifyQueryResponseReq_t;
```

6.10.1.2. HA_IdentifyInd_t

Type contains parameters of the received Identify command.

```
typedef struct _HA_IdentifyInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.5.2.3.1.1. */
  HA_Identify_t   commandParameters;
} HA_IdentifyInd_t;
```

6.10.2. Functions

6.10.2.1. HA_IdentifyClusterIdentifyQueryResponseReq()

Prototype:

```
void HA_IdentifyClusterIdentifyQueryResponseReq(
                    HA_IdentifyQueryResponseReq_t *req)
```

Description:

Sends Identify Query Response command to the destination device.

6.10.2.2. HA_IdentifyClusterIdentifyInd()

Prototype:

```
void HA_IdentifyClusterIdentifyInd(HA_IdentifyInd_t *ind)
```

Description:

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

6.10.2.3. HA_IdentifyClusterIdentifyQueryInd()

Prototype:

```
void HA_IdentifyClusterIdentifyQueryInd(
                    HA_IndicationAddressingInfo_t    *addrInfo)
```

Description:

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

6.11. Groups Cluster Client

6.11.1. Data Types

6.11.1.1. HA_AddGroupReq_t

Type contains parameters of the Add Group command and addressing information.

```
typedef struct _HA_AddGroupReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
```

```
  /** Command parameters as described in [4] section 3.6.2.2.3.1. */
  HA_AddGroup_t   commandParameters;
  void (*HA_AddGroupConf)(
                  HA_AddGroupReq_t *origReq,
                  HA_Status_t status);
} HA_AddGroupReq_t;
```

6.11.1.2. HA_ViewGroupReq_t

Type contains parameters of the View Group command and addressing information.

```
typedef struct _HA_ViewGroupReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.6.2.2.4.1. */
  HA_ViewGroup_t   commandParameters;
  void (*HA_ViewGroupConf)(
                  HA_ViewGroupReq_t *origReq,
                  HA_Status_t status);
} HA_ViewGroupReq_t;
```

6.11.1.3. HA_GetGroupMembershipReq_t

Type contains parameters of the Get Group Membership command and addressing information.

```
typedef struct _HA_GetGroupMembershipReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.6.2.2.5.1. */
  HA_GetGroupMembership_t   commandParameters;
  void (*HA_GetGroupMembershipConf)(
                  HA_GetGroupMembershipReq_t *origReq,
                  HA_Status_t status);
} HA_GetGroupMembershipReq_t;
```

6.11.1.4. HA_RemoveGroupReq_t

Type contains parameters of the Remove Group command and addressing information.

```
typedef struct _HA_RemoveGroupReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.6.2.2.6.1. */
  HA_RemoveGroup_t   commandParameters;
  void (*HA_RemoveGroupConf)(
                  HA_RemoveGroupReq_t *origReq,
                  HA_Status_t status);
```

```
} HA_RemoveGroupReq_t;
```

6.11.1.5. HA_AddGroupResponseInd_t

Type contains parameters of the Add Group Response command and addressing information.

```
typedef struct _HA_AddGroupResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.6.2.4.1.1. */
  HA_AddGroupResponse_t   commandParameters;
} HA_AddGroupResponseInd_t;
```

6.11.1.6. HA_ViewGroupResponseInd_t

Type contains parameters of the View Group Response command and addressing information.

```
typedef struct _HA_ViewGroupResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.6.2.4.2.1. */
  HA_ViewGroupResponse_t   commandParameters;
} HA_ViewGroupResponseInd_t;
```

6.11.1.7. HA_GetGroupMembershipResponseInd_t

Type contains parameters of the Get Group Membership Response command and addressing information.

```
typedef struct _HA_GetGroupMembershipResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 3.6.2.4.3.1. */
  HA_GetGroupMembershipResponse_t   commandParameters;
} HA_GetGroupMembershipResponseInd_t;
```

6.11.1.8. HA_RemoveGroupResponseInd_t

Type contains parameters of the Remove Group Response command and addressing information.

```
typedef struct _HA_RemoveGroupResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
```

```
    /** Command parameters as described in [4] section 3.6.2.4.4.1. */
    HA_RemoveGroupResponse_t   commandParameters;
  } HA_RemoveGroupResponseInd_t;
```

6.11.2. Functions

6.11.2.1. HA_GroupsClusterAddGroupReq()

Prototype:

```
void HA_GroupsClusterAddGroupReq(HA_AddGroupReq_t *req)
```

Description:

Sends Add Group command to the destination device.

6.11.2.2. HA_GroupsClusterViewGroupReq()

Prototype:

```
void HA_GroupsClusterViewGroupReq(HA_ViewGroupReq_t *req)
```

Description:

Sends View Group command to the destination device.

6.11.2.3. HA_GroupsClusterGetGroupMembershipGroupReq()

Prototype:

```
void HA_GroupsClusterGetGroupMembershipReq(
                    HA_GetGroupMembershipReq_t *req)
```

Description:

Sends Get Group Membership command to the destination device.

6.11.2.4. HA_GroupsClusterRemoveGroupReq()

Prototype:

```
void HA_GroupsClusterRemoveGroupReq(HA_RemoveGroupReq_t *req)
```

Description:

Sends Remove Group command to the destination device.

6.11.2.5. HA_GroupsClusterRemoveAllGroupsReq()

Prototype:

```
void HA_GroupsClusterRemoveAllGroupsReq(
                    HA_RequestAddressingInfo_t *addrInfo)
```

Description:
Sends Remove All Groups command to the destination device.

6.11.2.6. HA_GroupsClusterAddGroupIfIdentifyingReq()

Prototype:
```
void HA_GroupsClusterStoreGroupReq(HA_AddGroupReq_t *req)
```

Description:
Sends Add Group If Identifying command to the destination device.

6.11.2.7. HA_GroupsClusterAddGroupResponseInd()

Prototype:
```
void HA_GroupsClusterAddGroupResponseInd(
                    HA_AddGroupResponseInd_t *ind)
```

Description:
Indicates parameters of received Add Group Response command. Should be defined on the application level.

6.11.2.8. HA_GroupsClusterViewGroupResponseInd()

Prototype:
```
void HA_GroupsClusterViewGroupResponseInd(
                    HA_ViewGroupResponseInd_t *ind)
```

Description:
Indicates parameters of received View Group Response command. Should be defined on the application level.

6.11.2.9. HA_GroupsClusterGetGroupMembershipResponseInd()

Prototype:
```
void HA_GroupsClusterGetGroupMembershipResponseInd(
                    HA_GetGroupMembershipResponseInd_t *ind)
```

Description:
Indicates parameters of received Get Group Membership Response command. Should be defined on the application level.

6.11.2.10. HA_GroupsClusterRemoveGroupResponseInd()

Prototype:

```
void HA_GroupsClusterRemoveGroupResponseInd(
                    HA_RemoveGroupResponseInd_t *ind)
```

Description:

Indicates parameters of received Remove Group Response command. Should be defined on the application level.

6.12. Door Lock Cluster Client

6.12.1. Data Types

6.12.1.1. HA_LockDoorReq_t

Type contains parameters of the Lock Door command and addressing information.

```
typedef struct _HA_LockDoorReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.1.2.15.1. */
  HA_LockDoor_t   commandParameters;
  void (*HA_LockDoorConf)(
                  HA_LockDoorReq_t *origReq,
                  HA_Status_t status);
} HA_LockDoorReq_t;
```

6.12.1.2. HA_UnlockDoorReq_t

Type contains parameters of the Unlock Door command and addressing information.

```
typedef struct _HA_UnlockDoorReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.1.2.15.2. */
  HA_UnlockDoor_t   commandParameters;
  void (*HA_UnlockDoorConf)(
                  HA_UnlockDoorReq_t *origReq,
                  HA_Status_t status);
} HA_UnlockDoorReq_t;
```

6.12.1.3. HA_LockDoorResponseInd_t

Type contains parameters of the Lock Door Response command and addressing information.

```
typedef struct _HA_LockDoorResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.1.2.16.1. */
  HA_Status_t   status;
} HA_LockDoorResponseInd_t;
```

6.12.1.4. HA_UnlockDoorResponseInd_t

Type contains parameters of the Unlock Door Response command and addressing information.

```
typedef struct _HA_UnlockDoorResponseInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.1.2.16.2. */
  HA_Status_t   status;
} HA_UnlockDoorResponseInd_t;
```

6.12.2. Functions

6.12.2.1. HA_DoorLockClusterLockDoorReq()

Prototype:

```
void HA_DoorLockClusterLockDoorReq(HA_LockDoorReq_t *req)
```

Description:

Sends Lock Door command to the destination device.

6.12.2.2. HA_DoorLockClusterUnlockDoorReq()

Prototype:

```
void HA_DoorLockClusterUnlockDoorReq(HA_UnlockDoorReq_t *req)
```

Description:

Sends Unlock Door command to the destination device.

6.12.2.3. HA_DoorLockClusterLockDoorResponseInd()

Prototype:

```
void HA_DoorLockClusterLockDoorResponseInd(
                     HA_LockDoorResponseInd_t *ind)
```

Description:

Indicates parameters of received Lock Door Response command. Should be defined on the application level.

6.12.2.4. HA_DoorLockClusterUnlockDoorResponseInd()

Prototype:

```
void HA_DoorLockClusterUnlockDoorResponseInd(
                    HA_UnlockDoorResponseInd_t *ind)
```

Description:

Indicates parameters of received Unlock Door Response command. Should be defined on the application level.

6.13. Level Control Cluster Client

6.13.1. Data Types

6.13.1.1. HA_MoveToLevelReq_t

Type contains parameters of the Move To Level command and addressing information.

```
typedef struct _HA_MoveToLevelReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  bool   withOnOffEffect;
  /** Command parameters as described in [4] section 3.10.2.3.1.1. */
  HA_MoveToLevel_t   commandParameters;
  void (*HA_MoveToLevelConf)(
                    HA_MoveToLevelReq_t *origReq,
                    HA_Status_t status);
} HA_MoveToLevelReq_t
```

6.13.1.2. HA_MoveLevelReq_t

Type contains parameters of the Move command and addressing information.

```
typedef struct _HA_MoveLevelReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  bool   withOnOffEffect;
  /** Command parameters as described in [4] section 3.10.2.3.2.1. */
  HA_MoveLevel_t   commandParameters;
  void (*HA_MoveLevelConf)(
                  HA_MoveLevelReq_t *origReq,
                  HA_Status_t status);
} HA_MoveLevelReq_t
```

6.13.1.3. HA_StepLevelReq_t

Type contains parameters of the Step command and addressing information.

```
typedef struct _HA_StepLevelReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  bool   withOnOffEffect;
  /** Command parameters as described in [4] section 3.10.2.3.3.1. */
  HA_StepLevel_t   commandParameters;
  void (*HA_StepLevelConf)(
                  HA_StepLevelReq_t *origReq,
                  HA_Status_t status);
} HA_StepLevelReq_t
```

6.13.1.4. HA_StopReq_t

Type contains parameters of the Step command and addressing information.

```
typedef struct _HA_StopReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** There are no special parameters for this command. */
  void (*HA_StopConf)(
                  HA_StopReq_t *origReq,
                  HA_Status_t status);
} HA_StopReq_t
```

6.13.2. Functions

6.13.2.1. HA_LevelControlClusterMoveToLevelReq()

Prototype:

```
void HA_LevelControlClusterMoveToLevelReq(HA_MoveToLevelReq_t *req)
```

Description:
Sends Move To Level command to the destination device.

6.13.2.2. HA_LevelControlClusterMoveReq()

Prototype:
```
void HA_LevelControlClusterMoveReq(HA_MoveLevelReq_t *req)
```

Description:
Sends Move command to the destination device.

6.13.2.3. HA_LevelControlClusterStepReq()

Prototype:
```
void HA_LevelControlClusterStepReq(HA_Step_t *cmd)
```

Description:
Sends Step command to the destination device.

6.13.2.4. HA_LevelControlClusterStopReq()

Prototype:
```
void HA_LevelControlClusterStopReq(HA_StopReq_t *req)
```

Description:
Sends Stop command to the destination device.

6.14. Window Covering Cluster Client

6.14.1. Data Types

6.14.1.1. HA_UpOpenReq_t

Type contains parameters of the Up/Open command and addressing information.

```
typedef struct _HA_UpOpenReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** There are no special parameters for this command. */
  void (*HA_UpOpenConf)(
                  HA_UpOpenReq_t *origReq,
```

```
                    HA_Status_t status);
} HA_UpOpenReq_t
```

6.14.1.2. HA_DownCloseReq_t

Type contains parameters of the Down/Close command and addressing information.

```
typedef struct _HA_DownCloseReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** There are no special parameters for this command. */
  void (*HA_DownCloseConf)(
                  HA_DownCloseReq_t *origReq,
                  HA_Status_t status);
} HA_DownCloseReq_t
```

6.14.1.3. HA_StopReq_t

Type contains parameters of the Stop command and addressing information.

```
typedef struct _HA_StopReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** There are no special parameters for this command. */
  void (*HA_StopConf)(
                  HA_StopReq_t *origReq,
                  HA_Status_t status);
} HA_DownCloseReq_t
```

6.14.2. Functions

6.14.2.1. HA_WindowCoveringClusterUpOpenReq()

Prototype:

```
void HA_WindowCoveringClusterUpOpenReq(HA_UpOpenReq_t *req)
```

Description:

Sends Up/Open command to the destination device.

6.14.2.2. HA_WindowCoveringClusterDownCloseReq()

Prototype:

```
void HA_WindowCoveringClusterDownCloseReq(HA_DownCloseReq_t *req)
```

Description:

Sends Down/Close command to the destination device.

6.14.2.3. HA_WindowCoveringClusterStopReq()

Prototype:

```
void HA_WindowCoveringClusterStopReq(HA_StopReq_t *req)
```

Description:

Sends Stop command to the destination device.

6.15. Color Control Cluster Client

6.15.1. Data Types

6.15.1.1. HA_MoveToColorReq_t

Type contains parameters of the Move To Color command and addressing information.

```
typedef struct _HA_MoveToColorReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 5.2.2.3.9.1. */
  HA_MoveToColor_t   commandParameters;
  void (*HA_MoveToColorConf)(
                  HA_MoveToColorReq_t *origReq,
                  HA_Status_t status);
} HA_MoveToColorReq_t
```

6.15.1.2. HA_MoveColorReq_t

Type contains parameters of the Move To Color command and addressing information.

```
typedef struct _HA_MoveColorReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 5.2.2.3.10.1. */
  HA_MoveColor_t   commandParameters;
  void (*HA_MoveColorConf)(
                  HA_MoveColorReq_t *origReq,
                  HA_Status_t status);
} HA_MoveColorReq_t;
```

6.15.1.3. HA_StepColorReq_t

Type contains parameters of the Move To Color command and addressing information.

```
typedef struct _HA_StepColorReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 5.2.2.3.11.1. */
  HA_StepColor_t   commandParameters;
  void (*HA_StepColorConf)(
                  HA_StepColorReq_t *origReq,
                  HA_Status_t status);
} HA_StepColorReq_t;
```

6.15.2. Functions

6.15.2.1. HA_ColorControlClusterMoveToColorReq()

Prototype:

```
void HA_ColorControlClusterMoveToColorReq(HA_MoveToColorReq_t *req)
```

Description:

Sends Move To Color command to the destination device.

6.15.2.2. HA_ColorControlClusterMoveReq()

Prototype:

```
void HA_ColorControlClusterMoveToColorReq(HA_MoveColorReq_t *req)
```

Description:

Sends Move command to the destination device.

6.15.2.3. HA_ColorControlClusterStepReq()

Prototype:

```
void HA_ColorControlClusterStepReq(HA_StepColorReq_t *req)
```

Description:

Sends Step command to the destination device.

6.16. Pump Configuration and Control Cluster Client

There are no specific commands for this cluster.

6.17. IAS Zone Cluster Client

6.17.1. Data types

6.17.1.1. HA_ZoneEnrollResponseReq_t

Type contains parameters of the Zone Enroll Response command and addressing information.

```
typedef struct _HA_ZoneEnrollResponseReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 8.2.2.3.1.1. */
  HA_ZoneEnrollResponse_t   commandParameters;
  void (*HA_ZoneEnrollResponseConf)(
                  HA_ZoneEnrollResponseReq_t *origReq,
                  HA_Status_t status);
} HA_ZoneEnrollResponseReq_t
```

6.17.1.2. HA_ZoneStatusChangeNotificationInd_t

Type contains parameters of the Zone Status Change Notification command and addressing information.

```
typedef struct _HA_ZoneStatusChangeNotificationInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.7.4.3.1.1. */
  HA_ZoneStatusChangeNotification_t   commandParameters;
} HA_ZoneStatusChangeNotificationInd_t
```

6.17.1.3. HA_ZoneEnrollRequestInd_t

Type contains parameters of the Zone Enroll Request command and addressing information.

```
typedef struct _HA_ZoneEnrollRequestInd_t
{
```

```
    HA_IndicationAddressingInfo_t    addrInfo;
    /** Command parameters as described in [4] section 8.2.2.4.2.1. */
    HA_ZoneEnrollRequest_t   commandParameters;
  } HA_ZoneEnrollRequestInd_t
```

6.17.2. Functions

6.17.2.1. HA_IASZoneClusterZoneEnrollResponseReq()

Prototype:

```
void HA_IASZoneClusterZoneEnrollResponseReq(
                    HA_ZoneEnrollResponseReq_t *req)
```

Description:

Sends Zone Enroll Response command to the destination device.

6.17.2.2. HA_IASZoneClusterZoneStatusChangeNotificationInd()

Prototype:

```
void HA_IASZoneClusterZoneStatusChangeNotificationInd(
                    HA_ZoneStatusChangeNotificationInd_t *ind)
```

Description:

Indicates parameters of received Zone Status Change Notification command. Should be defined on the application level.

6.17.2.3. HA_IASZoneClusterZoneEnrollRequestInd()

Prototype:

```
void HA_IASZoneClusterZoneEnrollRequestInd(
                    HA_ZoneEnrollRequestInd_t *ind)
```

Description:

Indicates parameters of received Zone Enroll Request command. Should be defined on the application level.

6.18. IAS WD Cluster Client

6.18.1. Data types

6.18.1.1. HA_StartWarningReq_t

Type contains parameters of the Start Warning command and addressing information.

```
typedef struct _HA_StartWarningReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.9.4.3.1.1. */
  HA_StartWarning_t   commandParameters;
  void (*HA_StartWarningConf)(
                  HA_StartWarningReq_t *origReq,
                  HA_Status_t status);
} HA_StartWarningReq_t
```

6.18.1.2. HA_SquawkReq_t

Type contains parameters of the Squawk command and addressing information.

```
typedef struct _HA_SquawkReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 8.4.2.3.2.1. */
  HA_Squawk_t   commandParameters;
  void (*HA_SquawkConf)(
                  HA_SquawkReq_t *origReq,
                  HA_Status_t status);
} HA_SquawkReq_t
```

6.18.2. Functions

6.18.2.1. HA_IASWDClusterStartWarningReq()

Prototype:

```
void HA_IASWDClusterStartWarningReq(HA_StartWarningReq_t *req)
```

Description:

Sends Start Warning command to the destination device.

6.18.2.2. HA_IASWDClusterSquawkReq()

Prototype:

```
void HA_IASWDClusterSquawkReq(HA_SquawkReq_t *req)
```

Description:

Sends Squawk command to the destination device.

6.19. IAS ACE Cluster Server

6.19.1. Data types

6.19.1.1. HA_ArmResponseReq_t

Type contains parameters of the Arm Response command and addressing information.

```
typedef struct _HA_ArmResponseReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 8.3.2.5.1.1. */
  HA_ArmResponse_t   commandParameters;
  void (*HA_ArmResponseConf)(
                  HA_ArmResponseReq_t *origReq,
                  HA_Status_t status);
} HA_ArmResponseReq_t
```

6.19.1.2. HA_GetZoneIDMapResponseReq_t

Type contains parameters of the Get Zone ID Map Response command and addressing information.

```
typedef struct _HA_GetZoneIDMapResponseReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 8.3.2.5.2.1. */
  HA_GetZoneIDMapResponse_t   commandParameters;
  void (*HA_GetZoneIDMapResponseConf)(
                  HA_GetZoneIDMapResponseReq_t *origReq,
                  HA_Status_t status);
} HA_GetZoneIDMapResponseReq_t
```

6.19.1.3. HA_GetZoneInformationResponseReq_t

Type contains parameters of the Get Zone Information Response command and addressing information.

```
typedef struct _HA_GetZoneInformationResponseReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 8.3.2.5.3.1. */
```

```
  HA_GetZoneInformationResponse_t   commandParameters;
  void (*HA_GetZoneInformationResponseConf)(
                  HA_GetZoneInformationResponseReq_t *origReq,
                  HA_Status_t status);
} HA_GetZoneInformationResponseReq_t
```

6.19.1.4. HA_ZoneStatusChangedReq_t

Type contains parameters of the Zone Status Changed command and addressing information.

```
typedef struct _HA_ZoneStatusChangedReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.8.4.4.2.1. */
  HA_ZoneStatusChanged_t   commandParameters;
  void (*HA_ZoneStatusChangedConf)(
                  HA_ZoneStatusChangedReq_t *origReq,
                  HA_Status_t status);
} HA_ZoneStatusChangedReq_t
```

6.19.1.5. HA_PanelStatusChangedReq_t

Type contains parameters of the Panel Status Changed command and addressing information.

```
typedef struct _HA_PanelStatusChangedReq_t
{
  HA_RequestAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.8.4.4.3.1. */
  HA_PanelStatusChanged_t   commandParameters;
  void (*HA_PanelStatusChangedConf)(
                  HA_PanelStatusChangedReq_t *origReq,
                  HA_Status_t status);
} HA_PanelStatusChangedReq_t
```

6.19.1.6. HA_ArmInd_t

Type contains parameters of the received Arm command.

```
typedef struct _HA_ArmInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [2] section 10.8.4.3.1.1. */
  HA_Arm_t   commandParameters;
} HA_ArmInd_t;
```

6.19.1.7. HA_BypassInd_t

Type contains parameters of the received Bypass command.

```
typedef struct _HA_BypassInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 8.3.2.4.2.1. */
  HA_Bypass_t   commandParameters;
} HA_BypassInd_t;
```

6.19.1.8. HA_GetZoneInformationInd_t

Type contains parameters of the received Get Zone Information command.

```
typedef struct _HA_GetZoneInformationInd_t
{
  HA_IndicationAddressingInfo_t    addrInfo;
  /** Command parameters as described in [4] section 8.3.2.4.5.1. */
  HA_GetZoneInformation_t   commandParameters;
} HA_GetZoneInformationInd_t;
```

6.19.2. Functions

6.19.2.1. HA_IASACEClusterArmResponseReq()

Prototype:

```
void HA_IASACEClusterArmResponseReq(HA_ArmResponseReq_t *req)
```

Description:

Sends Arm Response command to the destination device.

6.19.2.2. HA_IASACEClusterGetZoneIDMapResponseReq()

Prototype:

```
void HA_IASACEClusterGetZoneIDMapResponseReq(
                    HA_GetZoneIDMapResponseReq_t *req)
```

Description:

Sends Get Zone ID Map Response command to the destination device.

6.19.2.3. HA_IASACEClusterGetZoneInformationResponseReq()

Prototype:

```
void HA_IASACEClusterGetZoneInformationResponseReq(
                    HA_GetZoneInformationResponseReq_t *req)
```

Description:
Sends Get Zone Information Response command to the destination device.

6.19.2.4. HA_IASACEClusterZoneStatusChangedReq()

Prototype:
```
void HA_IASACEClusterZoneStatusChangedReq(
                    HA_ZoneStatusChangedReq_t *req)
```

Description:
Sends Get Zone Information Response command to the destination device.

6.19.2.5. HA_IASACEClusterPanelStatusChangedReq()

Prototype:
```
void HA_IASACEClusterPanelStatusChangedReq(
                    HA_PanelStatusChangedReq_t *req)
```

Description:
Sends Panel Status Changed command to the destination device.

6.19.2.6. HA_IASACEClusterArmInd()

Prototype:
```
void HA_IASACEClusterArmInd(HA_ArmInd_t *ind)
```

Description:
Indicates parameters of received Arm command. Should be defined on the application level.

6.19.2.7. HA_IASACEClusterBypassInd()

Prototype:
```
void HA_IASACEClusterBypassInd(HA_BypassInd_t *ind)
```

Description:
Indicates parameters of received Bypass command. Should be defined on the application level.

6.19.2.8. HA_IASACEClusterEmergencyInd()

Prototype:

```
void HA_IASACEClusterEmergencyInd(
                    HA_IndicationAddressingInfo_t  *  addrInfo)
```

Description:

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

6.19.2.9. HA_IASACEClusterFireInd()

Prototype:

```
void HA_IASACEClusterFireInd(HA_IndicationAddressingInfo_t *addrInfo)
```

Description:

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

6.19.2.10. HA_IASACEClusterPanicInd()

Prototype:

```
void HA_IASACEClusterPanicInd(HA_IndicationAddressingInfo_t *addrInfo)
```

Description:

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

6.19.2.11. HA_IASACEClusterGetZoneIdMapInd()

Prototype:

```
void HA_IASACEClusterGetZoneIdMapInd(
                    HA_IndicationAddressingInfo_t *addrInfo)
```

Description:

Indicates parameters of received Get Zone Id Map command. Should be defined on the application level.

6.19.2.12. HA_IASACEClusterGetZoneInformationInd()

Prototype:

```
void HA_IASACEClusterGetZoneInformationInd(
                    HA_GetZoneInformationInd_t *ind)
```

Description:

Indicates parameters of received Get Zone Information command. Should be defined on the application level.

6.20. Network Services

6.20.1. Data types

6.20.1.1. HA_EnterNetworkConf_t

Type contains parameters of the network the device has entered to.

```
typedef struct _HA_EnterNetworkConf_t
{
    /* Logical channel number of the network. */
    uint8_t     channel;
    /* Short address assigned to the device. */
    uint16_t    shortAddress;
    /* PAN identifier of the entered network. */
    uint16_t    PANId;
    /* Extended PAN identifier of the entered network. */
    uint64_t    extPANId;
    /* Short address of the parent device. */
    uint16_t    parentAddress;
    /* Result status of Enter Network procedure. If contains
       not successful status other fields shall be ignored. */
    HA_Status_t status;
} HA_EnterNetworkConf_t;
```

6.20.1.2. HA_EnterNetworkReq_t

Type describes parameters to be used for entering network.

```
typedef struct _HA_EnterNetworkReq_t
{
    /* Callback function to be called after the Enter Network request
    handling is done. */
    void (*HA_EnterNetworkConf)(
                  HA_EnterNetworkReq_t *origReq,
                  HA_EnterNetworkConf_t *conf);
} HA_EnterNetworkReq_t;
```

6.20.1.3. HA_EZModeNetworkSteeringReq_t

Type contains parameters of EZ-Mode Network Steering request. Confirm callback should be set before request issuing.

```
typedef struct HA_EZModeNetworkSteeringReq_t
{
    HA_EndpointId_t    srcEndpoint;
    void (*HA_EZModeNetworkSteeringConf)(
                    HA_EZModeNetworkSteeringReq_t *origReq,
                    HA_Status_t status);
} HA_EZModeNetworkSteeringReq_t;
```

6.20.1.4. HA_EZModeFindingAndBindingReq_t

Type contains parameters of EZ-Mode Finding and Binding request. Confirm callback should be set before request issuing.

```
typedef struct HA_EZModeFindingAndBindingReq_t
{
    HA_EndpointId_t    srcEndpoint;
    void (*HA_EZModeFindingAndBindingConf)(
                    HA_EZModeFindingAndBindingReq_t,
                    HA_Status_t status);
} HA_EZModeFindingAndBindingReq_t;
```

6.20.1.5. HA_PermitJoiningReq_t

Type describes parameters of Permit Joining request.

```
typedef struct _HA_PermitJoiningReq_t
{
    HA_RequestAddressingInfo_t  addrInfo;
    /** Time span in seconds during which the ZigBee coordinator or
    router will allow associations. The value 0x00 or 0xff indicate
    that permission is, respectively disabled or enabled
    permanently.*/
    uint8_t permitDuration;
    /** If this is set to 0x01 and the remote device is the Trust
    Center, the command affects the Trust Center authentication
    policy as described in [1] section 2.4.3.3.7.2.*/
    uint8_t tcSignificance;
    void (*HA_PermitJoiningConf)(
                    HA_PermitJoiningReq_t *origReq,
                    HA_Status_t status);
} HA_PermitJoiningReq_t;
```

6.20.1.6. HA_ZbProAttributeId_t

Type defines identifiers of the attributes related to the ZigBee PRO network activity.

```
typedef enum _HA_ZbProAttributeId_t
```

```
{
     /* PHY attributes */
     /* From 0x00 to 0x07 – attributes from [1] table 23*/
     /* From 0x08 to 0x39 – reserved for additional attributes, not
described in specification. For example, parameters of the frame
filter. */

     /* MAC attributes */
     /* From 0x40 to 0x55 – attributes from [1] table 86. */
     /* From 0x56 to 0x7F – reserved for additional attributes, not
described in specification. */

     /* NWK attributes */
     /* From 0x80 to 0xAA – attributes from [1] table 3.44. */
     /* From 0xAB to 0xC0 – reserved for additional attributes, not
described in specification. */

     /* APS attributes */
     /* From 0xC1 to 0xCD – attributes from [1] table 2.24. */
     /* From 0xCE to 0xFE – reserved for additional attributes, not
described in specification. */
} HA_ZbProAttributeId_t;
```

6.20.1.7. HA_ZbProAttributeValue_t

Type describes the service structure which is used to convey arbitrary attribute value.

```
typedef union _HA_ZbProAttributeValue_t
{
     bool         boolean;
     uint32_t     integer;
     uint64_t     longInteger;
     struct {
         uint8_t  *ptr;
         uint8_t  length;
     };
} HA_ZbProAttributeValue_t;
```

6.20.1.8. HA_SetReq_t

Type describes parameters of Set request.

```
typedef struct _HA_SetReq_t
{
     HA_ZbProAttributeId_t attrId;
     HA_ZbProAttributeValue_t value;
     void (*HA_SetConf)(HA_SetReq_t *origReq, HA_Status_t status);
} HA_SetReq_t;
```

6.20.1.9. HA_GetReq_t

Type describes parameters of Get request.

```
typedef struct _HA_GetReq_t
{
     HA_ZbProAttributeId_t attrId;
     void (*HA_GetConf)(HA_GetReq_t *origReq, HA_GetConf_t *conf);
} HA_GetReq_t;
```

6.20.1.10. HA_GetConf_t

Type describes parameters of Get confirm.

```
typedef struct _HA_GetConf_t
{
     HA_Status_t status;
     HA_ZbProAttributeValue_t value;
} HA_GetConf_t;
```

6.20.1.11. HA_ResetToFactoryFreshReq_t

Type describes parameters of Reset To Factory Fresh request.

```
typedef struct _HA_ResetToFactoryFreshReq_t
{
     void (*HA_ResetToFactoryFreshConf)(
                    HA_ResetToFactoryFreshReq_t *origReq,
                    HA_Status_t status);
} HA_ResetToFactoryFreshReq_t;
```

6.20.2. Functions

6.20.2.1. HA_EnterNetworkReq()

Prototype:

```
void HA_EnterNetworkReq(HA_EnterNetworkReq_t *req)
```

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 HA_COORDINATOR_DEVICE_TYPE) 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 HA_COORDINATOR_DEVICE_TYPE 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:

```
HA_EnterNetworkReq_t haEnterNetworkReq;
```

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

```
haEnterNetworkReq.HA_EnterNetworkConf = haEnterNetworkConfirmCb;
```

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

```
void haEnterNetworkConfirmCb(
                     HA_EnterNetworkReq_t *origReq,
                     HA_EnterNetworkConf_t *conf)
{
  /* 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:

```
HA_EnterNetworkReq(&haEnterNetworkReq);
```

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.

6.20.2.2. HA_ResetToFactoryFreshReq()

Prototype:

```
void HA_ResetToFactoryFreshReq(HA_ResetToFactoryFreshReq_t *req)
```

Description:

Initiates creation of a new network with specified parameters.

Usage example:

This section explains how the user application can perform typical operations with HA Reset To Factory Fresh request primitive.

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

```
HA_ResetToFactoryFreshReq_t haResetToFactoryFreshReq;
```

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

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

3. The HA_ResetToFactoryFreshReq() can be issued using the callback function defined at the previous step as argument:

```
HA_ResetToFactoryFreshReq(&haResetToFactoryFreshReq);
```

4. When the haResetToFactoryFreshConfirmCb() function is called with the status equal to HA_SUCCESS_STATUS the procedure of reverting to the default settings is completed.

6.20.2.3. HA_EZModeNetworkSteeringReq()

Prototype:

```
void HA_EZModeNetworkSteeringReq(HA_EZModeNetworkSteeringReq_t *req)
```

Description:

Implements EZ-Mode Network Steering procedure as described in [2] paragraph 8.3.4.

Usage example:

This section explains how the user application can perform typical operations with HA EZ-Mode Network Steering request primitive.

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

```
HA_EZModeNetworkSteeringReq_t haEZModeNetworkSteeringReq;
```

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

```
haEZModeNetworkSteeringReq.srcEndpoint = THIS_APPLICATION_ENDPOINT;
haEZModeNetworkSteeringReq.HA_EZModeNetworkSteeringConf =
                                haEZModeNetworkSteeringConfirmCb;
```

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

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

4. After the haEZModeNetworkSteeringConfirmCb() function is called a new HA EZ-Mode Network Steering request can be issued using the same haEZModeNetworkSteeringReq structure.

6.20.2.4. HA_EZModeFindingAndBindingReq()

Prototype:

```
void HA_EZModeFindingAndBindingReq(
                        HA_EZModeFindingAndBindingReq_t *req)
```

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:

```
HA_EZModeFindingAndBindingReq_t haEZModeFindingAndBindingReq;
```

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

```
haEZModeFindingAndBindingReq.srcEndpoint = THIS_APPLICATION_ENDPOINT;
haEZModeFindingAndBindingReq.HA_EZModeFindingAndBindingConf =
                                haEZModeFindingAndBindingConfirmCb;
```

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

```
void haEZModeFindingAndBindingConfirmCb(
                    HA_EZModeFindingAndBindingReq_t *origReq,
```

```
                      HA_Status_t status)
{
  /* Put here a code to analyze the result status */
}
```

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

6.20.2.5. HA_PermitJoiningReq()

Prototype:

```
void HA_PermitJoiningReq(HA_PermitJoiningReq_t *req)
```

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.

6.20.2.6. HA_SetReq()

Prototype:

```
void HA_SetReq(HA_SetReq_t *req)
```

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:

```
HA_SetReq_t haSetReq; /* The set request to be used by the application.
                         */
```

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

```
haSetReq.attrId = MAC_MAX_CSMA_BACKOFFS;
haSetReq.value.integer = USER_SPECIFIED_MAC_MAX_CSMA_BACKOFFS;
haSetReq.HA_SetConf = haSetConfirmCb;
```

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

```
void haSetConfirmCb(
                     HA_SetReq_t *req,
                     HA_Status_t status)
{
  /* Put here a code to analyze the result status. */
}
```

4. 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.

6.20.2.7. HA_GetReq()

Prototype:

```
void HA_GetReq(HA_GetReq_t *req)
```

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:

```
HA_GetReq_t haGetReq = {
    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:

```
void haGetConfirmCb(
                     HA_GetReq_t *req,
                     HA_GetConf_t *conf)
{
  /* 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.

7. API for ZigBee System

7.1. 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.

7.1.1. Data Types

7.1.1.1. BroadBee_HostAccessResult_t

The result of access to and from Host:

```
typedef enum
{
    /* Completed successfully. */
    BROADBEE_NO_ERROR;
    /* Address is out of range. */
    BROADBEE_ADDRESS_OUT_OF_RANGE;
    /* Access has been denied. */
    BROADBEE_ACCESS_DENIED;
    /* Access has taken too long time. */
    BROADBEE_ACCESS_TIME_OUT;
```

```
    /* Data is not available. */
    BROADBEE_DATA_NOT_AVAILABLE;
} BroadBee_HostAccessResult_t;
```

7.1.2. Functions

7.1.2.1. BroadBee_SwDownloadAndStart()

Prototype:

```
BroadBee_HostAccessResult_t BroadBee_SwDownloadAndStart(
                    uint32_t *BroadBeeBinary, uint16_t length);
```

Description:

This function is to download the BroadBee binary file into the internal instruction memory of ZigBee hardware, and start the ZigBee CPU. **This function is not asynchronous, but 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.

7.2. 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.

7.2.1. Data Types

7.2.1.1. MAC_SetConfParams_t

```
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. */
    MAC_PibAttributeIndex_t  attributeIndex;       /*!< The index
within the table of the specified PIB attribute to write. */
} MAC_SetConfParams_t;
```

7.2.1.2. MAC_SetConfCallback_t

```
typedef void (*MAC_SetConfCallback_t)(MAC_SetReqDescr_t *const
reqDescr, MAC_SetConfParams_t *const confParams);
```

7.2.1.3. MAC_SetReqParams_t

```
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. */
    MAC_PibAttributeIndex_t  attributeIndex;       /*!< The index
within the table of the specified PIB attribute to write. */
} MAC_SetReqParams_t;
```

7.2.1.4. MAC_SetReqDescr_t

```
struct _MAC_SetReqDescr_t
{
    /* 32-bit data. */
    MAC_SetConfCallback_t callback;        /*!< Entry point of the
confirmation callback function. */
    MAC_SetReqParams_t     params;          /*!< Request parameters
structured object. */
};
```

7.2.1.5. MAC_GetConfParams_t

```
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. */
```

```
    MAC_PibAttributeIndex_t  attributeIndex;        /*!< The index
within the table or array of the specified PIB attribute to read. */
} MAC_GetConfParams_t;
```

7.2.1.6. MAC_GetConfCallback_t

```
typedef void (*MAC_GetConfCallback_t)(MAC_GetReqDescr_t *const
reqDescr, MAC_GetConfParams_t *const confParams);
```

7.2.1.7. MAC_GetReqParams_t

```
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;
```

7.2.1.8. MAC_GetReqDescr_t

```
typedef struct _MAC_GetReqDescr_t
{
    /* 32-bit data. */
    MAC_GetConfCallback_t callback;        /*!< Entry point of the
confirmation callback function. */
    MAC_GetReqParams_t     params;          /*!< Request parameters
structured object. */
}MAC_GetReqDescr_t;
```

7.2.2. Functions

7.2.2.1. ZBPRO_MAC_SetReq()

Prototype:

```
void ZBPRO_MAC_SetReq(MAC_SetReqDescr_t *const reqDescr);
```

Description:

Set the IEEE address for ZigBee-PRO stack.

Parameters:

reqDescr: Pointer to the request descriptor object

Return value:

None

7.2.2.2. ZBPRO_MAC_GetReq()

Prototype:

```
void ZBPRO_MAC_GetReq(MAC_GetReqDescr_t *const reqDescr);
```

Description:

Get the IEEE address for ZigBee-PRO stack.

Parameters:

reqDescr: Pointer to the request descriptor object

Return value:

None

7.2.2.3. RF4CE_MAC_SetReq()

Prototype:

```
void RF4CE_MAC_SetReq(MAC_SetReqDescr_t *const reqDescr);
```

Description:

Set the IEEE address for RF4CE stack.

Parameters:

reqDescr: Pointer to the request descriptor object

Return value:

None

7.2.2.4. RF4CE_MAC_GetReq()

Prototype:

```
void RF4CE_MAC_GetReq(MAC_GetReqDescr_t *const reqDescr);
```

Description:

Get the IEEE address for RF4CE stack.

Parameters:

reqDescr: Pointer to the request descriptor object

Return value:

None

7.3. 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.

7.3.1. Data Types

7.3.1.1. SYS_DataPointer_t

```
typedef struct _SYS_DataPointer_t
{
    /* Pointer to the linked dynamic or static memory */
    Uint8_t *data;
    /* Current size of the allocated dynamic or static Payload object
       */
    Uint8_t size;
} SYS_DataPointer_t;
```

7.3.1.2. NVM_ReadFileIndParams_t

```
typedef struct _NVM_ReadFileIndParams_t
{
    /* Filename Index */
    Uint8_t id;
    /* 32-bit address offset from the beginning of the file in Host */
    Uint32_t address;
    /* Length to read in byte unit */
    Uint32_t length;
} NVM_ReadFileIndParams_t;
```

7.3.1.3. NVM_ReadFileRespParams_t

```
typedef struct _NVM_ReadFileRespParams_t
{
    /* Result status */
    BroadBee_HostAccessResult_t status;
    /* Requested data */
    SYS_DataPointer_t payload;
} NVM_ReadFileRespParams_t;
```

7.3.1.4. NVM_ReadFileResp_t

```
typedef void (*NVM_ReadFileResp_t)(NVM_ReadFileIndDescr_t *orgInd,
                                   NVM_ReadFileRespParams_t *resp);
```

7.3.1.5. NVM_ReadFileIndDescr_t

```
typedef struct _NVM_ReadFileIndDescr_t
{
    /* Indication parameters. */
    NVM_ReadFileIndParams_t params;
    /* Response callback. */
    NVM_ReadFileResp_t callback;
} NVM_ReadFileIndDescr_t;
```

7.3.1.6. NVM_OpenFileIndParams_t

```
typedef struct _NVM_OpenFileIndParams_t
{
    /* Filename Index */
    Uint8_t id;
} NVM_OpenFileIndParams_t;
```

7.3.1.7. NVM_OpenFileRespParams_t

```
typedef struct _NVM_OpenFileRespParams_t
{
    /* Result status */
    BroadBee_HostAccessResult_t status;
} NVM_OpenFileRespParams_t;
```

7.3.1.8. NVM_OpenFileResp_t

```
typedef void (*NVM_OpenFileResp_t)(NVM_OpenFileIndDescr_t *orgInd,
                                   NVM_OpenFileRespParams_t *resp);
```

7.3.1.9. NVM_OpenFileIndDescr_t

```
typedef struct _NVM_OpenFileIndDescr_t
{
    /* Indication parameters. */
    NVM_OpenFileIndParams_t params;
    /* Response callback. */
    NVM_OpenFileResp_t callback;
} NVM_OpenFileIndDescr_t;
```

If the `FilenameIndex` is not for already existing file, then the open operation should generate a new file with this `FilenameIndex`.

7.3.1.10. NVM_WriteFileToHostIndParams_t

```
typedef struct _NVM_WriteFileToHostIndParams_t
{
    /* Filename Index */
    Uint8_t id;
    /* 32-bit address offset from the beginning of the file in Host */
```

```
    Uint32_t address;
    /* Data to be written to the file */
    SYS_DataPointer_t payload;
} NVM_WriteFileToHostIndParams_t;
```

7.3.1.11. NVM_WriteFileToHostRespParams_t

```
typedef struct _NVM_WriteFileToHostRespParams_t
{
    /* Result status */
    BroadBee_HostAccessResult_t status;
} NVM_WriteFileToHostRespParams_t;
```

7.3.1.12. NVM_WriteFileToHostResp_t

```
typedef void
(*NVM_WriteFileToHostResp_t)(NVM_WriteFileToHostIndDescr_t *orgInd,
                             NVM_WriteFileToHostRespParams_t *resp);
```

7.3.1.13. NVM_WriteFileToHostIndDescr_t

```
typedef struct _NVM_WriteFileToHostIndDescr_t
{
    /* Indication parameters. */
    NVM_WriteFileToHostIndParams_t params;
    /* Response callback. */
    NVM_WriteFileToHostResp_t callback;
} NVM_WriteFileToHostIndDescr_t;
```

If the `FilenameIndex` is not for already existing file, then the write operation should generate a new file with this `FilenameIndex`.

7.3.1.14. NVM_CloseFileIndParams_t

```
typedef struct _NVM_CloseFileIndParams_t
{
    /* Filename Index */
    Uint8_t id;
} NVM_CloseFileIndParams_t;
```

7.3.1.15. NVM_CloseFileRespParams_t

```
typedef struct _NVM_CloseFileRespParams_t
{
    /* Result status */
    BroadBee_HostAccessResult_t status;
} NVM_CloseFileRespParams_t;
```

7.3.1.16. NVM_CloseFileResp_t

```
typedef void (*NVM_CloseFileResp_t)(NVM_CloseFileIndDescr_t *orgInd,
```

```
                                        NVM_CloseFileRespParams_t *resp);
```

7.3.1.17. NVM_CloseFileIndDescr_t

```
typedef struct _NVM_CloseFileIndDescr_t
{
    /* Indication parameters. */
    NVM_CloseFileIndParams_t params;
    /* Response callback. */
    NVM_CloseFileResp_t callback;
} NVM_CloseFileIndDescr_t;
```

7.3.2. 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.

7.3.2.1. NVM_ReadFileInd()

Prototype:

```
void NVM_ReadFileInd(NVM_ReadFileIndDescr_t *indDescr);
```

Description:

This function is to read data from a file in Host Non-volatile memory.

Parameters:

indDescr: pointer to the indication parameters

Return value:

None.

7.3.2.2. NVM_OpenFileInd()

Prototype:

```
void NVM_OpenFileInd(NVM_OpenFileIndDescr_t *indDescr);
```

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.

Parameters:
indDescr: pointer to the indication parameters

Return value:
None.

7.3.2.3. NVM_WriteFileInd()

Prototype:

```
void NVM_WriteFileInd(NVM_WriteFileIndDescr_t *indDescr);
```

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.

Parameters:
indDescr: pointer to the parameters to write data into a file in Host

Return value:
None.

7.3.2.4. NVM_CloseFileInd()

Prototype:

```
void NVM_CloseFileInd(NVM_CloseFileIndDescr_t *indDescr);
```

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.

Parameters:
indDescr: pointer to the indication parameters

Return value:
None.

7.4. 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.

7.5. 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.

7.6. 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.

7.6.1. Data Types

7.6.1.1. RF4CE_WAKE_UP_ACTION_CODE_FILTER_LENGTH

```
#define RF4CE_WAKE_UP_ACTION_CODE_FILTER_LENGTH  (256/8)
```

This specifies the length of “wakeUpActionCodeFilter”in the byte unit

7.6.1.2. RF4CE_SetWakeUpActionCodeReqParams_t

```
typedef struct _RF4CE_SetWakeUpActionCodeReqParams_t
{
    uint8_t
wakeUpActionCodeFilter[RF4CE_WAKE_UP_ACTION_CODE_FILTER_LENGTH];
} RF4CE_SetWakeUpActionCodeReqParams_t;
```

7.6.1.3. RF4CE_SetWakeUpActionCodeConfParams_t

```
typedef struct _RF4CE_SetWakeUpActionCodeConfParams_t
{
    uint8_t
wakeUpActionCodeFilter[RF4CE_WAKE_UP_ACTION_CODE_FILTER_LENGTH];
   uint8_t status;  //0 if success, otherwise fail.
} RF4CE_SetWakeUpActionCodeConfParams_t;
```

7.6.1.4. RF4CE_ SetWakeUpActionCodeReqDescr_t

```
typedef struct _RF4CE_SetWakeUpActionCodeReqDescr_t
{
    RF4CE_SetWakeUpActionCodeReqParams_t params;
```

```
    void (*callback)(RF4CE_SetWakeUpActionCodeReqDescr_t *,
RF4CE_SetWakeUpActionCodeConfParams_t *);
} RF4CE_SetWakeUpActionCodeReqDescr_t;
```

7.6.1.5. RF4CE_GetWakeUpActionCodeReqParams_t

```
typedef struct _RF4CE_GetWakeUpActionCodeReqDescr_t
{
   void (*callback)(RF4CE_GetWakeUpActionCodeReqDescr_t *,
RF4CE_GetWakeUpActionCodeConfParams_t *);
} RF4CE_GetWakeUpActionCodeReqDescr_t;
```

7.6.2. Functions

7.6.2.1. RF4CE_SetWakeUpActionCodeReq()

Prototype:

```
void RF4CE_SetWakeUpActionCodeReq(
                RF4CE_SetWakeUpActionCodeReqDescr_t *req);
```

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.

Return value:

None. Actual return of the “wakeUpActionCodeFilter” value and the status will be returned through a callback function.

7.6.2.2. RF4CE_GetWakeUpActionCodeReq()

Prototype:

```
void RF4CE_GetWakeUpActionCodeReq(RF4CE_GetWakeUpActionCodeReqDescr_t
*req);
```

Description:

This function is to read the “wakeUpActionCodeFilter” from the BroadBee stack.

Return value:

None. Actual return of the “wakeUpActionCodeFilter” value and the status will be returned through a callback function.

8. 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.

8.1. RF4CE Remote Control Profile

8.1.1. 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.

8.1.2. 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.

8.1.3. 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.

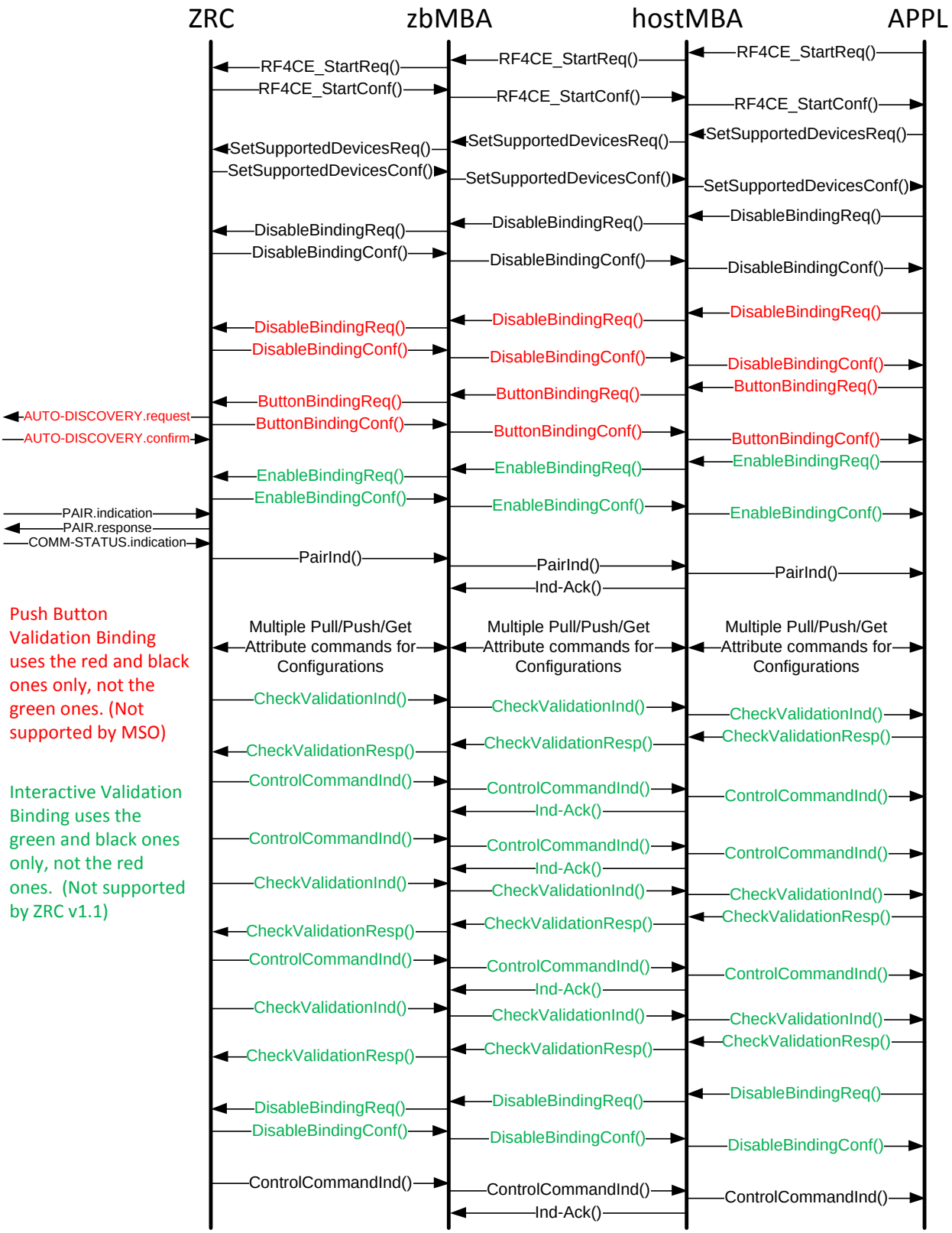

Figure 18: Binding Flow on Host Side

8.2. ZigBee-PRO Home Automation Profile

8.2.1. Form Network

To form a ZigBee-PRO Home Automation network, application software needs to call the function, **HA_EnterNetworkReq()**. 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.

8.2.2. Permit Joining

The newly formed Home Automation network should permit joining process so that the other end devices can join this network. To do this, we need to call the function, **HA_PermitJoiningReq()**. This function is also used to prohibit some devices from joining this network.

8.2.3. 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, **HA_RegisterEndpointReq()** 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 **HA_EZModeFindingAndBindingReq()** 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.

8.2.4. 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 **HA_OnOffClusterOnReq()**, **HA_OnOffClusterOffReq()**, or **HA_OnOffClusterToggleReq()** function to turn on, turn off, or toggle the target home automation device.