Application note

ZSDK API implementation for Zigbee® on STM32WB Series

Introduction

This application note describes the implementation of the ZSDK (Zigbee® software design kit) API (application programming interface) for Zigbee® on the STM32WB Series.

It describes all APIs exposed on Cortex® M4 side used to control the stack. In particular, it covers APIs used to access to the following layers: BDB (base device behavior), APS (applications services), NWK (network layer services) and ZDO (Zigbee® device object).

This document also includes the ZCL (Zigbee® cluster library) generic functions used to control the clusters.

1 General information

This document applies to STM32WB Series Arm®-based devices.

arm

Note: Arm is a registered trademark of Arm Limited (or its subsidiaries) in the US and/or elsewhere.

The following table defines the acronyms needed for a better understanding of this document.

Table 1. List of acronyms

Acronym	Description
API	Application programming interface
APS	Applications services
BDB	Base device behavior
NWK	Network layer services
TC	Trust center
TCSO	Trust center swap out
ZCL	Zigbee cluster library
ZDO	Zigbee device object
ZSDK	Zigbee software design kit

2 Reference documents

Table 2. **Reference documents**

Reference number	Document title	Document number
[R1]	Zigbee specification revision 22	5-3474-22
[R2]	Base device behavior specification version 1.0	13-0402-13
[R3]	Zigbee cluster library specification revision 6	07-5123-06

3 Preliminaries

3.1 Zigbee stack handle

The stack is initialized by calling the function `ZbInit()`, which returns a stack instance handle of type `ZigBeeT *`. This is an anonymous pointer meaning to the application that its contents are private to the stack. There is nothing that the application can or must do to this pointer, other than provides it back to the stack when calling virtually all the stack API functions. For this reason, the stack API functions look like:

```
ZbFunc(struct ZigBeeT *zb, ...
```

3.2 Conventions

The stack makes a frequent use of fixed minimum size datatypes such as uint8_t, uint64_t. These are defined in and the applications have to include the following header file:

```
#include <stdint.h>
```

The use of `bool` is less common for logical types. This is defined in:

```
#include <stdbool.h>
```

3.3 Status codes

Many functions in the Zigbee stack return a status code as an enumeration. Below are the enumerations for the standard Zigbee status codes along with the Exegin add-ons.

3.3.1 General status codes

Table 3. General status codes

Status value	Status code
0x00	ZB_STATUS_SUCCESS
0x70	ZB_STATUS_ALLOC_FAIL
0x71	ZB_STATUS_TIMEOUT

3.3.2 ZDP status codes

Table 4. ZDP status codes

Status value	Status code
0x00	ZB_ZDP_STATUS_SUCCESS
0x80	ZB_ZDP_STATUS_INV_REQTYPE
0x81	ZB_ZDP_STATUS_DEVNOTFOUND
0x82	ZB_ZDP_STATUS_INVALID_EP
0x83	ZB_ZDP_STATUS_NOT_ACTIVE
0x84	ZB_ZDP_STATUS_NOT_SUPPORTED
0x85	ZB_ZDP_STATUS_TIMEOUT
0x86	ZB_ZDP_STATUS_NO_MATCH
0x88	ZB_ZDP_STATUS_NO_ENTRY
0x89	ZB_ZDP_STATUS_NO_DESCRIPTOR

Status value	Status code
0x8a	ZB_ZDP_STATUS_INSUFFICIENT_SPACE
0x8b	ZB_ZDP_STATUS_NOT_PERMITTED
0x8c	ZB_ZDP_STATUS_TABLE_FULL
0x8d	ZB_ZDP_STATUS_NOT_AUTHORIZED
0x8e	ZB_ZDP_STATUS_DEVICE_BINDING_TABLE_FULL
0x8f	ZB_ZDP_STATUS_INVALID_INDEX

3.3.3 APS status codes

Table 5. APS status codes

Status value	Status code
0x00	ZB_APS_STATUS_SUCCESS
0xa0	ZB_APS_STATUS_ASDU_TOO_LONG
0xa1	ZB_APS_STATUS_DEFRAG_DEFERRED
0xa2	ZB_APS_STATUS_DEFRAG_UNSUPPORTED
0xa3	ZB_APS_STATUS_ILLEGAL_REQUEST
0xa4	ZB_APS_STATUS_INVALID_BINDING
0xa5	ZB_APS_STATUS_INVALID_GROUP
0xa6	ZB_APS_STATUS_INVALID_PARAMETER
0xa7	ZB_APS_STATUS_NO_ACK
0xa8	ZB_APS_STATUS_NO_BOUND_DEVICE
0xa9	ZB_APS_STATUS_NO_SHORT_ADDRESS
0xaa	ZB_APS_STATUS_NOT_SUPPORTED
0xab	ZB_APS_STATUS_SECURED_LINK_KEY
0xac	ZB_APS_STATUS_SECURED_NWK_KEY
0xad	ZB_APS_STATUS_SECURITY_FAIL
0xae	ZB_APS_STATUS_TABLE_FULL
0xaf	ZB_APS_STATUS_UNSECURED
0xb0	ZB_APS_STATUS_UNSUPPORTED_ATTRIBUTE
0xbd	ZB_APS_STATUS_INVALID_INDEX

3.3.4 NWK status codes

Table 6. NWK status codes

Status value	Status code
0x00	ZB_NWK_STATUS_SUCCESS
0xc1	ZB_NWK_STATUS_INVALID_PARAMETER
0xc2	ZB_NWK_STATUS_INVALID_REQUEST
0xc3	ZB_NWK_STATUS_NOT_PERMITTED
0xc4	ZB_NWK_STATUS_STARTUP_FAILURE
0xc5	ZB_NWK_STATUS_ALREADY_PRESENT

Status value	Status code
0xc6	ZB_NWK_STATUS_SYNC_FAILURE
0xc7	ZB_NWK_STATUS_TABLE_FULL
0xc8	ZB_NWK_STATUS_UNKNOWN_DEVICE
0xc9	ZB_NWK_STATUS_UNSUPPORTED_ATTRIBUTE
0xca	ZB_NWK_STATUS_NO_NETWORKS
0xcb	ZB_NWK_STATUS_LEAVE_UNCONFIRMED
0xcc	ZB_NWK_STATUS_MAX_FRM_CNTR
0xcd	ZB_NWK_STATUS_NO_KEY
0xce	ZB_NWK_STATUS_BAD_CCM_OUTPUT
0xcf	ZB_NWK_STATUS_NO_ROUTING_CAPACITY
0xd0	ZB_NWK_STATUS_ROUTE_DISCOVERY_FAILED
0xd1	ZB_NWK_STATUS_ROUTE_ERROR
0xd2	ZB_NWK_STATUS_BT_TABLE_FULL
0xd3	ZB_NWK_STATUS_FRAME_NOT_BUFFERED
0xd4	ZB_NWK_STATUS_INVALID_INDEX
0xd5	ZB_NWK_STATUS_INTERNAL_ERR

3.3.5 WPAN status codes

Table 7. WPAN status codes

Status value	Status code
0x00	ZB_WPAN_STATUS_SUCCESS
0xdb	ZB_WPAN_STATUS_COUNTER_ERROR
0xdc	ZB_WPAN_STATUS_IMPROPER_KEY_TYPE
0xdd	ZB_WPAN_STATUS_IMPROPER_SECURITY_LEVEL
0xde	ZB_WPAN_STATUS_UNSUPPORTED_LEGACY
0xdf	ZB_WPAN_STATUS_UNSUPPORTED_SECURITY
0xe0	ZB_WPAN_STATUS_BEACON_LOSS
0xe1	ZB_WPAN_STATUS_CHANNEL_ACCESS_FAILURE
0xe2	ZB_WPAN_STATUS_DENIED
0xe3	ZB_WPAN_STATUS_DISABLE_TRX_FAILURE
0xe4	ZB_WPAN_STATUS_SECURITY_ERROR
0xe5	ZB_WPAN_STATUS_FRAME_TOO_LONG
0xe6	ZB_WPAN_STATUS_INVALID_GTS
0xe7	ZB_WPAN_STATUS_INVALID_HANDLE
0xe8	ZB_WPAN_STATUS_INVALID_PARAMETER
0xe9	ZB_WPAN_STATUS_NO_ACK
0xea	ZB_WPAN_STATUS_NO_BEACON
0xeb	ZB_WPAN_STATUS_NO_DATA
0xec	ZB_WPAN_STATUS_NO_SHORT_ADDRESS
0xed	ZB_WPAN_STATUS_OUT_OF_CAP

Status value	Status code
0xee	ZB_WPAN_STATUS_PAN_ID_CONFLICT
0xef	ZB_WPAN_STATUS_REALIGNMENT
0xf0	ZB_WPAN_STATUS_TRANSACTION_EXPIRED
0xf1	ZB_WPAN_STATUS_TRANSACTION_OVERFLOW
0xf2	ZB_WPAN_STATUS_TX_ACTIVE
0xf3	ZB_WPAN_STATUS_UNAVAILABLE_KEY
0xf4	ZB_WPAN_STATUS_UNSUPPORTED_ATTRIBUTE
0xf5	ZB_WPAN_STATUS_INVALID_ADDRESS
0xf6	ZB_WPAN_STATUS_ON_TIME_TOO_LONG
0xf7	ZB_WPAN_STATUS_PAST_TIME
0xf8	ZB_WPAN_STATUS_TRACKING_OFF
0xf9	ZB_WPAN_STATUS_INVALID_INDEX
0xfa	ZB_WPAN_STATUS_LIMIT_REACHED
0xfb	ZB_WPAN_STATUS_READ_ONLY
0xfc	ZB_WPAN_STATUS_SCAN_IN_PROGRESS
0xfd	ZB_WPAN_STATUS_SUPERFRAME_OVERLAP
0xfe	ZB_WPAN_STATUS_DRIVER_ERROR
0xff	ZB_WPAN_STATUS_DEVICE_ERROR

3.3.6 ZCL status codes

Table 8. ZCL status codes

Status value	Status code
0z00	ZCL_STATUS_SUCCESS
0x01	ZCL_STATUS_FAILURE
0x70	ZCL_STATUS_ALLOC_FAIL
0x7e	ZCL_STATUS_NOT_AUTHORIZED
0x80	ZCL_STATUS_MALFORMED_COMMAND
0x81	ZCL_STATUS_UNSUPP_CLUSTER_COMMAND
0x82	ZCL_STATUS_UNSUPP_GENERAL_COMMAND
0x83	ZCL_STATUS_UNSUPP_MFR_CLUSTER_COMMAND
0x84	ZCL_STATUS_UNSUPP_MFR_GENERAL_COMMAND
0x85	ZCL_STATUS_INVALID_FIELD
0x86	ZCL_STATUS_UNSUPP_ATTRIBUTE
0x87	ZCL_STATUS_INVALID_VALUE
0x88	ZCL_STATUS_READ_ONLY
0x89	ZCL_STATUS_INSUFFICIENT_SPACE
0x8a	ZCL_STATUS_DUPLICATE_EXISTS
0x8b	ZCL_STATUS_NOT_FOUND
0x8c	ZCL_STATUS_UNREPORTABLE_ATTRIBUTE
0x8d	ZCL_STATUS_INVALID_DATA_TYPE

Status value	Status code
0x8e	ZCL_STATUS_INVALID_SELECTOR
0x8f	ZCL_STATUS_WRITE_ONLY
0x90	ZCL_STATUS_INCONSISTENT_STARTUP_STATE
0x91	ZCL_STATUS_DEFINED_OUT_OF_BAND
0x92	ZCL_STATUS_INCONSISTENT
0x93	ZCL_STATUS_ACTION_DENIED
0x94	ZCL_STATUS_TIMEOUT
0x95	ZCL_STATUS_ABORT
0x96	ZCL_STATUS_INVALID_IMAGE
0x97	ZCL_STATUS_WAIT_FOR_DATA
0x98	ZCL_STATUS_NO_IMAGE_AVAILABLE
0x99	ZCL_STATUS_REQUIRE_MORE_IMAGE
0x9A	ZCL_STATUS_NOTIFICATION_PENDING
0xc0	ZCL_STATUS_HARDWARE_FAILURE
0xc1	ZCL_STATUS_SOFTWARE_FAILURE
0xc2	ZCL_STATUS_CALIBRATION_ERROR
0xc3	ZCL_STATUS_UNSUPP_CLUSTER
0xc4	ZCL_STATUS_LIMIT_REACHED

3.3.7 Trust center swap out status codes

Table 9. Trust center swap out status codes

Status value	Status code
0x00	ZB_TCSO_STATUS_SUCCESS
0x01	ZB_TCSO_STATUS_DISCOVERY_UNDERWAY
0x02	ZB_TCSO_STATUS_REJOIN_PREV
0x03	ZB_TCSO_STATUS_NOT_FOUND
0x04	ZB_TCSO_STATUS_FATAL

3.3.8 Key establishment status codes

Table 10. Key establishment status codes

Status value	Status code
0x00	ZCL_KEY_STATUS_SUCCESS
0x01	ZCL_KEY_STATUS_UNKNOWN_ISSUER
0x02	ZCL_KEY_STATUS_BAD_KEY_CONFIRM
0x03	ZCL_KEY_STATUS_BAD_MESSAGE
0x04	ZCL_KEY_STATUS_NO_RESOURCES
0x05	ZCL_KEY_STATUS_UNSUPPORTED_SUITE
0x06	ZCL_KEY_STATUS_INVALID_CERTIFICATE
0x07	ZCL_KEY_STATUS_TIMEOUT

Status value	Status code
0x08	ZCL_KEY_STATUS_MATCH_DESC_FAILED

3.4 Application callbacks

Invoking the stack API functions often results in immediate effects, and sometimes produces results the application wants to know about in the future. In these cases, the stack API allows the application to provide a callback function. Some time after the API function has returned, when the requested action is completed (or in some cases a need to notify the application), the application's callback function is invoked.

When your application callback function is invoked, the callback function often requires contextual information known only to the application and not the stack. So that the application callback has the context information needed, the stack API functions take both a callback function and anonymous arg pointer parameter (known to the application, anonymous to the stack). When the stack invokes the callback function it provides the callback with that arg parameter. For example:

```
struct app_data app;
ZbFunc(zb, app_callback, &app);
```

Where `app_callback` is the application providing callback, which must match the function prototype defined by the stack. The application can provide any pointer as the argument (even NULL, if the callback accepts NULL), in this case it provides a pointer to struct of type app_data, of which the stack has no knowledge.

The application defined callback then looks something like `static void`.

The function prototype defined by the stack for most callback functions begins with the stack pointer.

Note that the arg parameter is type `void *(anonymous)` and the application must cast it back to its known type.

It is critical that the application carefully manages the type of the callback arg. The type provided to the stack function must be of the same type used by the callback function. C does not provide the type checking for these anonymous pointers, it is the responsibility of the application to ensure that they are the same type. Using different types result in an unexpected behavior, including the application crashing in the callback.

3.5 Application data structure paradigm

Most applications use some variant of the following technique. The stack does not mandate the following approach, the user is free to use another approach in his application, it is only a one possible approach.

Because of the callback paradigm, with callbacks passed the anonymous `void *` arg parameter, it is convenient to define a common application data structure. The content of this, is entirely up to the application but the stack handle and IEEE address are used virtually in all applications using this technique:

```
struct app_data {
struct ZigBeeT *zb;
uint64_t ieee_addr;
...
};
```

This structure is then used throughout the application, in the main thread of execution and this is the structure used for all callback functions. Having the stack handle `ZigBeeT*` in this structure is convenient, allowing any callback to invoke a stack function. The `ieee_addr` is another common element that is convenient to have accessible.

The reminder of this structure depends on the application. It typically contains many of the main components of the application. Care should be taken to only include elements that need to be accessed from multiple places (in the main thread and callbacks) so that it does not become unnecessarily large and unwieldy. The content is a mix of application specific components and shared stack components such as the ZCL cluster handles.

It may be tempting to segment the application structure for different components of the application, however the experience has shown that it is preferable to base all shared application data of one instance, otherwise there is confusion over which information is available in which context.

The information in this section is only a guideline based on experience, any specific application is free to choose a different approach.

3.6 Timers

3.6.1 ZbTimer

The timers are fundamental to the operation of the stack. The stack uses many different timers internally and the timer operation is evident by side effects such as timeouts. It may also be useful for the application to make use of a timer, so some of the APIs are exposed.

A timer is created by calling `ZbTimerAlloc()`, where a callback function is provided (`app_timer_callback`). This callback is invoked when the timer expires.

```
struct ZbTimerT *timer;
timer = ZbTimerAlloc(zb, app_timer_callback, NULL);
```

However, the timer is not active until it is reseted. The following example resets and starts the timer with a delay of 1000 ms or 1 sec.

```
ZbTimerReset(timer, 1000);
```

The application callback function:

```
static void
app_timer callback(struct ZigBeeT *zb, void *arg)
{
...
}
```

To stop a timer before it expires:

```
ZbTimerStop(timer);
```

To get the time remaining in milliseconds before the timer expires:

```
ZbTimerRemaining(timer); /* Returns UINT_MAX if the timer is not running. */
```

To free a timer:

```
ZbTimerFree(timer);
```

3.6.2 ZbTimerChangeCallback

Changes the given timer's callback function, without having to free and allocate a new timer.

Prototype

```
void ZbTimerChangeCallback(struct ZbTimerT *timer, void (*callback)(struct ZigBeeT *zb, void
*cb_arg), void *arg);
```

Parameters

- timer – Zigbee timer structure
- zb – Zigbee stack structure
- arg – callback argument

3.6.3 ZbTimerRunning

Returns true if the given timer is active.

Prototype

```
bool ZbTimerRunning(struct ZbTimerT *timer);
```

Parameters

- timer – Zigbee timer structure

Returns true on success, false otherwise.

3.7 Uptime

3.7.1 ZbUptime

Retrieves the current uptime in milliseconds. This is typically only used internally by the stack for the timers.

Prototype

```
ZbUptimeT ZbUptime(void);
```

Returns the current uptime in milliseconds.

3.7.2 ZbTimeoutRemaining

Used in conjunction with `ZbUptime`. This function returns the time difference between `expire_time` and now, or 0 if the time has expired (now >= expire_time). This function handles the case where the time value has rolled over starting back from 0.

Prototype

```
unsigned int ZbTimeoutRemaining(ZbUptimeT now, ZbUptimeT expire_time);
```

Parameters

- now – current time
- expire_time – time of timeout expiry

Returns 0 if now >= expire_time, the difference in milliseconds between now and timeout if now < expire_time.

3.8 Stack threads of execution

The stack is designed to be used without change, in a full pre-emptive OS such as Linux®, an RTOS such as FreeRTOS™, or non-OS bare-metal (forever loop while(true)). This is achieved by turning control of the thread of execution over to the application. The stack requires the specific API functions to be invoked in a timely fashion. In a pre-emptive OS typically a stack thread, separate from the application thread, is used. In bare-metal applications the same API functions are used from the application main loop, sequentially before or after the application steps. Care should be taken in the main loop that the application does not take too long or block, so that the stack can meet the timing requirements mandated by [R1].

When the application invokes the stack API, the stack processes anything that can be handled immediately. In many cases, often when a specific timing is required, the stack must defer the processing for some later time. In order to process these deferred operations the application must call `ZbTimerWork()` on a regular basis. This function evaluates the internal queue of pending operations and executes all operations whose time has come. Because the stack and MAC operate independently, the MAC provides a similar API which must also be called on a regular basis: `WpanMacWork()`.

3.8.1 Bare-metal

A bare-metal application main loop looks something like this:

```
while (true){
ZbTimerWork();
WpanMacWork();
application(&app_data);
}
```

3.8.2 OS threading

In an OS or RTOS, the application may run in a separate thread from the stack kicking thread. The stack thread looks similar to the bare-metal example; however, a tight spin-loop is not ideal in a multi-threaded environment. For this reason the stack and MAC provide `ZbCheckTime()` and `WpanMacCheck()`. These functions return the number of milliseconds until the next operation in the stack and MAC respectively. Instead of spinning, the stack thread can then sleep for the minimum value returned by these two APIs, allowing other threads (application) to run without wasting time of unnecessarily and repeatedly checkings for pending events.

Sleepy applications, the ones where the MCU halts and goes into a low power state, can use this same technique to determine how long the MCU can be in the low power halted state before it needs to wake and allow the stack and MAC to run.

3.8.2.1 *ZbWakeupCallbackConfig*

While the stack thread is waiting, an event may occur – for example a received packet -- that requires the attention of the stack to process. This function configures a callback for the stack to wake up and run the stack thread. For example, the stack thread may be waiting on a semaphore with a timeout. The callback simply posts to the semaphore for the stack thread to run immediately.

Prototype

```
void ZbWakeupCallbackConfig(struct ZigBeeT *zb, void (*wakeup_cb)(void));
```

Parameters

- zb – Zigbee stack structure
- wakeup_cb – wakeup callback function

3.9 Packet and message filters

For the most part, the message filters are used within the stack and the typical applications do not need to make use of message filters. However, the message filters are so central to stack operation that they afford a mechanism for applications to extend the stack functionality.

As the messages traverse the stack, through the MAC, NWK, APS, ZDO or ZCL layers, filters are used to direct messages to the appropriate destination. A filter is defined with certain criteria. When a message is processed, the filters are checked and matching messages directed to the requesting filter.

The following example shows how the application can register to receive network update device indications.

```
/* callback intialization */
{
struct ZigBeeT *zb;
struct ZbMsgFilterT * filter;
filter = ZbMsgFilterRegister(zb,
ZB_MSG_FILTER_STATUS_IND | ZB_MSG_FILTER_UPDATE_DEVICE_IND,
ZB_MSG_DEFAULT_PRIO, app_msg_filter_callback, &app);
}
/* callback function */
static int
app_msg_filter_callback(struct ZigBeeT *zb, uint32_t id, void *msg, void *arg)
{
struct app_data *app = arg;
if (id == ZB_MSG_FILTER_STATUS_IND) {
ZbNlmeNetworkStatusIndT *ind = msg;
return ZB_MSG_CONTINUE;
}
}
```

The application defines a callback `app_msg_filter_callback` and registers it by calling `ZbMsgFilterRegister()`. Once registered the callback is invoked for every message that matches the filter.

Note that the callback has a return value. In this case the callback is ZB_MSG_CONTINUE, which allows further filters to receive this message. If instead the callback returns ZB_MSG_CONTINUE then the stack discards the message.

Without very careful consideration of the ramifications, the application must never discard a message as it could alter stack behavior.

There are a lot of additional possible functionalities in message filters, however the above example demonstrates the functionality needed in those occasional cases where the application needs to use the filter functionality.

3.10 ZbMsgFilterRemove

Removes the given message filter from the stack.

```
void ZbMsgFilterRemove(struct ZigBeeT *zb, struct ZbMsgFilterT *filter)
```

Parameters

- zb – Zigbee stack structure

filter – pointer to filter struct returned by ZbMsgFilterRegister.

4 Stack startup

4.1 Initialization

4.1.1 MAC initialization

The stack can operate on many different real and virtual MAC interfaces. Additionally, physical devices can be connected by SPI, UART, or USB interfaces. The MAC and stack can be co-resident or operate on a separate MCU. On a separate MCU, the MCP (MAC control protocol) or a proprietary protocol may be used. There are many variations between devices, their options, and how they are configured and the specifics of the API depend on the device. However, at the end a WpanPublicT *device is created and passed to the stack, which has no knowledge of the device used. The following examples give an outline of how are used very different interfaces (SPI, USB, Linux named-pipes) on very different platforms (from bare-metal to Linux).

4.1.1.1 Example: bare metal device initialization

In this example of a bare-metal implementation, the device accesses the radio through the use of memory-mapped registers. On power-up the radio hardware must be initialized. In this case the PHY is local, so a hardware specific PHY device is created and a MAC instance created that uses this PHY interface. The main steps in this process are:

```
#include "phy.h"
#include "wpan_mac_init.h"
WpanPublicT *device;
struct phyif phy;
uint64_t ext_addr;
/* initialize radio */
radio_init(&phy);
ext_addr = radio_get_ext_addr(); /* obtain from device */
/* optional callbacks not used for simplicity */
device = WpanMacInit(&phy, ext_addr, NULL, NULL);
```

4.1.2 ZbInit

Initializes the Zigbee Layers and returns the stack handle used on all subsequent stack calls.

Prototype

```
struct ZigBeeT * ZbInit(uint64_t extAddr, ZbInitTblSizesT *tblSizes,
        ZbInitSetLoggingT *setLogging)
```

Parameters

- extAddr - extended IEEE address (EUI64)
- tblSizes - table sizes (optional, may be NULL)
  - nwkNeighborTblSz - NWK neighbor table size (default = 64)
  - nwkRouteTblSz - NWK routing table size (default = 32)
  - nwkAddrMapTblSz - NWK address map table (default = 32)
  - nwkBttSz - NWK broadcast transaction table (default = 32)
  - nwkRReqSz - NWK route request table (default = 16)
  - apsPeerLinkKeyTblSz - APS link key table (default = 32)
- setLogging - logging info struct pointer (optional, may be NULL). Refer to the ZbSetLogging section below for more detailed info.

4.1.3 ZbMemConfig

For platforms that do not support malloc, this function configures the memory to be used by the stack. This function is called before calling ZbInit.

Prototype

```
void ZbMemConfig(void *mallocPtr, unsigned int mallocSz, void *heapPtr,
```

unsigned int heapSz);

Parameters

- mallocPtr – memory for initializing the stack instance. The size depends on the table sizes required for the stack (ZbInit tblSizes) and the type of device. A typical Zigbee router requires approximately 30 Kbytes. If not enough memory is provided, the call to ZbInit fails.
- mallocSz – size of memory pointed to by mallocPtr
- heapPtr – memory for the stack to use for temporary buffers (for exemple packets, timers)
- heapSz – size of memory pointed to by heapPtr

4.1.4 ZbSetLogging

Sets the logging to use by the Zigbee stack, and the function that implements logging.

It is not thread-safe to call this function while the stack is running.

Prototype

```
void ZbSetLogging(struct ZigBeeT *zb, uint32_t mask,
       void (*func)(struct ZigBeeT *zb, uint32_t mask, const char *hdr,
       const char *fmt, va_list argptr));
```

Parameters

- zb - zigbee stack instance
- mask - log mask bits. There are some predefined masks:
  - ZB_LOG_MASK_LEVEL_0 = fatal errors
  - ZB_LOG_MASK_LEVEL_1 = level_0 + run-time errors
  - ZB_LOG_MASK_LEVEL_2 = level_1 + run-time warnings
  - ZB_LOG_MASK_LEVEL_3 = level_2 + general info
  - ZB_LOG_MASK_LEVEL_4 = level_3 + ZCL cluster debug
  - ZB_LOG_MASK_LEVEL_5 = level_4 + general debug
  - ZB_LOG_MASK_LEVEL_6 = all
- func - function callback. Arguments:
  - zb - stack pointer
  - mask - log mask that generated this message (e.g. ZB_LOG_MASK_INFO)
  - hdr - header string (typically function name that generated debug message)
  - fmt - format string
  - argptr - va_list arguments (stdarg.h)

4.2 Network management

4.2.1 ZbStartup

Executes the startup procedure as described in section 2.5.5.5.6.2 of [R1].

The startup code also handles callbacks from the stack to maintain the network and security processes. For example, these include the handling of APSME-UPDATE-DEVICE.indication messages when acting as a coordinator trust center, to handle the authentication procedure described in section 4.6.3.2 of [R1].

The message callback handlers can be overridden by the application by creating message filters using ZbMsgFilterRegister. This function and the entire startup code can be bypassed by the application, and the network layer APIs used directly, if so desired.

If security is enabled (nwkSecurityLevel != 0), then this function also overloads the APSME callback functions and performs the authentication procedure described in section 4.6.3.2 of [R1].

This is purely a helper function, and if finer-grained control of startup and authentication is necessary then the application can perform these tasks manually.

Prototype

```
enum ZbStatusCodeT ZbStartup(struct ZigBeeT *zb, struct ZbStartupT *configPtr, void
(*callback)(enum ZbStatusCodeT status, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- ConfigPtr – startup configuration
- callback – function to call after the startup is complete
- arg – callback argument

Returns Zigbee status code whether the startup procedure has been started. If ZB_STATUS_SUCCESS, then the callback is called with the final result.

4.2.1.1 *Stack configuration*

Most stack configuration parameters are controlled by the ZbStartupT structure passed to ZbStartup.

```
struct ZbStartupT config;
```

There are many configuration settings. Most applications want to first initialize the configuration using one of the following:

```
ZbStartupConfigGetProDefaults(&config);
```

or

```
ZbStartupConfigGetProSeDefaults(&config);
```

The first sets the configuration to the Defaults for Zigbee PRO, the second has additional settings for Zigbee Smart Energy applications (SE).

These functions do not set the channel list, which must be set manually by the application. The following example sets the available channels (channel mask) to 11 and 15 on channel page 0 (2.4 GHz):

```
config.channelList.list[0].page = 0;
config.channelList.list[0].channelMask = (uint32_t)((1ul << 11) | (1ul << 15));
config.channelList.count = 1;
```

Each page must have its own mask. The list can contain a total of MAX_CHANNEL_LIST_ENTRIES.

Another common configuration change is the Commissioning mode:

```
config.bdbCommissioningMode |= BDB_COMMISSION_MODE_FIND_BIND;
```

or instead for Touchlink:

```
config.bdbCommissioningMode |= BDB_COMMISSION_MODE_TOUCHLINK;
config.touchlink.flags = 0;config.touchlink.zb_info = ZCL_TL_ZBINFO_TYPE_ROUTER;
config.touchlink.zb_info |= ZCL_TL_ZBINFO_RX_ON_IDLE;
```

A detailed description of all the configuration settings is done in a separate application note.

4.2.2 ZbStartupRejoin

ZbStartupRejoin is a wrapper for ZbNlmeJoinReq(ZB_NWK_REJOIN_TYPE_NWKREJOIN). Use ZbStartupRejoin instead, because the internal startup handler restarts any timers needed to maintain our parent. It must already be connected to a network. If not connected on a network and the user wants to rejoin as a way to connect, use ZbStartup with ZbStartTypeRejoin.

Prototype

```
enum ZbStatusCodeT ZbStartupRejoin(struct ZigBeeT *zb, void (*callback)(ZbNlmeJoinConfT
*conf, void *arg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure
- callback – function to call on completion
- cbarg – callback argument

4.2.3 ZbTrustCenterRejoin

In case of an unsecured rejoin after already joined to a network, call this if it is determined that a key update may have been missed. The parent issues an Update_Device to the TC (trust center), and the TC must send us the (updated) NWK key.

Prototype

```
enum ZbStatusCodeT ZbTrustCenterRejoin(struct ZigBeeT *zb, void (*callback)(enum
ZbStatusCodeT status, void *arg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure
- callback – function to call on completion
- cbarg – callback argument

4.2.4 ZbStartupTouchlinkTargetStop

If Touchlink Target is started with ZbStartup, this API can be used to stop it.

Prototype

```
enum ZbStatusCodeT ZbStartupTouchlinkTargetStop(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns status.

4.2.5 ZbStartupFindBindStart

It manually starts the finding and binding. The finding and binding are also started automatically after joining the network. See 8.5 and 8.6 in [R3].

Prototype

```
enum ZbStatusCodeT ZbStartupFindBindStart(struct ZigBeeT *zb, void (*callback)(enum
ZbStatusCodeT status, void *arg), void *arg);
```

Parameters

- zb – Zigbee stack structure

Returns status.

4.2.6 ZbStartupFindBindStartEndpoint

Same as ZbStartupFindBindStart, but only for a single endpoint. See 8.5 and 8.6 in [R3].

Prototype

```
enum ZbStatusCodeT ZbStartupFindBindStartEndpoint(struct ZigBeeT *zb, uint8_t endpoint, void
(*callback)(enum ZbStatusCodeT status, void *arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- endpoint - Endpoint to perform the finding and binding from
- arg - Application callback argument

Returns status.

4.2.7 ZbStartupConfigGetProDefaults

Gets the default configuration for a PRO network.

Prototype

```
void ZbStartupConfigGetProDefaults(struct ZbStartupT *configPtr);
```

Parameters

- configPtr – startup configuration

4.2.8 ZbStartupConfigGetProSeDefaults

Gets the default configuration for a Smart Energy (SE) network, and clears the preconfigured global link keys.

Prototype

```
void ZbStartupConfigGetProSeDefaults(struct ZbStartupT *configPtr);
```

Parameters

- configPtr – startup configuration

4.2.9 ZbStartupTcsoStart

The applications can call ZbStartupTcsoStart if they have lost communication with the trust center. The trust center swap out process is then performed. The callback status is set to ZB_STATUS_SUCCESS if the stack is operational.

Prototype

```
bool ZbStartupTcsoStart(struct ZigBeeT *zb, void (*callback)(enum ZbTcsoStatusT status, void
*arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- arg - Application callback argument

True if TCSO is started and to wait for callback to indicate completion. False if TCSO is not started.

4.2.10 ZbStartupTcsoAbort

Aborts a TCSO, if it is running. This function should not be necessary, but an application may require it.

Prototype

```
bool ZbStartupTcsoAbort(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

True if TCSO is aborted, false otherwise (it is not running).

4.2.11 ZbDestroy

Deallocates a Zigbee stack instance. This function is for diagnostics only. This function should not be called multiple times.

Prototype

```
void ZbDestroy(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

4.2.12 ZbSeedRand

Helps seed the stack's PRNG. If the data has real entropy, set the has_entropy flag to true.

Prototype

```
void ZbSeedRand(struct ZigBeeT *zb, uint8_t *randBuf, unsigned int len, bool has_entropy);
```

Parameters

- zb – Zigbee stack structure
- randBuf – buffer of entropy data
- len – length of seed number

has_entropy – has entropy

4.2.13 ZbChangeExtAddr

Configures the Zigbee stack and attached interfaces with the extended address provided.

Prototype

```
void ZbChangeExtAddr(struct ZigBeeT *zb, uint64_t extAddr);
```

Parameters

zb – Zigbee stack structure

extAddr – exit address

void Zb returns true on success, false otherwise.

4.2.14 Reset

A helper function to perform an APS and NWK reset.

1. Resets the APS layer
   a. resets the APS IB (including binding table, groups table, key pairs, channel list)
   b. releases any outstanding APS frames
   c. sends internal stack messages to reset other components (e.g. ZCL) back to defaults: ZB_MSG_FILTER_FACTORY_RESET and ZB_MSG_FILTER_RESET_REPORTS
2. Performs an NLME-RESET.request

Prototype

```
void ZbReset(struct ZigBeeT *zb);
```

Parameters

zb – Zigbee stack structure

4.2.15 ZbGetLogging

Retrieves the current logging configuration to the stack (mask and callback).

Prototype

```
oid ZbGetLogging(struct ZigBeeT *zb, uint32_t *mask,
void(**func)(struct ZigBeeT *zb, uint32_t mask, const char *hdr, const char *fmt, va_list
argptr));
```

Parameters

- zb - zigBee stack instance
- mask - log mask bits. There are some predefined masks:
  - ZB_LOG_MASK_LEVEL_0 = fatal errors
  - ZB_LOG_MASK_LEVEL_1 = level_0 + run-time errors
  - ZB_LOG_MASK_LEVEL_2 = level_1 + run-time warnings
  - ZB_LOG_MASK_LEVEL_3 = level_2 + general info
  - ZB_LOG_MASK_LEVEL_4 = level_3 + ZCL cluster debug
  - ZB_LOG_MASK_LEVEL_5 = level_4 + general debug
  - ZB_LOG_MASK_LEVEL_6 = all
- func - function callback. Arguments:
  - zb - stack pointer
  - mask - log mask that generated this message (for example ZB_LOG_MASK_INFO)
  - hdr - header string (typically function name that generated debug message)
  - fmt - format string
  - argptr - va_list arguments (stdarg.h)

4.3 Persistence

To maintain a device's network parameters through a time of shutdown, the data can be stored in a buffer and used when the device is started again.

4.3.1 ZbPersistNotifyRegister

Registers a callback that notifies the application every time persistence data should be saved by calling ZbPersistGet. The callback can be disabled by setting it to NULL.

Prototype

```
bool ZbPersistNotifyRegister(struct ZigBeeT *zb, void (*callback)(struct ZigBeeT *zb, void
*cbarg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure
- callback – function to register as callback, or NULL to disable persistence
- cbarg – callback argument

Returns true on success, false otherwise.

4.3.2 ZbPersistGet

The function populates a buffer with network persistent data. This buffer should be stored in non-volatile memory for it to be available for when the device is restarted.

Prototype

```
unsigned int ZbPersistGet(struct ZigBeeT *zb, uint8_t *buf, unsigned int maxlen);
```

Parameters

- zb – Zigbee stack structure
- buf – (OUT) buffer to store persistant data, set to NULL to get the current size of persistant data
- maxlen – maximum number of bytes that can be written to buf. If buf is NULL, may be 0 (infinite) or set to the maximum size available for persistent data storage

Returns length of persistent data or 0 for no data or error.

4.3.3 ZbStartupPersist

When the device is to be restarted with the persistent data, the data is passed in as a buffer. The function parses the buffer for network parameters and rejoins the network.

Prototype

```
enum ZbStatusCodeT ZbStartupPersist(struct ZigBeeT *zb, const void *pdata,
 unsigned int plen, struct ZbStartupCbkeT *cbke_config);
```

Parameters

- zb – Zigbee stack structure
- pdata – persist data buffer
- plen – length of buffer
- cbke_config - Optional pointer to CBKE configuration data structure.

Returns status.

4.3.4 ZbLeaveReq

This is a wrapper function to external APIs to help with leaving a network and putting the stack into an initialized state.

If joined to a network, this first performs an NLME-LEAVE.request (ZbNlmeLeaveReq) to leave the network.

Once the leave is completed, or if not joined to a network, ZbReset is called to reset the network configuration back to an initialized state.

Prototype

```
enum ZbStatusCodeT ZbLeaveReq(struct ZigBeeT *zb, void (*callback)(struct ZbNlmeLeaveConfT
*conf, void *arg), void *cbarg);
```

Parameters

- zb - Zigbee stack structure
- callback – function to call on completion
- cbarg – callback argument

4.4 Enumerations

4.4.1 ZbStartType

The startup control-codes as listed in [R3], are commissioning the cluster's startup control attribute in the startup parameters attribute set.

ZbStartTypePreconfigured	(0x00)Preconfigured. No explicit form, join or rejoin to be performed.
ZbStartTypeForm	(0x01) form network
ZbStartTypeRejoin	(0x02) rejoin network
ZbStartTypeJoin	(0x03) join network
ZbStartTypeTouchlink	[Internal stack use only] touchlink
ZbStartTypeFindBind	[Internal stack use only] finding and binding

4.5 Stack startup structures

4.5.1 ZbStartupCbkeT

CBKE configuration parameters for ZbStartup. This configuration is only applicable if the 'suite_mask' is non-zero.

Parameters

- uint8_t endpoint: endpoint to assign ZCL Key Exchange cluster. Default is ZB_ENDPOINT_CBKE_DEFAULT (240)
- uint16_t deviceId: device Id to assign to the endpoint created for the ZCL Key Exchange cluster. Default is ZCL_DEVICE_METER
- uint16_t suite_mask: the key exchange suite bitmask. E.g. ZCL_KEY_SUITE_CBKE2_ECMQV for CBKE version 2 (cbke_v2).
- struct ZbZclCbkeInfoT cbke_v1: CBKE version 1 certificate and security keys configuration. Only applicable if ZCL_KEY_SUITE_CBKE_ECMQV is set in suite_mask.
- struct ZbZclCbke2InfoT cbke_v2: CBKE version 2 certificate and security keys configuration. Only applicable if ZCL_KEY_SUITE_CBKE2_ECMQV is set in suite_mask.
- bool tc_keepalive_server_enableIf: CBKE is enabled (suite_mask != 0), this flag determines whether to allocate the Trust Center Keep Alive Server (true) or Client (false).
- uint8_t tc_keepalive_base: trust center keep alive server 'Base' attribute value in minutes. If zero, let the stack choose a default value.
- uint16_t tc_keepalive_jitter: trust center keep alive server 'Jitter' attribute value in seconds. If zero, let the stack choose a default value.
- tcso_callback (callback) void (*tcso_callback)(enum ZbTcsoStatusT status, void *arg): application callback that is called for any trust center swap out (TCSO) process initiated by the keep alive client cluster. This includes when a TCSO has been started and the final outcome of a TCSO process.
- void *tcso_arg

4.5.2 ZbStartupT

The set of configuration parameters used to form or join a network using ZbStartup. Must be first initialized using ZbStartupConfigGetProDefaults or ZbStartupConfigGetProSeDefaults.

Parameters

- uint16_t shortAddress: network short address. Only applicable if startupControl is ZbStartTypePreconfigured or ZbStartTypeRejoin
- uint16_t panId: network PAN Id. Only applicable if startupControl is ZbStartTypePreconfigured or ZbStartTypeRejoin
- uint16_t networkManagerAddress: network manager address. Only applicable if startupControl is ZbStartTypePreconfigured or ZbStartTypeRejoin
- uint64_t extendedPanId: extended PAN Id.
  - If startupControl is ZbStartTypeForm and extendedPanId is zero, then the device's extended address (EUI) is used.
  - If startupControl is ZbStartTypeRejoin a non-zero extendedPanId must be provided.
  - If startupControl is ZbStartTypeJoin, the device only attempts to join a network matching the extendedPanId.
  - If extendedPanId is zero, then the device attempts to join any available network, sorted by LQI.
  - If startupControl is ZbStartTypePreconfigured a non-zero extendedPanId must be provided.
- struct ZbChannelListT channelList: specify the channel mask(s) to use.
  - If no channel masks are specified, ZB_BDB_PrimaryChannelSet and ZB_BDB_SecondaryChannelSet are used instead.
- uint8_t stackProfile: network stack profile.
  - If not ZB_NWK_STACK_PROFILE_PRO, the application must configure the following NIB parameters before calling ZbStartup: nwkMaxDepth, nwkMaxChildren, nwkReportConstantCost, nwkLinkStatusPeriod, nwkTransactionPersistenceTime, nwkPassiveAckTimeout, nwkMaxBroadcastRetries, nwkSecureAllFrames, nwkSecurityLevel
- uint8_t bdbCommissioningModeBDB Commissioning Mode mask. Set to 0 by default. If BDB_COMMISSION_MODE_TOUCHLINK is set, then Touchlink is used. If BDB_COMMISSION_MODE_FIND_BIND is set, then finding and binding are used after joining.
- enum ZbStartType startupControlStartup Type. Not applicable if BDB_COMMISSION_MODE_TOUCHLINK commissioning mode is set. If startup type is ZbStartTypeJoin, the ZB_APS_IB_ID_SCAN_COUNT attribute is used to control the number of scans to perform while searching for a network to join. Similarly, the ZB_BDB_ScanDuration attribute is used to configure the MAC scan duration to use for scans during network joining and forming.
- uint8_t levelSecurity Level. Default is 0x05.
- bool useInsecureRejoin: configures ZB_APS_IB_ID_USE_INSECURE_JOIN.
- uint64_t trustCenterAddress: configures ZB_APS_IB_ID_TRUST_CENTER_ADDRESS. If forming the network and assuming the role of Trust Center, the device's extended address is used instead, unless this parameter has been explicitly set to ZB_DISTRIBUTED_TC_ADDR.
- uint8_t preconfiguredLinkKey: preconfigured Link Key
- uint8_t distributedGlobalKey: preconfigured Distributed Global Link Key
- uint8_t networkKey: configures the Network Key with key type set to ZB_SEC_KEYTYPE_STANDARD_NWK. Only applicable if startupControl is ZbStartTypePreconfigured.
- uint8_t networkKeySeqNum: configures the Network Key Sequence Number for the Network Key. Also sets ZB_NWK_NIB_ID_ActiveKeySeqNumber to this value. Only applicable if startupControl is ZbStartTypePreconfigured.
- enum ZbSecKeyTypeT networkKeyType: deprecated and not used.
- struct ZbStartupCbkeT cbke: CBKE certificate configuration
- uint16_t timeBetweenScans: deprecated and not used.
- uint16_t rejoinInterval: deprecated and not used.
- uint16_t maxRejoinInterval: deprecated and not used.
- uint8_t capability: device capability mask. Default value includes: MCP_ASSOC_CAP_PWR_SRC (mains power), MCP_ASSOC_CAP_RXONIDLE (radio receiver is on, not sleepy), MCP_ASSOC_CAP_ALLOC_ADDR (parent allocates network address), MCP_ASSOC_CAP_DEV_TYPE (full function device)

- uint8_t endDeviceTimeout: end-device timeout is only used by end-devices. It configures the time used to periodically update the Parent device so this device is not removed from the Parent's NWK neighbor table. Configures ZB_NWK_NIB_ID_EndDeviceTimeoutDefault. Timeout = (60 * 2^n) seconds for n > 0. If n = 0, timeout = 10 seconds. Setting to ZB_NWK_CONST_ENDDEV_TIMEOUT_DISABLED (0xff) disables end-device timeout.
- uint16_t fastPollPeriod: configures ZB_NWK_NIB_ID_FastPollPeriod.
- uint8_t tl_endpoint: endpoint for the Touchlink Cluster (e.g.)
- uint8_t bind_endpoint: endpoint to use when binding clusters from Initiator to Target.
- uint16_t deviceId: such as ZCL_DEVICE_ONOFF_SWITCH
- uint8_t zb_info: such as ZCL_TL_ZBINFO_TYPE_ROUTER
- uint8_t flags: such as ZCL_TL_FLAGS_IS_TARGET
- const void *persist_buf: pointer to persistence data. Only applicable if ZCL_TL_ZBINFO_USE_PERSIST flag is set in zb_info.
- unsigned int persist_len: length of persistence data.

5 Information bases

5.1 BDB functions

Functions: ZbBdbGet / ZbBdbSet
Attribute IDs are found in zigbee.nwk.h in "enum ZbBdbAttrIdT".

Table 11. BDB functions

Attribute	Description
ZB_BDB_CommissioningMode	-
ZB_BDB_JoiningNodeEui64	-
ZB_BDB_JoiningNodeNewTCLinkKey	-
ZB_BDB_JoinUsesInstallCodeKey	-
ZB_BDB_NodeCommissioningCapability	-
ZB_BDB_NodeIsOnANetwork	-
ZB_BDB_NodeJoinLinkKeyType	-
ZB_BDB_PrimaryChannelSet	-
ZB_BDB_ScanDuration	-
ZB_BDB_SecondaryChannelSet	-
ZB_BDB_TCLK_ExchangeAttempts	-
ZB_BDB_TCLK_ExchangeAttemptsMax	-
ZB_BDB_TCLinkKeyExchangeMethod	-
ZB_BDB_TrustCenterNodeJoinTimeout	-
ZB_BDB_TrustCenterRequiresKeyExchange	-
ZB_BDB_AcceptNewUnsolicitedTCLinkKey	-
ZB_BDB_AcceptNewUnsolicitedApplicationLinkKey	-
ZB_BDB_JoiningNodeParent	-
ZB_BDB_vDoPrimaryScan	-
ZB_BDB_FreeNetAddrBegin	-
ZB_BDB_FreeNetAddrCurrent	-
ZB_BDB_FreeNetAddrEnd	-
ZB_BDB_FreeGroupIDBegin	-
ZB_BDB_FreeGroupIDEnd	-
ZB_BDB_TLRssiMin	-
ZB_BDB_TLTestFlags	-
ZB_BDB_UpdateDeviceKeyId	-
ZB_BDB_JoinScanType	-
ZB_BDB_JoinIgnoreLqi	-
ZB_BDB_NlmeSyncFailNumBeforeError	-
ZB_BDB_ZdoTimeout	-
ZZB_BDB_TLStealFlags	-
ZB_BDB_JoinTclkNodeDescReqDelay	-

Attribute	Description
ZB_BDB_JoinTclkRequestKeyDelay	-
ZB_BDB_TLDenyFactoryNew	-
ZB_BDB_TLKey	-
ZB_BDB_TLKeyIndex	-
ZB_BDB_ZdoPermitJoinAfterJoin	-
ZB_BDB_ZdoZigbeeProtocolRevision	-
ZB_BDB_NwkAllowRouterLeaveRejoin	-
ZB_BDB_PersistTimeoutMs	-
ZB_BDBC_MaxSameNetworkRetryAttempts	-
ZB_BDBC_MinCommissioningTime	-
ZB_BDBC_RecSameNetworkRetryAttempts	-
ZB_BDBC_TCLinkKeyExchangeTimeout	-
ZB_BDBC_TLInterPANTransIdLifetime	-
ZB_BDBC_TLMinStartupDelayTime	-
ZB_BDBC_TLRxWindowDuration	-
ZB_BDBC_TLScanTimeBaseDuration	-

5.2 APS functions

Functions: ZbApsGet / ZbApsSet
Attribute IDs are found in zigbee.aps.h in "enum ZbApsmeIbAttrIdT".

Table 12. APS funtions

Attribute	Description
ZB_APS_IB_ID_ADDRESS_MAP	-
ZB_APS_IB_ID_BINDING_TABLE	-
ZB_APS_IB_ID_DESIGNATED_COORD	-
ZB_APS_IB_ID_CHANNEL_MASK	-
ZB_APS_IB_ID_USE_EPID	-
ZB_APS_IB_ID_GROUP_TABLE	-
ZB_APS_IB_ID_NONMEMBER_RADIUS	-
ZB_APS_IB_ID_USE_INSECURE_JOIN	-
ZB_APS_IB_ID_INTERFRAME_DELAY	-
ZB_APS_IB_ID_LAST_CHANNEL_ENERGY	-
ZB_APS_IB_ID_LAST_CHANNEL_FAILRATE	-
ZB_APS_IB_ID_CHANNEL_TIMER	-
ZB_APS_IB_ID_MAX_WINDOW_SIZE	-
ZB_APS_IB_ID_DEVICE_KEY_PAIR_SET	-
ZB_APS_IB_ID_TRUST_CENTER_ADDRESS	-
ZB_APS_IB_ID_SECURITY_TIMEOUT_PERIOD	-
ZB_APS_IB_ID_TRUST_CENTER_POLICY	-

Attribute	Description
ZB_APS_IB_ID_FRAGMENTATION_THRESH	-
ZB_APS_IB_ID_SCAN_COUNT	-
ZB_APS_IB_ID_LEAVE_REMOVE_CHILDREN	-
ZB_APS_IB_ID_PRECONFIGURED_LINK_KEY	-
ZB_APS_IB_ID_DISTRIBUTED_GLOBAL_KEY	-
ZB_APS_IB_ID_KEY_UPDATE_PERIOD	-
ZB_APS_IB_ID_MANUFACTURER_ID	-

5.3 NWK functions

Functions: ZbNwkGet / ZbNwkSet
Attribute IDs are found in zigbee.nwk.h in "enum ZbNwkNibAttrIdT"

Table 13. NWK functions

Attribute	Description
ZB_NWK_NIB_ID_PanId	-
ZB_NWK_NIB_ID_SequenceNumber	-
ZB_NWK_NIB_ID_PassiveAckTimeout	-
ZB_NWK_NIB_ID_MaxBroadcastRetries	-
ZB_NWK_NIB_ID_MaxChildren	-
ZB_NWK_NIB_ID_MaxDepth	-
ZB_NWK_NIB_ID_MaxRouters	-
ZB_NWK_NIB_ID_NeighborTable	-
ZB_NWK_NIB_ID_NetworkBroadcastDeliveryTime	-
ZB_NWK_NIB_ID_ReportConstantCost	-
ZB_NWK_NIB_ID_RouteDiscoveryRetriesPermitted	-
ZB_NWK_NIB_ID_RouteTable	-
ZB_NWK_NIB_ID_TimeStamp	-
ZB_NWK_NIB_ID_TxTotal	-
ZB_NWK_NIB_ID_SymLink	-
ZB_NWK_NIB_ID_CapabilityInformation	-
ZB_NWK_NIB_ID_AddrAlloc	-
ZB_NWK_NIB_ID_UseTreeRouting	-
ZB_NWK_NIB_ID_ManagerAddr	-
ZB_NWK_NIB_ID_MaxSourceRoute	-
ZB_NWK_NIB_ID_UpdateId	-
ZB_NWK_NIB_ID_TransactionPersistenceTime	-
ZB_NWK_NIB_ID_NetworkAddress	-
ZB_NWK_NIB_ID_StackProfile	-
ZB_NWK_NIB_ID_BroadcastTransactionTable	-
ZB_NWK_NIB_ID_GroupIdTable	-

Attribute	Description
ZB_NWK_NIB_ID_ExtendedPanId	-
ZB_NWK_NIB_ID_UseMulticast	-
ZB_NWK_NIB_ID_RouteRecordTable	-
ZB_NWK_NIB_ID_IsConcentrator	-
ZB_NWK_NIB_ID_ConcentratorRadius	-
ZB_NWK_NIB_ID_ConcentratorDiscoveryTime	-
ZB_NWK_NIB_ID_SecurityLevel	-
ZB_NWK_NIB_ID_SecurityMaterialSet	-
ZB_NWK_NIB_ID_ActiveKeySeqNumber	-
ZB_NWK_NIB_ID_AllFresh	-
ZB_NWK_NIB_ID_SecureAllFrames	-
ZB_NWK_NIB_ID_LinkStatusPeriod	-
ZB_NWK_NIB_ID_RouterAgeLimit	-
ZB_NWK_NIB_ID_UniqueAddr	-
ZB_NWK_NIB_ID_AddressMap	-
ZB_NWK_NIB_ID_Depth	-
ZB_NWK_NIB_ID_FrameCounterSet	-
ZB_NWK_NIB_ID_RouteDiscoverySendDelay	-
ZB_NWK_NIB_ID_FastPollPeriod	-
ZB_NWK_NIB_ID_SlowPollPeriod	-
ZB_NWK_NIB_ID_FrameCounterCooldown	-
ZB_NWK_NIB_ID_OutgoingCounter	-
ZB_NWK_NIB_ID_PersistCounter	-
ZB_NWK_NIB_ID_LeaveRequestAllowed	-
ZB_NWK_NIB_ID_ParentInformation	-
ZB_NWK_NIB_ID_EndDeviceTimeoutDefault	-
ZB_NWK_NIB_ID_LeaveRequestWithoutRejoinAllowed	-
ZB_NWK_NIB_ID_TxPowerMgmtSupported	-
ZB_NWK_NIB_ID_LinkPowerDeltaPeriod	-
ZB_NWK_NIB_ID_JoiningListUpdateId	-
ZB_NWK_NIB_ID_JoiningPolicy	-
ZB_NWK_NIB_ID_JoiningListTotal	-
ZB_NWK_NIB_ID_JoiningListExpiryInterval	-
ZB_NWK_NIB_ID_ActiveChannelList	-
ZB_NWK_NIB_ID_PermitJoinCounter	-
ZB_NWK_NIB_ID_DiscoveryTable	-

6 BDB – base device behavior

6.1 ZbBdbCommissionModeBitSupported

Returns whether or not the specified BDB commission mode bit is supported.
Prototype

```
bool ZbBdbCommissionModeBitSupported(struct ZigBeeT *zb, uint8_t new_mode_bit);
```

Parameters

- zb – Zigbee stack structure
- new_mode_bit – mode bit

Returns true on success, false otherwise.
For more information, refer to the table 4 in [R2].

6.2 ZbBdbCommissionModeBitSet

Sets the BDB commission mode.
Prototype

```
enum ZbStatusCodeT ZbBdbCommissionModeBitSet(struct ZigBeeT *zb,
 uint8_t new_mode_bit);
```

Parameters

- zb – Zigbee stack structure
- new_mode_bit – mode bit

Returns status.
For more information, refer to the table 4 in [R2].

6.3 ZbBdbCommissionModeBitClear

Clears the BDB commissioning mode.
Prototype

```
enum ZbStatusCodeT ZbBdbCommissionModeBitClear(struct ZigBeeT *zb,
uint8_t new_mode_bit);
```

Parameters

- zb – Zigbee stack structure
- new_mode_bit – mode bit

Returns status.
For more information, refer to the table 4 in [R2].

6.4 ZbBdbGetEndpointStatus

Gets the BDB endpoint status usually resulting from find and bind.
Prototype

```
int ZbBdbGetEndpointStatus(struct ZigBeeT *zb, uint8_t endpoint);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint

Returns BDB Status code or -1 if the endpoint is not found.

For more information, refer to section 5.3.3 of [R2].

6.5 ZbBdbSetEndpointStatus

Sets the endpoint status.
Prototype

```
void ZbBdbSetEndpointStatus(struct ZigBeeT *zb,
        enum ZbBdbCommissioningStatusT status, uint8_t endpoint);
```

Parameters

- zb – Zigbee stack structure
- status – BDB status
- endpoint – endpoint

For more information, refer to section 5.3.3 of [R2].

6.6 ZbBdbGetIndex

Used to read BDB IB attributes.
Prototype

```
enum ZbStatusCodeT ZbBdbGetIndex(struct ZigBeeT *zb, enum ZbBdbAttrIdT attrId, void
*attrPtr, unsigned int attrSz, unsigned int attrIndex);
```

Parameters

- zb – Zigbee stack structure
- attrId – attribute ID
- attrPtr – pointer to the attribute
- attrSz – size of the attribute

Returns the status code.

6.7 ZbBdbSetIndex

Used to write BDB IB attributes.
Prototype

```
enum ZbStatusCodeT ZbBdbSetIndex(struct ZigBeeT *zb, enum ZbBdbAttrIdT attrId, const void
*attrPtr, unsigned int attrSz, unsigned int attrIndex);
```

Parameters

- zb – Zigbee stack structure
- attrId – attribute ID
- attrPtr – pointer to the attribute
- attrSz – size of the attribute

Returns the status code.

7 APS – Application services

7.1 APSME

7.1.1 ZbApsGet

Gets an APS information base attribute.

Prototype

```
enum ZbStatusCodeT ZbApsGet(struct ZigBeeT *zb, enum ZbApsmeIbAttrIdT attrId,
        void *attrPtr, unsigned int attrSz);
```

Parameters

- zb – Zigbee stack structure
- attrId – attribute identifier to get
- attrPtr – (OUT) buffer to write the attribute into
- attrSz – size of the attribute

Returns the status.

7.1.2 ZbApsSet

Sets an APS information base attribute.

Prototype

```
enum ZbStatusCodeT ZbApsSet(struct ZigBeeT *zb, enum ZbApsmeIbAttrIdT attrId,
        const void *attrPtr, unsigned int attrSz);
```

Parameters

- zb – Zigbee stack structure
- attrId – attribute identifier to set
- attrPtr – value to write into the AIB
- attrSz – size of the attribute

Returns the status.

7.1.3 ZbApsmeGetReq

Performs the APSME-GET.request. The confirm message is returned via the getConfPtr parameter.

Prototype

```
void ZbApsmeGetReq(struct ZigBeeT *zb, ZbApsmeGetReqT *get,
        ZbApsmeGetConfT *getConfPtr);
```

Parameters

- zb - Zigbee stack structure
- getReqPtr – APSME-GET.request
- getConfPtr – APSME-GET.confirm

For more information, refer to section 2.2.4.4.1 of [R1].

7.1.4 ZbApsmeSetReq

Performs the APSME-SET.request. The confirm message is returned via the setConfPtr parameter.

Prototype

```
void ZbApsmeSetReq(struct ZigBeeT *zb, ZbApsmeSetReqT *setReqPtr,
        ZbApsmeSetConfT *setConfPtr);
```

Parameters

- zb – Zigbee stack structure
- setReqPtr – APSME-SET.request
- setConfPtr – APSME-SET.confirm

For more information, refer to section 2.2.4.4.3 of [R1].

7.1.5 ZbApsmeBindReq

Adds an entry to the stack's binding table. The size of the binding table is determined in the tableSizes parameter to ZbInit(). The binding table is maintained by the stack.

Prototype

```
void ZbApsmeBindReq(struct ZigBeeT *zb, ZbApsmeBindReqT *bindReqPtr,
        ZbApsmeBindConfT *bindConfPtr);
```

Parameters

- zb – Zigbee stack structure
- bindReqPtr – APSME-BIND.request
- bindConfPtr – APSME-BIND.confirm

For more information, refer to section 2.2.4.3.1 of [R1].

7.1.6 ZbApsmeUnbindReq

Removes an entry from the stack's binding table. The size of the binding table is determined in the tableSizes parameter to ZbInit(). The binding table is maintained by the stack.

Prototype

```
void ZbApsmeUnbindReq(struct ZigBeeT *zb, ZbApsmeUnbindReqT *unbindReqPtr,
        ZbApsmeUnbindConfT *unbindConfPtr);
```

Parameters

- zb – Zigbee stack structure
- unbindReqPtr – APSME-UNBIND.request
- unbindConfPtr – APSME-UNBIND.confirm

For more information, refer to section 2.2.4.3.3 of [R1].

7.1.7 ZbApsUnbindAllReq

Removes all entries from the binding table.

Prototype

```
void ZbApsUnbindAllReq(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

7.1.8 ZbApsmeAddGroupReq

Adds an entry to group table.

Prototype

```
void (struct ZigBeeT *zb, ZbApsmeAddGroupReqT *r, ZbApsmeAddGroupConfT *c);
```

Parameters

- zb – Zigbee stack structure
- r – APSME-ADD-GROUP.request
- c – APSME-ADD-GROUP.confirm

For more information, refer to section 2.2.4.5.1 of [R1].

7.1.9 ZbApsmeRemoveGroupReq

Removes an entry from the group table.

Prototype

```
void ZbApsmeRemoveGroupReq(struct ZigBeeT *zb, ZbApsmeRemoveGroupReqT *r,
        ZbApsmeRemoveGroupConfT *c);
```

Parameters

- zb – Zigbee stack structure
- r – APSME-REMOVE-GROUP.request
- c – APSME-REMOVE-GROUP.confirm

For more information, refer to section 2.2.4.5.3 of [R1].

7.1.10 ZbApsmeRemoveAllGroupsReq

Removes all entries from the group table.

Prototype

```
void ZbApsmeRemoveAllGroupsReq(struct ZigBeeT *zb, ZbApsmeRemoveAllGroupsReqT *r,
        ZbApsmeRemoveAllGroupsConfT *c);
```

Parameters

- zb – Zigbee stack structure
- r – APSME-REMOVE-ALL-GROUPS.request
- c – APSME-REMOVE-ALL-GROUPS.confirm

For more information, refer to section 2.2.4.5.5 of [R1].

7.1.11 ZbApsmeAddEndpoint

Registers an endpoint with the Zigbee stack. This is typically called during initialization. This should be safe to call from an init function.

Prototype

```
void ZbApsmeAddEndpoint(struct ZigBeeT *zb, ZbApsmeAddEndpointReqT *addReqPtr,
        ZbApsmeAddEndpointConfT *addConfPtr);
```

Parameters

- zb – Zigbee stack structure
- addReqPtr – APSME-ADD-ENDPOINT.request
- addConfPtr – APSME-ADD-ENDPOINT.confirm

7.1.12 ZbApsmeRemoveEndpoint

Unregisters and cleans up an endpoint from the APS endpoint table. Typically this is called when terminating an app.

Prototype

```
void ZbApsmeRemoveEndpoint(struct ZigBeeT *zb, ZbApsmeRemoveEndpointReqT *r,
        ZbApsmeRemoveEndpointConfT *c);
```

Parameters

- zb – Zigbee stack structure
- r – APSME-REMOVE-ENDPOINT.request
- c – APSME-REMOVE-ENDPOINT.confirm

7.1.13 ZbApsmeEndpointClusterListAppend

Adds a cluster to an endpoint’s cluster list.

Prototype

```
bool ZbApsmeEndpointClusterListAppend(struct ZigBeeT *zb, uint8_t endpoint,
        uint16_t cluster_id, bool is_input);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint
- cluster_id – ID of cluster to be added
  - is_input – true if input cluster, false if output cluster

Returns true on success, false otherwise.

7.1.14 ZbApsmeEndpointClusterListRemove

Removes a cluster from an endpoint's cluster list.

Prototype

```
bool ZbApsmeEndpointClusterListRemove(struct ZigBeeT *zb, uint8_t endpoint,
        uint16_t cluster_id, bool is_input);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint
  - cluster_id – ID of cluster to be removed
  - is_input – true if input cluster, false if output cluster

Returns true on success, false otherwise.

7.1.15 ZbApsmeEndpointClusterPresent

Determines if a cluster is present on the specified endpoint.

Prototype

```
bool ZbApsmeEndpointClusterPresent(struct ZigBeeT *zb, uint8_t endpoint,
        uint16_t cluster_id, bool is_input);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint

cluster_id – ID of cluster to be checked

is_input – true if input cluster, false if output cluster

Returns true on success, false otherwise.

7.1.16 ZbApsmeAddKeyReq

Implements APSME-ADD-KEY.request.

Prototype

```
void ZbApsmeAddKeyReq(struct ZigBeeT *zb, ZbApsmeAddKeyReqT *req,
        ZbApsmeAddKeyConfT *conf);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-ADD-KEY.request
- conf – APSME-ADD-KEY.confirm

7.1.17 ZbApsmeGetKeyReq

Implements APSME-GET-KEY.request.

Prototype

```
void ZbApsmeGetKeyReq(struct ZigBeeT *zb, ZbApsmeGetKeyReqT *req,
        ZbApsmeGetKeyConfT *conf);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-GET-KEY.request
- conf – APSME-GET-KEY.confirm

7.1.18 ZbApsmeRemoveKeyReq

Implements APSME-REMOVE-KEY.request.

Prototype

```
void ZbApsmeRemoveKeyReq(struct ZigBeeT *zb, ZbApsmeRemoveKeyReqT *req,
        ZbApsmeRemoveKeyConfT *conf);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-REMOVE-KEY.request

7.1.19 ZbApsmeConfirmKeyReq

Implements APSME-CONFIRM-KEY.request.

Prototype

```
void ZbApsmeConfirmKeyReq(struct ZigBeeT *zb, ZbApsmeConfirmKeyReqT *req);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-CONFIRM-KEY.request

For more information, refer to section 4.4.9.1 of [R1]].

7.1.20 ZbApsmeRemoveDeviceReq

Implements the APSME-REMOVE-DEVICE.request.

Prototype

```
void ZbApsmeRemoveDeviceReq(struct ZigBeeT *zb, ZbApsmeRemoveDeviceReqT *req);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-REMOVE-DEVICE.request

For more information, refer to section 4.4.4.1 of [R1].

7.1.21 ZbApsmeRequestKeyReq

Implements APSME-REQUEST-KEY.request.

Prototype

```
enum ZbStatusCodeT ZbApsmeRequestKeyReq(struct ZigBeeT *zb, ZbApsmeRequestKeyReqT *req);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-REQUEST-KEY.request

Returns status.

For more information, refer to section 4.4.5.1 of [R1]].

7.1.22 ZbApsmeSwitchKeyReq

Implements APSME-SWITCH-KEY.request.

Prototype

```
void ZbApsmeSwitchKeyReq(struct ZigBeeT *zb, ZbApsmeSwitchKeyReqT *req);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-SWITCH-KEY.request

7.1.23 ZbApsmeTransportKeyReq

Implements APSME-TRANSPORT-KEY.request.

Prototype

```
void ZbApsmeTransportKeyReq(struct ZigBeeT *zb, ZbApsmeTransportKeyReqT *req);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-TRANSPORT-KEY.request

For more information, refer to section 4.4.2.1 of [R1].

7.1.24 ZbApsmeUpdateDeviceReq

Implements APSME-UPDATE-DEVICE.request.

Prototype

```
enum ZbSecHdrKeyIdT
        ZbApsmeUpdateDeviceReq(struct ZigBeeT *zb, ZbApsmeUpdateDeviceReqT *req);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-UPDATE-DEVICE.request

Returns the security used to send update device.

For more information, refer to section 4.4.3.1 of [R1].

7.1.25 ZbApsmeVerifyKeyReq

Implements the APSME-VERIFY-KEY.request.

Prototype

```
void ZbApsmeVerifyKeyReq(struct ZigBeeT *zb, ZbApsmeVerifyKeyReqT *req);
```

Parameters

- zb – Zigbee stack structure
- req – APSME-VERIFY-KEY.request

For more information, refer to section 4.4.7.1 of [R1].

7.2 APSDE

7.2.1 ZbApsdeDataReqCallback

Sends an APSDE-DATA.request If the packet is queued, ZB_STATUS_SUCCESS is returned. Otherwise, an error status is returned.

Prototype

```
enum ZbStatusCodeT ZbApsdeDataReqCallback(struct ZigBeeT *zb, struct ZbApsdeDataReqT *req,
void (*callback)(struct ZbApsdeDataConfT *conf, void *arg), void *arg);
```

Parameters

- zb – Zigbee instance

- Req – APSDE-DATA.request data structure
  - callback – Application callback function to call for APSDE-DATA.confirm. May be set to NULL.
  - arg – Application callback argument, which is passed to callback()

Returns the Zigbee status code

7.3 Utility

7.3.1 ZbApsAddrIsBcast

Checks if the addess is set to broadcast.
Prototype

```
bool ZbApsAddrIsBcast(struct ZbApsAddrT *addr);
```

Parameters

- addr – address

Returns true if address is set to broadcast, false otherwise.

7.3.2 ZbApsAddrIsLocal

Checks if the given address is that of the provided Zigbee stack structure.
Prototype

```
bool ZbApsAddrIsLocal(struct ZigBeeT *zb, struct ZbApsAddrT *addr);
```

Parameters

- zb – Zigbee stack structure
- addr – address to check

Returns true if address is local, false otherwise.

7.3.3 ZbApsCommandSecurityCheck

Checks if an incoming command has the appropriate security level.
Prototype

```
bool ZbApsCommandSecurityCheck(struct ZigBeeT *zb, enum ZbApsCmdIdT cmdId,
        uint64_t srcExtAddr, enum ZbSecEncryptT encryptKeyType);
```

Parameters

- zb – Zigbee stack structure
- cmdId – command ID
- srcExtAddr – source extended address
- encryptKeyType – key type

Returns true if the command is allowed, false otherwise.
For more information, refer to table 4-6 in [R1].

7.3.4 ZbApsBindTblNumEntries

Retrieves the number of entries in the binding table.
Prototype

```
uint8_t ZbApsBindTblNumEntries(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns number of entries.

7.3.5 ZbApsEndpointExists

Determines if the provided endpoint exists in the stack. The endpoint was previously created by the application.

Prototype

```
bool ZbApsEndpointExists(struct ZigBeeT *zb, uint8_t endpoint);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint

Returns true if endpoint is found, false otherwise.

7.3.6 ZbApsEndpointProfile

Retrieves the profile ID of an endpoint.

Prototype

```
uint16_t ZbApsEndpointProfile(struct ZigBeeT *zb, uint8_t endpoint);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint

Returns profile ID, or ZCL_PROFILE_WILDCARD (0xffff) on error.

For more information, refer to section 2.6.1.1 of [R3].

7.3.7 ZbApsFilterEndpointAdd

Creates an APS indication filter for a specific endpoint, with no specific cluster being filtered.

Prototype

```
struct ZbApsFilterT * ZbApsFilterEndpointAdd(struct ZigBeeT *zb, uint8_t endpoint, uint16_t
profileId, int (*callback)(struct ZbApsdeDataIndT *dataInd, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee instance
- endpoint – Endpoint Id to match with incoming packets
- profileId – Profile Id to match with incoming packets
- callback – APSDE-DATA.indication callback if the filter matches the incoming packet
- arg – Application callback argument, which is passed to callback()

7.3.8 ZbApsFilterClusterAdd

Creates an APSDE-DATA.indication filter for a specific endpoint, with no specific cluster being filtered. For any incoming APS data packets that match this filter, the provided 'callback' function is called.

Prototype

```
struct ZbApsFilterT * ZbApsFilterClusterAdd(struct ZigBeeT *zb, uint8_t endpoint, uint16_t
clusterId, uint16_t profileId, void (*callback)(ZbApsdeDataIndT *dataInd, void *cb_arg),
void *arg);
```

Parameters

- Zb - Zigbee instance
- endpoint - Endpoint Id to match with incoming packets.
- profileId - Profile Id to match with incoming packets.
- Callback - APSDE-DATA.indication callback if the filter matches the incoming packet.
- Arg - Application callback argument, which is passed to callback().

7.3.9 ZbApsFilterEndpointFree

Removes a filter created by ZbApsFilterEndpointAdd or ZbApsFilterClusterAdd.

Prototype

```
void ZbApsFilterEndpointRemove(struct ZigBeeT *zb, struct ZbApsFilterT *filter);
```

Parameters

- zb – Zigbee instance
- filter – Filter to remove

7.3.10 ZbApsGroupsGetCapacity

Retrieves the number of free entries in the group table.

Prototype

```
uint8_t ZbApsGroupsGetCapacity(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns the number of free entries in the group table.

For more information, refer to section 2.2.8.3.1 of [R1].

7.3.11 ZbApsGroupsGetMembership

Returns all group addresses on this endpoint.

Prototype

```
uint8_t ZbApsGroupsGetMembership(struct ZigBeeT *zb, uint8_t endpoint,
        uint16_t *group_list, uint8_t max_len);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint
- group_list – (OUT) list of groups on this endpoint
- max_len – maxinum number of groups to be returned

Returns number of group addresses.

For more information, refer to section 2.2.8.3.1 of [R1].

7.3.12 ZbApsGroupIsMember

Checks if the local device is a member of a specified group.

Prototype

```
bool ZbApsGroupIsMember(struct ZigBeeT *zb, uint16_t groupAddr, uint8_t endpoint);
```

Parameters

- zb – Zigbee stack structure
- groupAddr – group address
- endpoint – may be ZB_ENDPOINT_BCAST if the user does not care about the endpoint

Returns true if device is a member, false otherwise.

7.3.13 ZbApsLinkKeyExists

Checks if a link key exists in the AIB.

Prototype

```
bool ZbApsLinkKeyExists(struct ZigBeeT *zb, uint64_t partner);
```

Parameters

- zb – Zigbee stack structure
- partner – partner address

Returns true if key exists, false otherwise.

For more information, refer to section 4.4.11 in [R1].

7.3.14 ZbApsLookupKey

Looks up an APS key pair descriptor from apsDeviceKeyPairSet AIB attribute.

Prototype

```
ZbApsmeKeyPairT * ZbApsLookupKey(struct ZigBeeT *zb, ZbApsmeKeyPairT *key,
        uint64_t addr, unsigned int *idx);
```

Parameters

- zb – Zigbee stack structure
- key – (OUT) key pair descriptor
- addr – partner addresses
- idx – (OUT) index value into device key pair set table for the APS key returned, if successful

Returns key pair on success, or NULL if no such entry is found.

For more information, refer to section 4.4.11 in [R1].

7.4 Enumerations

The following table lists the APSDE and interpan addressing modes

Table 14. APSDE and interpan addressing modes

Code	Mode
ZB_APSDE_ADDRMODE_NOTPRESENT	Binding table addressing
ZB_APSDE_ADDRMODE_GROUP	Group addressing
ZB_APSDE_ADDRMODE_SHORT	Short addressing
ZB_APSDE_ADDRMODE_EXT	Extended addressing
ZB_APSDE_ADDRMODE_IPGROUP	InterPAN

7.5 APS structures

7.5.1 ZbApsAddrT

APS address information data structure.

Parameters

- enum ZbApsAddrModeT mode - Address mode (e.g. short, group, extended)
- uint16_t endpoint - Destination endpoint.
- If set to ZB_ENDPOINT_INTERPAN, then packet is sent as an InterPAN packet.
- uint16_t panId - Destination PAN Id.
- This is for InterPAN packets, when the endpoint == ZB_ENDPOINT_INTERPAN.
- uint16_t nwkAddr - Destination network address when mode == ZB_APSDE_ADDRMODE_SHORT, ZB_APSDE_ADDRMODE_GROUP or ZB_APSDE_ADDRMODE_IPGROUP.
- uint64_t extAddr - Destination extended address when mode == ZB_APSDE_ADDRMODE_EXT.

7.5.2 ZbApsBufT

Buffer descriptor for vectored/scatter-gather API. Used when ZB_APSDE_DATAREQ_TXOPTIONS_VECTOR TX Options bit is set.

Parameters

- const uint8_t *data
- unsigned int len

7.5.3 ZbApsdeDataConfT

APSDE-DATA.confirm data structure.

Parameters

- struct ZbApsAddrT dst - Destination address information of the APSDE-DATA.request
- uint8_t srcEndpt - Source endpoint of the APSDE-DATA.request
- uint32_t asduHandle - The optional handle Id from the APSDE-DATA.request
- enum ZbStatusCodeT status - Zigbee Status Code

7.5.4 ZbApsdeDataIndT

- APSDE-DATA.indication data structure. These messages are received from the stack after setting up the proper message filter using ZbApsFilterEndpointAdd or ZbApsFilterClusterAdd.
- struct ZbApsAddrT dst - Destination address information of the incoming APS message.
- struct ZbApsAddrT src - Source address information of the incoming APS message.
- uint16_t profileId - The profile for which this frame is intended.
- uint16_t clusterId - The object for which this frame is intended.
- uint8_t *asdu - The set of octets comprising the ASDU that is received.
- uint16_t asduLength - The number of octets comprising the ASDU.
- enum ZbStatusCodeT securityStatus - The security level used to encrypt the incoming frame. One of: ZB_APS_STATUS_UNSECURED (no decryption necessary) ZB_APS_STATUS_SECURED_NWK_KEY (decrypted with the Network Key) ZB_APS_STATUS_SECURED_LINK_KEY (decrypted with a Link Key)
- uint8_t linkQuality - The incoming Link Quality value, or set to 0 if not supported by lower layers.
- int8_t rssi - The incoming RSSI value, or set to PHY_RSSI_INVALID (-128) if not supported by lower layers.
- uint16_t linkAddr - Exegin Addon for Inter-PAN portability.

7.5.5 ZbApsdeDataReqT

APSDE-DATA.request data structure

Parameters

- struct ZbApsAddrT dst - Destination address information, including address mode, address and endpoint.
- uint16_t profileId - The profile for which this frame is intended.
- uint16_t clusterId - The object for which this frame is intended.
- uint16_t srcEndpt - The source endpoint.
- const void *asdu - The set of octets comprising the ASDU to be transfered.
- uint16_t asduLength - The number of octets comprising the ASDU.
- uint32_t asduHandle - An optional handle Id to use in order to match up with the APSDE- DATA.confirm.
- uint16_t txOptions - Transmit options bitmask (e.g. ZB_APSDE_DATAREQ_TXOPTIONS_SECURITY)
- bool discoverRoute - If the user performs route discovery separately using ZbNlmeRouteDiscReq(), then he can set discoverRoute to zero, decreasing the length of time an APS data request may take if there is a problem sending the packet to the target.
- uint8_t radius - Network radius. If 0, default value is used.
- uint16_t aliasAddr - Requires ZB_APSDE_DATAREQ_TXOPTIONS_ALIAS to be enabled.
- uint8_t aliasSeqnum - Requires ZB_APSDE_DATAREQ_TXOPTIONS_ALIAS to be enabled.

7.5.6 ZbApsmeAddKeyConfT

APSME-ADD-KEY.confirm - Exegin custom

Parameters

- enum ZbStatusCodeT status

8 NWK – Network layer services

8.1 NLME

8.1.1 ZbNwkGet

Gets a NWK information base attribute.

Prototype

```
enum ZbStatusCodeT ZbNwkGet(struct ZigBeeT *zb, enum ZbNwkNibAttrIdT attrId,
        void *attrPtr, unsigned int attrSz);
```

Parameters

- zb – Zigbee stack structure
- attrId – attribute ID to get
- attrPtr – (OUT) buffer to store the attribute
- attrSz – size of the attribute

8.1.2 ZbNwkSet

Sets a NWK information base attribute.

Prototype

```
enum ZbStatusCodeT ZbNwkSet(struct ZigBeeT *zb, enum ZbNwkNibAttrIdT attrId,
        void *attrPtr, unsigned int attrSz);
```

Parameters

- zb – Zigbee stack structure
- attrId – attribute ID to set
- attrPtr – pointer to the attribute
- attrSz – size of the attribute

Returns the status.

8.1.3 ZbNlmePermitJoinReq

Implements NLME-PERMIT-JOIN.request.

Prototype

```
void ZbNlmePermitJoinReq(struct ZigBeeT *zb, ZbNlmePermitJoinReqT *permitReq,
        ZbNlmePermitJoinConfT *permitConf);
```

Parameters

- zb – Zigbee stack structure
- permitReq – NLME-PERMIT-JOIN.request
- permitConf – NLME-PERMIT-JOIN.confirm

For more information, refer to section 3.2.2.7 of [R1].

8.1.4 ZbNlmeJoiningPolicyGet

Gets a network's joining policy.

Prototype

```
enum WpanJoinPolicyT ZbNlmeJoiningPolicyGet(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns the join policy as an enumeration.

For more information, refer to Table 2-143 in [R1].

8.1.5 ZbNlmeJoiningPolicyConfigure

Configures the joining policy and IEEE joining list. Must only be called by the coordinator application. The router and end devices applications must not call this, but rather obtain these parameters from the coordinator through ZDP commands.

Prototype

```
bool ZbNlmeJoiningPolicyConfigure(struct ZigBeeT *zb, enum WpanJoinPolicyT policy,
        uint64_t *extAddrList, unsigned int numExtAddr, uint8_t *updateIdOverride);
```

Parameters

- zb – Zigbee stack structure
- policy – joining policy to set
- extAddrList – IEEE List to append (duplicates are ignored)
- numExtAddr – number of IEEE List entries to append
- updateIdOverride – update Id to use rather than incrementing it on policy or IEEE List change. The user may not want to increment the updateId if he is appending IEEE List entries from multiple ZDP responses.

Returns true on success, failure otherwise.

For more information, refer to section 2.4.4.4.11 of [R1].

8.1.6 ZbNlmeIeeeJoiningListClear

Clears the IEEE Joining List.

Prototype

```
void ZbNlmeIeeeJoiningListClear(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

8.1.7 ZbNlmeIeeeJoiningListRemove

Removes an extended address from the IEEE Joining List.

Prototype

```
bool ZbNlmeIeeeJoiningListRemove(struct ZigBeeT *zb, uint64_t extAddr);
```

Parameters

- zb – Zigbee stack structure
- extAddr – extended address to remove

Returns true on success, false otherwise.

8.1.8 ZbNlmeLeaveReq

Implements NLME-LEAVE.request.

Prototype

```
enum ZbStatusCodeT ZbNlmeLeaveReq(struct ZigBeeT *zb, ZbNlmeLeaveReqT *leaveReqPtr,
        void (*callback)(ZbNlmeLeaveConfT *leaveConfPtr, void *arg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure
- leaveReqPtr – NLME-LEAVE.request
- leaveConfPtr – NLME-LEAVE.confirm

Returns the status.

For more information, refer to section 3.2.2.18 of [R1].

8.1.9 ZbNlmeRouteDiscReq

Implements NLME-ROUTE-DISCOVERY.request.

Prototype

```
enum ZbStatusCodeT ZbNlmeRouteDiscReq(struct ZigBeeT *zb, ZbNlmeRouteDiscReqT
*routeDiscReqPtr,
        void (*callback)(ZbNlmeRouteDiscConfT *discConf, void *cbarg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- routeDiscReqPtr – NLME-ROUTE-DISCOVERY.request
- routeDiscConfPtr – NLME-ROUTE-DISCOVERY.confirm

Returns the status.

For more information, refer to section 3.2.2.33 of [R1].

8.1.10 ZbNlmeSetInterface

Implements NLME-SET-INTERFACE.request.

Prototype

```
void ZbNlmeSetInterface(struct ZigBeeT *zb, ZbNlmeSetInterfaceReqT *req,
        ZbNlmeSetInterfaceConfT *conf);
```

Parameters

- zb – Zigbee stack structure
- req – NLME-SET-INTERFACE.request
- conf – NLME-SET-INTERFACE.confirm

For more information, refer to section 3.2.2.35 of [R1].

8.1.11 ZbNlmeGetInterface

Implements NLME-SET-INTERFACE.request.

Prototype

```
void ZbNlmeGetInterface(struct ZigBeeT *zb, ZbNlmeGetInterfaceReqT *req,
        ZbNlmeGetInterfaceConfT *conf);
```

Parameters

- zb – Zigbee stack structure
- req – NLME-GET-INTERFACE.request
- conf – NLME-GET-INTERFACE.confirm

For more information, refer to section 3.2.2.37 of [R1].

8.1.12 ZbNlmeGetReq

Performs NLME-GET request to read NWK IB attributes.

Prototype

```
void ZbNlmeGetReq(struct ZigBeeT *zb, ZbNlmeGetReqT *getReqPtr, ZbNlmeGetConfT *getConfPtr);
```

Parameters

- zb – Zigbee stack structure
- getReqPtr – NLME-GET.request
- getConfPtr – NLME-GET.confirm

8.1.13 ZbNlmeSetReq

Performs NLME-SET request to write NWK IB attributes.

Prototype

```
void ZbNlmeSetReq(struct ZigBeeT *zb, ZbNlmeSetReqT *setReqPtr, ZbNlmeSetConfT *setConfPtr);
```

Parameters

- zb – Zigbee stack structure
- setReqPtr – NLME-SET.request
- setConfPtr – NLME-SET.confirm

8.1.14 ZbNlmeResetReq

Performs an NLME-RESET.request/confirm to reset the NWK attributes to defaults.

Prototype

```
void ZbNlmeResetReq(struct ZigBeeT *zb, ZbNlmeResetReqT *resetReqPtr, ZbNlmeResetConfT
*resetConfPtr);
```

Parameters

- zb – Zigbee stack structure
- resetReqPtr – NLME-RESET.request struct
- resetConfPtr – NLME-RESET.confirm struct

8.1.15 ZbNlmeSyncReq

Performs an NLME-SYNC.request to poll the device's parent for data. The NLME-SYNC.confirm is received via the callback provided. Used by a sleepy device.

Prototype

```
enum ZbStatusCodeT ZbNlmeSyncReq(struct ZigBeeT *zb, ZbNlmeSyncReqT *syncReqPtr,
        void (*callback)(ZbNlmeSyncConfT *syncConfPtr, void *arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- syncReqPtr – NLME-SYNC.request

Returns the status code.

8.1.16 ZbNlmeNetDiscReq

NLME-NETWORK-DISCOVERY.request to perform a scan of the device's personal operating space to discover networks.

Prototype

```
enum ZbStatusCodeT ZbNlmeNetDiscReq(struct ZigBeeT *zb, ZbNlmeNetDiscReqT *req,
        void (*callback)(ZbNlmeNetDiscConfT *conf, void *cbarg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure
- req – NLME-NETWORK-DISCOVERY.request struct
- callback – this callback is called to deliver the NLME-NETWORK-DISCOVERY.confirm

Returns the status code.

8.1.17 ZbNlmeNetFormReq

NLME-NETWORK-FORMATION.request for the device starts a new Zigbee network with itself as the coordinator and subsequently makes changes to its superframe configuration.

Prototype

```
enum ZbStatusCodeT ZbNlmeNetFormReq(struct ZigBeeT *zb, ZbNlmeNetFormReqT *req,
       void (*callback)(ZbNlmeNetFormConfT *formConf, void *arg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure

- req – NLME-NETWORK-FORMATION.request struct

Returns the status code.

8.1.18 ZbNlmeJoinReq

NLME-JOIN.request to join or rejoin a network, or to change the operating channel for the device while within an operating network.

Prototype

```
enum ZbStatusCodeT ZbNlmeJoinReq(struct ZigBeeT *zb, ZbNlmeJoinReqT *joinReqPtr,
        void (*callback)(ZbNlmeJoinConfT *joinConf, void *arg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure

Returns status code.

8.1.19 ZbNlmeDirectJoinReq

NLME-DIRECT-JOIN.request to directly join another device to its network.

Prototype

```
void ZbNlmeDirectJoinReq(struct ZigBeeT *zb, ZbNlmeDirectJoinReqT *directJoinReqPtr,
ZbNlmeDirectJoinConfT
        *directJoinConfPtr);
```

Parameters

- zb – Zigbee stack structure
- directJoinReqPtr – NLME-DIRECT-JOIN.request
- directJoinConfPtr – NLME-DIRECT-JOIN.confirm

8.1.20 ZbNlmeStartRouterReq

NLME-START-ROUTER.request initiates the activities expected of a Zigbee router, including the routing of data frames, route discovery, and the accepting of requests to join the network from other devices.

Prototype

```
void ZbNlmeStartRouterReq(struct ZigBeeT *zb, ZbNlmeStartRouterReqT *req,
ZbNlmeStartRouterConfT *conf);
```

Parameters

- zb – Zigbee stack structure
- req – NLME-START-ROUTER.request
- conf – NLME-START-ROUTER.confirm

8.1.21 ZbNlmeEdScanReq

NLME-ED-SCAN.request initiates an energy scan to evaluate channels in the local area.

Prototype

```
enum ZbStatusCodeT ZbNlmeEdScanReq(struct ZigBeeT *zb, ZbNlmeEdScanReqT *req,
        void (*callback)(ZbNlmeEdScanConfT *scanConf, void *arg), void *cbarg);
```

Parameters

- zb – Zigbee stack structure
- req – NLME-ED-SCAN.request

Returns the status code.

8.2 NLDE

8.2.1 ZbNldeDataReqCallback

Performs the NLDE-DATA.request. The NLDE-DATA.confirm is returned asynchronously through the callback function provided.

Prototype

```
enum ZbStatusCodeT ZbNldeDataReqCallback(struct ZigBeeT *zb, ZbNldeDataReqT *req,
        void (*callback)(ZbNldeDataConfT *dataConf, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – NLDE-DATA.request
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 3.2.1.1 of [R1].

8.3 Utility

8.3.1 ZbNwkSendEdkaReq

Starts the Zigbee end-device timeout keepalive.

Prototype

```
bool ZbNwkSendEdkaReq(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns true on success, false otherwise.

For more information, refer to section 3.6.10.3 of [R1].

8.3.2 ZbNwkFastPollRequest

Requests the fast polling at the network layer.

Prototype

```
bool ZbNwkFastPollRequest(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns true on success, false otherwise.

8.3.3 ZbNwkFastPollRelease

Releases the fast polling from the network layer.

Prototype

```
void ZbNwkFastPollRelease(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

8.3.4 ZbNwkFastPollResourceCount

Retrieves the fast poll resource count.

Prototype

```
unsigned int ZbNwkFastPollResourceCount(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns the fast poll resource count.

8.3.5 ZbNwkGetParentExtAddr

Gets the parent's extended address.

Prototype

```
uint64_t ZbNwkGetParentExtAddr(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns the parent's extended address, or 0 if not found.

8.3.6 ZbNwkGetParentShortAddr

Gets the parent's short address.

Prototype

```
uint16_t ZbNwkGetParentShortAddr(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns the parent's short address, or ZB_NWK_ADDR_UNDEFINED if not found.

8.3.7 ZbNwkIeeeJoiningListEnabled

Checks if the IEEE joining list is enabled.

Prototype

```
bool ZbNwkIeeeJoiningListEnabled(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns true if IEEE joining list is enabled, false otherwise.

8.3.8 ZbNwkIfGetTxPower

Gets the TX power on an interface.

Prototype

```
bool ZbNwkIfGetTxPower(struct ZigBeeT *zb, const char *name, int8_t *tx_power)
```

Parameters

- zb – Zigbee stack structure
- name – interface name
- tx_power – (OUT) TX power

Returns true on success, false otherwise.

8.3.9 ZbNwkIfSetTxPower

Sets the TX power on an interface.

Prototype

```
bool ZbNwkIfSetTxPower(struct ZigBeeT *zb, const char *name, int8_t tx_power);
```

Parameters

- zb – Zigbee stack structure
- name – interface name
- tx_power – TX power

Returns true on success, false otherwise.

8.3.10 ZbNwkIfToggleByName

Toggles an interface on and off.

Prototype

```
bool ZbNwkIfToggleByName(struct ZigBeeT *zb, const char *name, bool enable);
```

Parameters

- zb – Zigbee stack structure
- name – interface name
- enable – true to enable, false to disable

Returns true on success, false otherwise.

8.3.11 ZbNwkGetSecMaterial

Gets a network security material set for a given sequence number.

Prototype

```
bool ZbNwkGetSecMaterial(struct ZigBeeT *zb, uint8_t keySeqno, ZbNwkSecMaterialT *material);
```

Parameters

- zb – Zigbee stack structure
- keySeqno – network key sequence number
- material – security material

Returns true on success, false otherwise.

For more information, refer to section 4.2.1 of [R1].

8.3.12 ZbNwkSetFrameCounter

Inserts or updates a frame counter in a given material set. If srcAddr is 0, or our local extended address then this call updates the outgoing frame counter. Otherwise this call updates (or creates) an incoming frame counter.

This function compares the new frame counter against the existing one (if found). If the new value of the frame counter is stale then the update is rejected and this function returns false. Otherwise, the update always succeeds (with a possible LRU eviction if the table is full).

However, due to some known buggy devices, frame counter resets have been observed out in the wild. Therefore, stale frame counters are accepted into the table subject to rate limiting and cooldown policies (controlled by the nwkFrameCounterCooldown NIB attribute).

Prototype

```
bool ZbNwkSetFrameCounter(struct ZigBeeT *zb, uint8_t keySeqno, uint64_t srcAddr,
        uint32_t newFrameCount);
```

Parameters

- zb – Zigbee stack structure
- keySeqno – network key sequence number
- srcAddr – source extended address
- newFrameCount – new frame counter value

Returns true on success, false otherwise.

8.3.13 ZbNwkAddrStoreMap

The attempts to store an address pair in the neighbor table or address map. The address map is stored in only one of the tables, with priority given to the neighbor table.

This function also checks the address pair for conflicts. If an address conflict is found, this function returns false. In all other cases, true is returned.

Prototype

```
bool ZbNwkAddrStoreMap(struct ZigBeeT *zb, uint16_t nwkAddr, uint64_t extAddr,
       bool resolve_conflict);
```

Parameters

- zb – Zigbee stack structure
- nwkAddr – network address
- extAddr – extended address
- resolve_conflict – true if a potential address conflict must be resolved, false otherwise

Returns true on success, false otherwise.

For more information, refer to table 3-60 [R1].

8.3.14 ZbNwkAddrLookupNwk

Retrieves the network address that corresponds to the extended address.

Prototype

```
uint16_t ZbNwkAddrLookupNwk(struct ZigBeeT *zb, uint64_t extAddr);
```

Parameters

- zb – Zigbee stack structure
- extAddr – extended address

Returns the network address that corresponds to the extended address if known, otherwise returns ZB_NWK_ADDR_UNDEFINED.

8.3.15 ZbNwkAddrLookupExt

Retrieves the extended address that corresponds to the network address if known.

Prototype

```
uint64_t ZbNwkAddrLookupExt(struct ZigBeeT *zb, uint16_t nwkAddr);
```

Parameters

- zb – Zigbee stack structure
- nwkAddr – network address

Returns the extended address that corresponds to the network address if known, otherwise returns 0.

8.3.16 ZbNwkToggleDutyCycle

Toggles duty cycle on and off.

Prototype

```
bool ZbNwkToggleDutyCycle(struct ZigBeeT *zb, bool enable);
```

Parameters

- zb – Zigbee stack structure
- enable – true to enable, false to disable

Returns true on success, false otherwise.

8.3.17 ZbNwkNeighborClearAll

Clears all entries of neighbor table, expect and/or children if specified.

Prototype

```
void ZbNwkNeighborClearAll(struct ZigBeeT *zb, bool keep_parent, bool keep_children);
```

Parameters

- zb – Zigbee stack structure
- keep_parent – true to keep parent and only reset its age, false otherwise
- keep_children – true to keep children, false otherwise

For more information, refer to section 3.6.1.5 in [R1].

8.3.18 ZbNwkCommissioningConfig

Configures the MAC interface to be able to send and receive commissioning InterPAN packets. This function must be called before the stack has been started (that is Zbstartup).

```
bool ZbNwkCommissioningConfig(struct ZigBeeT *zb, struct ZbNwkCommissioningInfo
*commission_info);
```

Parameters

- zb – Zigbee stack structure
- commission_info – Pointer to commissioning data structure

Returns true if configuration is successfully applied, false otherwise.

8.3.19 ZbExtendedAddress

Retrieves the extended address the zigbee stack is using.

Prototype

```
uint64_t ZbExtendedAddress(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns extended address.

8.3.20 ZbShortAddress

Retrieves the network short address the Zigbee stack is using.

Prototype

```
uint16_t ZbShortAddress(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns network short address.

9 ZDO – Zigbee device object

9.1 Device discovery

9.1.1 ZbZdoNwkAddrReq

Gets a device's 16-bit short address if given an extended address. Note that this causes a broadcast message.

Prototype

```
enum ZbStatusCodeT ZbZdoNwkAddrReq(struct ZigBeeT *zb, ZbZdoNwkAddrReqT *req, void
(*callback)(ZbZdoNwkAddrRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – NWK_Addr_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to table 3-60 in [R1].

9.1.2 ZbZdoIeeeAddrReq

Gets a device's 64-bit extended address if given a short network address.

Prototype

```
enum ZbStatusCodeT ZbZdoIeeeAddrReq(struct ZigBeeT *zb, ZbZdoIeeeAddrReqT *req, void
(*callback)(ZbZdoIeeeAddrRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – IEEE_Addr_req
- callback – function to call on completion
- arg – callback argument

For more information, refer to table 3-60 in [R1].

9.1.3 ZbZdoNodeDescReq

Retrieves a device's node descriptor.

Prototype

```
enum ZbStatusCodeT ZbZdoNodeDescReq(struct ZigBeeT *zb, ZbZdoNodeDescReqT *req, void
(*callback)(ZbZdoNodeDescRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Node_desc_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.1.3 of [R1].

9.1.4 ZbZdoPowerDescReq

Retrieves a device's power descriptor.

Prototype

```
enum ZbStatusCodeT ZbZdoPowerDescReq(struct ZigBeeT *zb, ZbZdoPowerDescReqT *req, void
(*callback)(ZbZdoPowerDescRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Power_desc_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.1.4 of [R1].

9.1.5 ZbZdoSimpleDescReq

Retrieves a device's simple descriptor.

Prototype

```
enum ZbStatusCodeT ZbZdoSimpleDescReq(struct ZigBeeT *zb, ZbZdoSimpleDescReqT *req, void
(*callback)(ZbZdoSimpleDescRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Simple_desc_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.1.5 of [R1].

9.1.6 ZbZdoSetComplexDesc

Configures the local complex descriptor. The information from the structPtr parameter is copied to the stack instance.

Prototype

```
enum ZbStatusCodeT ZB_WARN_UNUSED ZbZdoSetComplexDesc(struct ZigBeeT *zb, ZbZdoComplexDescT
*structPtr);
```

Parameters

- zb – Zigbee stack structure
- structPtr – Pointer to complex descriptor structure to copy

Returns the status.

9.1.7 ZbZdoActiveEpReq

Retrieves a device's active endpoint list.

Prototype

```
enum ZbStatusCodeT ZbZdoActiveEpReq(struct ZigBeeT *zb, ZbZdoActiveEpReqT *req, void
(*callback)(ZbZdoActiveEpRspT *rsp, void *cb_arg),
```

Parameters

- zb – Zigbee stack structure
- req – Active_ep_req
- callback – function to call on completion
- arg – callback argument

For more information, refer to section 2.4.3.1.6 of [R1].

9.1.8 ZbZdoMatchDescReq

Retrieves a device's match descriptor.

Prototype

```
enum ZbStatusCodeT ZbZdoMatchDescReq(struct ZigBeeT *zb, ZbZdoMatchDescReqT *req, void
(*callback)(ZbZdoMatchDescRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Match_desc_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.1.7 of [R1].

9.1.9 ZbZdoMatchDescMulti

Sends a ZDO Match-Desc request and receive multiple responses. The responses, if any, are received by the callback. Each callback contains the result of a callback from a single device.

The callback is called each time a response is received. After a timeout period, the callback is called with a status of ZB_ZDP_STATUS_TIMEOUT to indicate the internal response filter is removed and any additional responses for this request are not processed.

Prototype

```
enum ZbStatusCodeT ZbZdoMatchDescMulti(struct ZigBeeT *zb, ZbZdoMatchDescReqT *req, void
(*callback)(ZbZdoMatchDescRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Match_desc_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

9.1.10 ZbZdoDeviceAnnce

Sends a Device_annce message to the network.

Prototype

```
void ZbZdoDeviceAnnce(struct ZigBeeT *zb, ZbZdoDeviceAnnceT *deviceAnncePtr);
```

Parameters

- zb – Zigbee stack structure
- deviceAnncePtr – pointer to Device_annce structure

For more information, refer to section 2.4.3.1.11 of [R1].

9.1.11 ZbZdoDeviceAnnceAlias

Sends a Device_annce message to the network using aliasing.

Used primarily for Green Power.

Prototype

```
void ZbZdoDeviceAnnceAlias(struct ZigBeeT *zb, ZbZdoDeviceAnnceT *deviceAnncePtr);
```

Parameters

- zb – Zigbee stack structure
- deviceAnncePtr – pointer to Device_annce structure

For more information on Device_annce, refer to section 2.4.3.1.11 of [R1].

9.1.12 ZbZdoDeviceAnnceFilterRegister

Registers a filter in the ZDO for the application to receive Device_Annce messages.
Prototype

```
struct ZbZdoDeviceAnnceFilterT * ZbZdoDeviceAnnceFilterRegister(struct ZigBeeT *zb,
        void (*callback)(struct ZigBeeT *zb, ZbZdoDeviceAnnceT *annce, uint8_t seqno));
```

Parameters

- zb – Zigbee stack structure
- callback - callback function to call with Device_Annce messages

Returns pointer to filter structure.

9.1.13 ZbZdoDeviceAnnceFilterRemove

Removes a Device_annce filter description.
Prototype

```
void ZbZdoDeviceAnnceFilterRemove(struct ZigBeeT *zb,
       struct ZbZdoDeviceAnnceFilterT *handler);
```

Parameters

- zb Zigbee stack structure
- handler Device_annce filter handler

For more information on Device_annce, refer to section 2.4.3.1.11 of [R1].

9.1.14 ZbZdoExtSimpleDescReq

Retrieves a device's extended simple descriptor.
Prototype

```
void ZbZdoExtSimpleDescReq(struct ZigBeeT *zb, ZbZdoExtSimpleDescReqT *req,
       void (*callback)(ZbZdoExtSimpleDescRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Ext_simple_desc_req
- callback – function to call on completion
- arg – callback argument

For more information, refer to section 2.4.3.1.22 of [R1].

9.1.15 ZbZdoGetNextSeqNum

Returns and increments the next ZDO sequence number
Prototype

```
uint8_t ZbZdoGetNextSeqNum(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns the ZDO sequence number.

9.2 Binding

9.2.1 ZbZdoBindReq

Performs a ZDO bind operation.

Prototype

```
enum ZbStatusCodeT ZbZdoBindReq(struct ZigBeeT *zb, ZbZdoBindReqT *req, void (*callback)
(ZbZdoBindRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Bind_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.2.2 of [R1].

9.2.2 ZbZdoUnbindReq

Performs a ZDP unbind bind operation.

Prototype

```
enum ZbStatusCodeT ZbZdoUnbindReq(struct ZigBeeT *zb, ZbZdoBindReqT *req, void (*callback)
(ZbZdoBindRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Unbind_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.2.3 of [R1].

9.3 Network management

9.3.1 ZbZdoLqiReq

Performs a Mgmt_Lqi_req command.

Prototype

```
enum ZbStatusCodeT ZbZdoLqiReq(struct ZigBeeT *zb, ZbZdoLqiReqT *req, void (*callback)
(ZbZdoLqiRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Mgmt_Lqi_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.3.2 of [R1].

9.3.2 ZbZdoRtgReq

Performs a Mgmt_Rtg_req command.

Prototype

```
enum ZbStatusCodeT ZbZdoRtgReq(struct ZigBeeT *zb, ZbZdoRtgReqT *req, void (*callback)
(ZbZdoRtgRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure

- req – Mgmt_Rtg_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.3.3 of [R1].

9.3.3 ZbZdoMgmtBindReq

Performs a Mgmt_Bind_req command.

Prototype

```
enum ZbStatusCodeT ZbZdoMgmtBindReq(struct ZigBeeT *zb, ZbZdoMgmtBindReqT *req, void (*callback)
(ZbZdoMgmtBindRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Mgmt_bind_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.3.4 of [R1].

9.3.4 ZbZdoPermitJoinReq

Performs a Mgmt_permit_joining_req command.

Prototype

```
enum ZbStatusCodeT ZbZdoPermitJoinReq(struct ZigBeeT *zb, ZbZdoPermitJoinReqT *req,
        void (*callback)(ZbZdoPermitJoinRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Mgmt_permit_joining_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.3.7 of [R1].

9.3.5 ZbZdoNwkUpdateReq

Performs a Mgmt_Nwk_update_req command.

Prototype

```
enum ZbStatusCodeT ZbZdoNwkUpdateReq(struct ZigBeeT *zb, ZbZdoNwkUpdateReqT *req,
        void (*callback)(ZbZdoNwkUpdateNotifyT *reqPtr, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Mgmt_Nwk_update_req
- callback – function to call on completion
- arg – callback argument

For more information, refer to section 2.4.3.3.9 of [R1].

9.3.6 ZbZdoNwkEnhUpdateReq

Performs a Mgmt_Nwk_Enhanced_Update_req command.

Prototype

```
enum ZbStatusCodeT ZbZdoNwkEnhUpdateReq(struct ZigBeeT *zb, struct ZbZdoNwkEnhUpdateReqT
*req,
        void (*callback)(ZbZdoNwkUpdateNotifyT *reqPtr, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Mgmt_Nwk_Enhanced_Update_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.3.10 of [R1].

9.3.7 ZbZdoNwkUpdateNotify

Sends an unsolicited Mgmt_Nwk_Update_notify command to the network manager. Does not wait for any response.

Prototype

```
enum ZbStatusCodeT ZbZdoNwkUpdateNotify(struct ZigBeeT *zb, ZbZdoNwkUpdateNotifyT *reqPtr);
```

Parameters

- zb – Zigbee stack structure
- reqPtr – Mgmt_Nwk_Update_notify

Returns the status.

For more information, refer to section 2.4.4.4.12 of [R1].

9.3.8 ZbZdoLeaveReq

Performs a Mgmt_leave_req command.

Prototype

```
enum ZbStatusCodeT ZbZdoLeaveReq(struct ZigBeeT *zb, ZbZdoLeaveReqT *req,
        void (*callback)(ZbZdoLeaveRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – Mgmt_Leave_req
- callback – function to call on completion
- arg – callback argument

Returns the status.

For more information, refer to section 2.4.3.3.5 of [R1].

9.3.9 ZbZdoNwkIeeeJoinListRsp

Sends an IEEE-Joining-List.response.

Prototype

```
enum ZbStatusCodeT ZbZdoNwkIeeeJoinListReq(struct ZigBeeT *zb, struct
ZbZdoNwkIeeeJoinListReqT *req,
        void (*callback)(struct ZbZdoNwkIeeeJoinListRspT *rsp, void *cb_arg), void *arg);
        unsigned int ZbZdoNwkIeeeJoinListRsp(struct ZigBeeT *zb, uint16_t dstNwkAddr,
        uint8_t startIndex, uint8_t seqnum, bool fromRequest);
```

Parameters

- zb – Zigbee stack structure
- dstNwkAddr – destination network address

Returns the number of entries sent in response.

9.3.10 ZbZdoNwkIeeeJoinListBcastAll

Sends an IEEE-Joining-List.response broadcast message for all entries in the IEEE join list.

Prototype

```
unsigned int ZbZdoNwkIeeeJoinListBcastAll(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns the number of entries sent.

For more information, refer to section 2.4.4.4.11 of [R1].

10 ZCL – Zigbee cluster library

10.1 Endpoint Initialization

10.1.1 ZbZclAddEndpoint

Adds an endpoint with a basic server cluster allocated on it. Normally this is invoked at startup, however with ZSDK it is possible to add an endpoint at any time.

Prototype

```
void ZbZclAddEndpoint(struct ZigBeeT *zb, ZbApsmeAddEndpointReqT *addReqPtr,
        ZbApsmeAddEndpointConfT *addConfPtr);
```

Parameters

- zb – Zigbee stack structure
  - addReqPtr – addEndpoint request
  - addConfPtr – addEndpoint confirm

10.1.2 ZbZclAddEndpointNoBasic

Adds an endpoint without a basic server cluster allocated on it. Normally this is invoked at startup, however with ZSDK it is possible to add an endpoint at any time.

Prototype

```
void ZbZclAddEndpointNoBasic(struct ZigBeeT *zb, ZbApsmeAddEndpointReqT *addReqPtr,
        ZbApsmeAddEndpointConfT *addConfPtr);
```

Parameters

- zb – Zigbee stack structure
- addReqPtr – addEndpoint request
- addConfPtr – addEndpoint confirm

10.1.3 ZbZclRemoveEndpoint

Unregisters and cleans up an endpoint from the APS endpoint table. Typically, this is called when terminating an app. Also frees the basic cluster running on this endpoint.

The application is responsible for removing (and freeing) other clusters allocated for this endpoint.

Prototype

```
void ZbZclRemoveEndpoint(struct ZigBeeT *zb, ZbApsmeRemoveEndpointReqT *req,
        ZbApsmeRemoveEndpointConfT *conf);
```

Parameters

- zb – Zigbee stack structure
- addReqPtr – removeEndpoint request
- addConfPtr – removeEndpoint confirm

10.2 Cluster initialization

10.2.1 ZbZclClusterAlloc

Allocates and initializes the cluster prior to use with the settings required for Zigbee 3. If different settings are needed (for example a different Profile ID) these can be reset later using convenience functions.

Prototype

```
struct ZbZclClusterT * ZbZclClusterAlloc(struct ZigBeeT *zb, unsigned int alloc_sz, enum
ZbZclClusterIdT cluster_id, uint8_t endpoint, enum ZbZclDirectionT direction);
```

Parameters

- zb – Zigbee stack structure
- alloc_sz – size of memory allocated. Must be at least size of (struct ZbZclClusterT). Can be larger to include any cluster specific data.

10.2.2 ZbZclClusterInitCommandReq

Initializes a ZbZclCommandReqT command request struct with parameters from the cluster (for example: profile ID, cluster ID, source endpoint).

Prototype

```
void ZbZclClusterInitCommandReq(struct ZbZclClusterT *clusterPtr, ZbZclCommandReqT *cmdReq);
```

Parameters

- clusterPtr – ZCL cluster structure
- cmdReq – command request

10.2.3 ZbZclClusterInitApsdeReq

Initializes a ZbApsdeDataReqT APS data request struct with parameters from the cluster (for example: profile ID, cluster ID, source endpoint).

Prototype

```
void ZbZclClusterInitApsdeReq(struct ZbZclClusterT *clusterPtr, ZbApsdeDataReqT *apsReq,
ZbApsdeDataIndT *dataIndPtr);
```

Parameters

- clusterPtr – ZCL cluster structure
- dataIndPtr – source of originating command.

10.2.4 ZbZclClusterFree

Cleanup function to free an allocated cluster descriptor.

Prototype

```
void ZbZclClusterFree(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

10.2.5 ZbZclClusterEndpointRegister

Adds a cluster to its endpoint’s cluster list.

Prototype

```
bool ZbZclClusterEndpointRegister(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL Cluster structure

Returns true on success, failure otherwise.

10.2.6 ZbZclClusterEndpointRemove

Removes a cluster from its endpoint’s cluster list.

Prototype

```
bool ZbZclClusterEndpointRemove(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL Cluster structure

Returns true on success, failure otherwise.

10.2.7 ZbZclClusterBind

Binds the cluster structure to a particular endpoint, optional remote address, and sets up a packet filter to handle command reception.

Prototype

```
enum ZclStatusCodeT ZbZclClusterBind(struct ZbZclClusterT *clusterPtr,
        uint8_t endpoint, uint16_t profileId, enum ZbZclDirectionT direction);
```

Parameters

- clusterPtr – ZCL cluster structure
- profileId – profile ID of clusterId
- direction – direction of cluster

Returns the ZCL Status Code.

10.3 Cluster configuration

10.3.1 ZbZclClusterSetCallbackArg

Configures the application argument passed through the various callbacks to the application (such as custom attribute read/write).

Prototype

```
void ZbZclClusterSetCallbackArg(struct ZbZclClusterT *clusterPtr, void *app_cb_arg);
```

Parameters

- clusterPtr – ZCL cluster structure
- app_cb_arg – application callback argument

10.3.2 ZbZclClusterSetMfrCode

Sets the cluster manufacturer code.

Prototype

```
void ZbZclClusterSetMfrCode(struct ZbZclClusterT *clusterPtr, uint16_t mfrCode);
```

Parameters

- clusterPtr – ZCL cluster structure
- mfrCode – manufacturer code

For more information, refer to section 2.3.3 of [R3].

10.3.3 ZbZclClusterSetProfileId

Sets the cluster profile ID.

Prototype

```
void ZbZclClusterSetProfileId(struct ZbZclClusterT *clusterPtr, uint16_t profileId);
```

Parameters

- clusterPtr – ZCL cluster structure
- profileId – profile ID

For more information, refer to section 2.6.1.1 of [R3].

10.3.4 ZbZclClusterSetMinSecurity

Sets the cluster minimum security.

Prototype

```
bool ZbZclClusterSetMinSecurity(struct ZbZclClusterT *clusterPtr,
        enum ZbStatusCodeT minSecurity);
```

Parameters

- clusterPtr – ZCL cluster structure
- minSecurity – minimum security
  - ZB_APS_STATUS_UNSECURED – unsecured allowed (lowest level)
  - ZB_APS_STATUS_SECURED_NWK_KEY – must be at least NWK secured
  - ZB_APS_STATUS_SECURED_LINK_KEY – must be APS secured (highest level)

Returns true on success, false otherwise.

10.3.5 ZbZclClusterSetTxOptions

Sets the cluster TxOptions setting.

Prototype

```
void ZbZclClusterSetTxOptions(struct ZbZclClusterT *clusterPtr, uint16_t txOptions);
```

Parameters

- clusterPtr – ZCL cluster structure
- txOptions – TxOptions setting

10.3.6 ZbZclClusterSetDiscoverRoute

Sets the cluster route discovery setting.

Prototype

```
void ZbZclClusterSetDiscoverRoute(struct ZbZclClusterT *clusterPtr,
        bool discoverRoute);
```

Parameters

- clusterPtr – ZCL cluster structure
- discoverRoute – route discovery setting

10.3.7 ZbZclClusterSetRadius

Sets the cluster radius.

Note: the default value of 0 means use the stack default which is correct for most applications.

Prototype

```
void ZbZclClusterSetRadius(struct ZbZclClusterT *clusterPtr, uint8_t radius);
```

Parameters

- clusterPtr – ZCL cluster structure
- radius – radius

10.3.8 ZbZclClusterSetMaxAsduLength

Sets the cluster max ASDU length.

Warning: be careful changing this parameter, the default is correct for most applications.

Prototype

```
bool ZbZclClusterSetMaxAsduLength(struct ZbZclClusterT *clusterPtr,
        uint16_t maxAsduLength);
```

Parameters

- clusterPtr – ZCL cluster structure
- maxAsduLength – max ASDU length

Returns true on success, false otherwise.

10.3.9 ZbZclClusterGetEndpoint

Gets the cluster's endpoint.

Prototype

```
uint8_t ZbZclClusterGetEndpoint(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the endpoint.

10.3.10 ZbZclClusterGetClusterId

Gets the cluster's cluster ID.

Prototype

```
enum ZbZclClusterIdT ZbZclClusterGetClusterId(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the cluster ID.

10.3.11 ZbZclClusterGetProfileId

Gets the cluster's profile ID.

Prototype

```
uint16_t ZbZclClusterGetProfileId(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the profile ID.

10.3.12 ZbZclClusterGetTxOptions

Gets the cluster's TxOptions.

Prototype

```
uint16_t ZbZclClusterGetTxOptions(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the TxOptions.

10.3.13 ZbZclClusterGetDirection

Gets the cluster's direction.

Prototype

```
eenum ZbZclDirectionT ZbZclClusterGetDirection(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the direction.

10.3.14 ZbZclClusterGetDirectionStr

Gets the cluster's direction as a string.

Prototype

```
const char * ZbZclClusterGetDirectionStr(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the direction as a string.

10.3.15 ZbZclClusterGetRadius

Gets the cluster's radius.

Prototype

```
uint8_t ZbZclClusterGetRadius(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the radius.

10.3.16 ZbZclClusterGetMaxAsduLength

Gets the cluster's max ASDU length.

Prototype

```
unsigned int ZbZclClusterGetMaxAsduLength(struct ZbZclClusterT *clusterPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

Returns the max ASDU length.

10.4 Cluster attributes

10.4.1 ZbZclAttrIntegerRead

Reads an attribute of integer type.

Prototype

```
long long ZbZclAttrIntegerRead(struct ZbZclClusterT *clusterPtr,
        uint16_t attributeId, enum ZclDataTypeT *typePtr,
        enum ZclStatusCodeT *statusPtr);
```

Parameters

- clusterPtr – ZCL cluster structure
- attributeId – ZCL attribute Identifier
- typePtr – (OUT) attribute data type
- statusPtr – (OUT) ZCL Status Code

Returns the attribute value.

For more information, refer to section 2.3.5 of [R3].

10.4.2 ZbZclAttrIntegerWrite

Writes an attribute of integer type.

Prototype

```
enum ZclStatusCodeT ZbZclAttrIntegerWrite(struct ZbZclClusterT *clusterPtr,
        uint16_t attributeId, long long value);
```

Parameters

- clusterPtr – ZCL cluster structure
- attributeId – ZCL Attribute Identifier

- value – integer value (as unsigned 64-bit)

Returns the ZCL status code.

For more information, refer to section 2.3.5 of [R3].

10.4.3 ZbZclAttrIntegerIncrement

Increments an attribute of integer type by given value.

Prototype

```
enum ZclStatusCodeT ZbZclAttrIntegerIncrement(struct ZbZclClusterT *clusterPtr,
        uint16_t attributeId, long long value);
```

Parameters

- clusterPtr – ZCL cluster structure
- attributeId – ZCL Attribute Identifier
- value – integer value (as signed 64-bit)

Returns the ZCL status code.

10.4.4 ZbZclAttrEuiRead

Reads an attribute of EUI type.

Prototype

```
uint64_t ZbZclAttrEuiRead(struct ZbZclClusterT *clusterPtr, uint16_t attributeId,
        enum ZclStatusCodeT *statusPtr);
```

Parameters

- clusterPtr – ZCL cluster structure
- attributeId – ZCL Attribute Identifier
- statusPtr – (OUT) ZCL Status Code

Returns the attribute value.

For more information, refer to section 2.3.5 of [R3].

10.4.5 ZbZclAttrEuiWrite

Writes an attribute of EUI type.

Prototype

```
enum ZclStatusCodeT ZbZclAttrEuiWrite(struct ZbZclClusterT *clusterPtr,
        uint16_t attributeId, uint64_t eui);
```

Parameters

- clusterPtr – ZCL cluster structure
- attributeId – ZCL Attribute Identifier
- eui – EUI value (as unsigned 64-bit)

Returns the ZCL status code.

For more information, refer to section 2.3.5 of [R3].

10.4.6 ZbZclAttrStringWrite

Writes an attribute of string type.

Prototype

```
enum ZclStatusCodeT ZbZclAttrStringWrite(struct ZbZclClusterT *clusterPtr,
        uint16_t attributeId, const char *strPtr);
```

Parameters

- clusterPtr – ZCL cluster structure

- attributeId – ZCL attribute Identifier
- strPtr – string pointer (NULL terminated ASCII/UTF-8)

Returns the ZCL status code.

For more information, refer to section 2.3.5 of [R3].

10.4.7 ZbZclReadReq

Performs a ZCL read for an attribute over the network and returns the response via a callback.

Prototype

```
void ZbZclReadReq(struct ZbZclClusterT *clusterPtr, ZbZclReadReqT *req, void (*callback)
(const ZbZclReadRspT *readRsp, void *cb_arg), void *arg);
```

Parameters

clusterPtr – ZCL cluster structure

- req – attribute read request
- callback – function to call on completion
- arg – callback argument

For more information, refer to section 2.5.1 of [R3].

10.4.8 ZbZclDiscoverAttrReq

Discovers an attribute over the network and returns the response via a callback.

Prototype

```
void ZbZclDiscoverAttrReq(struct ZbZclClusterT *clusterPtr,
        ZbZclDiscoverAttrReqT *req,
        void (*callback)(const ZbZclDiscoverAttrRspT *rsp, void *cb_arg), void *arg);
```

Parameters

- clusterPtr – ZCL cluster structure
- req – discover attribute request
- callback – function to call on completion
- arg – callback argument

For more information, refer to section 2.5.13 of [R3].

10.5 Cluster commands

10.5.1 ZbZclCommandReq

Issues a ZCL command, with the response returned via a callback function.

Prototype

```
void ZbZclCommandReq(struct ZigBeeT *zb, ZbZclCommandReqT *zclReq,
        void (*callback)(struct ZbZclCommandRspT *rsp, void *arg), void *arg);
```

Parameters

- zb – Zigbee stack structure
- req – ZCL Command Request
- callback – function to call on completion
- arg – callback argument

For more information, refer to section 2.4 of [R3].

10.5.2 ZbZclClusterCommandReq

Issues a client command, with a callback on completion.

Prototype

```
void ZbZclClusterCommandReq(struct ZbZclClusterT *clusterPtr,
        struct ZbZclClusterCommandReqT *req,
        void (*callback)(struct ZbZclCommandRspT *zcl_rsp, void *arg), void *arg);
```

Parameters

- clusterPtr – ZCL cluster structure
- req – ZCL cluster command Request
- callback – function to call on completion
- arg – callback argument

10.5.3 ZbZclSendDefaultResponse

Sends a default response command.

Prototype

```
void ZbZclSendDefaultResponse(struct ZbZclClusterT *clusterPtr,
        ZbApsdeDataIndT *dataIndPtr, struct ZbZclHeaderT *zclHdrPtr,
        enum ZclStatusCodeT status);
```

Parameters

- clusterPtr – ZCL cluster structure
- dataIndPtr – source of originating command.
- zclHdrPtr – ZCL hHeader structure of the originating command
- status – ZCL status code to use in the Default Response

For more information, refer to section 2.5.12 of [R3].

10.6 Built-in clusters

10.6.1 ZCL device log (smart energy)

10.6.1.1 ZbZclDeviceLogEnable

Enables a ZCL device log.

Prototype

```
enum ZclStatusCodeT ZbZclClusterRegisterAlarmResetHandler(
        struct ZbZclClusterT *clusterPtr, ZbZclAlarmResetFuncT callback);
```

Parameters

- clusterPtr – ZCL cluster structure
- callback – function to call on completion

10.6.1.2 ZbZclDeviceLogAdd

Adds a ZCL device log for a device.

Prototype

```
bool ZbZclDeviceLogAdd(struct ZigBeeT *zb, uint64_t ext_addr);
```

Parameters

- zb – Zigbee stack structure
- ext_addr – extebded address of device

Returns true on success, false otherwise.

10.6.1.3 ZbZclDeviceLogRemove

Removes a device's ZCL device log.

Prototype

```
bool ZbZclDeviceLogRemove(struct ZigBeeT *zb, uint64_t ext_addr);
```

Parameters

- zb – Zigbee stack structure
- ext_addr – extended address of device

Returns true on success, false otherwise.

10.6.1.4 *ZbZclDeviceLogClear*

Prototype

```
bool ZbZclDeviceLogClear(struct ZigBeeT *zb);
```

Parameters

- zb – Zigbee stack structure

Returns true on success, false otherwise.

10.7 Utility

10.7.1 ZbZclParseHeader

Parses a ZCL header from a received frame.

Prototype

```
int ZbZclParseHeader(struct ZbZclHeaderT *zclHdrPtr, const uint8_t *buf,
        unsigned int len);
```

Parameters

- zclHdrPtr – ZCL header structure
- buf – Packet buffer
- len – Packet length

Returns the length of the ZCL header, or <0 on error.

10.7.2 ZbZclPrependHeader

Builds and appends a ZCL header to the end of a buffer.

Prototype

```
int ZbZclPrependHeader(struct ZbZclHeaderT *zclHdrPtr, uint8_t *data,
        unsigned int len);
```

Parameters

- zclHdrPtr – ZCL header structure
- data – (OUT) buffer
- len – Size of the buffer

Returns the length of data written, or <0 on error.

10.7.3 ZbZclAppendHeader

Builds and appends a ZCL header to the start of a buffer.

Prototype

```
int ZbZclAppendHeader(struct ZbZclHeaderT *zclHdrPtr, uint8_t *data,
        unsigned int max_len);
```

Parameters

- zclHdrPtr – ZCL header structure
- data – (OUT) buffer

- len – Size of the buffer

Returns the length of data written, or <0 on error.

10.7.4 ZbZclAppendReportConfig

Appends a reporting configuration record to the end of the provided buffer.

Prototype

```
int ZbZclAppendReportConfig(struct ZbZclReportConfigT *configPtr,
        uint8_t *payload, unsigned int max_len);
```

Parameters

- configPtr – reporting configuration structure
- payload – (OUT) buffer
- max_len – size of the buffer

Returns the length of data written, or <0 on error.

10.7.5 ZbZclClusterInitApsdeReq

A helper function that initializes the APS data request with appropriate defaults using the provided cluster instance. The resulting APS data request must be used with that cluster.

Prototype

```
void ZbZclClusterInitApsdeReq(struct ZbZclClusterT *clusterPtr, ZbApsdeDataReqT *apsReq,
ZbApsdeDataIndT *dataIndPtr);
```

Parameters

- clusterPtr – ZCL cluster structure
- apsReq – APS request to be initialized
- dataIndPtr – APS data indication used to populate the APS data request

10.7.6 ZbZclClusterInitCommandReq

A helper function that initializes the cluster command request with appropriate defaults using the provided cluster instance. The resulting command request must be used with that cluster.

Prototype

```
void ZbZclClusterInitCommandReq(struct ZbZclClusterT *clusterPtr,
        ZbZclCommandReqT *cmdReq);
```

Parameters

- clusterPtr – ZCL cluster structure
- cmdReq – command request

10.7.7 ZbZclClusterRegisterAlarmResetHandler

Registers a callback function which is invoked on receipt of a reset alarm command in order to reset the alarms on this specific cluster.

Prototype

```
enum ZclStatusCodeT ZbZclClusterRegisterAlarmResetHandler(
        struct ZbZclClusterT *clusterPtr, ZbZclAlarmResetFuncT callback);
```

Parameters

- clusterPtr – ZCL cluster structure
- callback – function to call on completion

Returns the ZCL status code.

10.7.8 ZbZclClusterSendAlarm

Generates a ZCL alarm message using the cluster in which it is detected, making appear that it originates from the alarms cluster (on the same endpoint).

Prototype

```
void ZbZclClusterSendAlarm(struct ZbZclClusterT *clusterPtr,
        uint8_t src_endpoint, uint8_t alarm_code);
```

Parameters

- clusterPtr – cluster where the condition is detected (not Alarms cluster)
- src_endpoint – source endpoint
- alarm_code – cluster specific alarm code

10.7.9 ZbZclGetNextSeqnum

Gets the next outgoing sequence number.

Prototype

```
uint8_t ZbZclGetNextSeqnum(void);
```

Returns the next sequence number.

10.8 Basic cluster

10.8.1 ZbZclBasicServerResetCmdConfig

Controls whether the basic server is allowed to process the ZCL_BASIC_RESET_FACTORY command

Prototype

```
void ZbZclBasicServerResetCmdConfig(struct ZigBeeT *zb, bool allow_reset);
```

Parameters

- zb - Zigbee stack instance
- allow_reset – see description

10.8.2 ZbZclBasicWriteDirect

Writes to the basic server's local attributes (for example: ZCL_BASIC_ATTR_MFR_NAME)

Prototype

```
enum ZclStatusCodeT ZbZclBasicWriteDirect(struct ZigBeeT *zb, uint8_t endpoint, uint16_t
attributeId, const void *ptr, unsigned int len);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint
- attributeId – attribute ID
- ptr – pointer
- len – length

Returns the status.

10.8.3 ZbZclBasicPostAlarm

Posts an alarm code to the basic server cluster.

Prototype

```
bool ZbZclBasicPostAlarm(struct ZigBeeT *zb, uint8_t endpoint, uint8_t alarm_code);
```

Parameters

- zb – Zigbee stack structure
- endpoint – endpoint
- alarm_code – alarm code

Returns true on success, false otherwise.

10.8.4 ZbZclBasicServerDefaults

Configures the default ZCL basic server attribute values. The basic server is integral to the stack in order for the attribute values to be made "global" and shared between all basic server instances on all endpoints.

```
void ZbZclBasicServerDefaults (struct ZigBeeT *zb, const struct ZbZclBasicServerDefaults
*defaults)
```

Parameters

- zb – Zigbee stack structure
- defaults – Pointer to the default configuration data structure

Revision history

Table 15. **Document revision history**

Date	Version	Changes
24-Jul-2020	1	Initial release.

Contents

List of tables