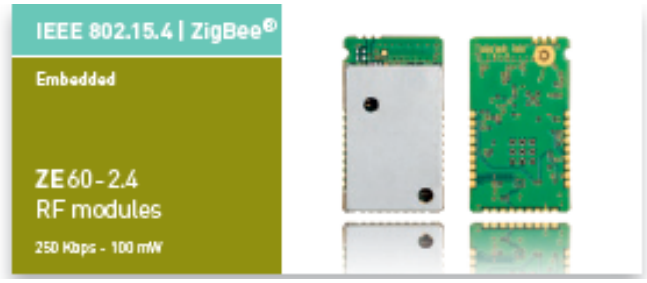
Z-ONE Protocol Stack User Guide

1vv0300820 Rev.0 – 17/04/2009

Making machines talk.

This document is related to the following products :



DISCLAIMER

The information contained in this document is the proprietary information of Telit Communications S.p.A. and its affiliates ("TELIT"). The contents are confidential and any disclosure to persons other than the officers, employees, agents or subcontractors of the owner or licensee of this document, without the prior written consent of Telit, is strictly prohibited.

Telit makes every effort to ensure the quality of the information it makes available. Notwithstanding the foregoing, Telit does not make any warranty as to the information contained herein, and does not accept any liability for any injury, loss or damage of any kind incurred by use of or reliance upon the information.

Telit disclaims any and all responsibility for the application of the devices characterized in this document, and notes that the application of the device must comply with the safety standards of the applicable country, and where applicable, with the relevant wiring rules.

Telit reserves the right to make modifications, additions and deletions to this document due to typographical errors, inaccurate information, or improvements to programs and/or equipment at any time and without notice. Such changes will, nevertheless be incorporated into new editions of this document.

.

Contents

1 INTRODUCTION

1.1 ZigBee Stack presentation

1.1.1 ZigBee Standard

What is ZigBee?

ZigBee and IEEE 802.15.4 are standards-based protocols that provide the network infrastructure required for wireless sensor network applications. 802.15.4 defines the physical and MAC layers, and ZigBee defines the network and application layers.
For sensor network applications, key design requirements revolve around long battery life, low cost, small footprint, and mesh networking to support communication between large numbers of devices in an interoperable and multi-application environment.

Typical Applications

There are numerous applications that are ideal for the redundant, self-configuring and self-healing capabilities of ZigBee wireless mesh networks. Key ones include

- Energy Management and Efficiency—To provide greater information and control of energy usage, provide customers with better service and more choice, better manage resources, and help to reduce environmental impact.
- Home Automation—To provide more flexible management of lighting, heating and cooling, security, and home entertainment systems from anywhere in the home.
- Building Automation—To integrate and centralize management of lighting, heating, cooling and security.
- Industrial Automation—To extend existing manufacturing and process control systems reliability.

The interoperable nature of ZigBee means that these applications can work together, providing even greater benefits.

About the ZigBee Alliance

The ZigBee Alliance is an association of over 285 companies working together to enable reliable, cost-effective, low-power, wirelessly networked, monitoring and control products based on an open global standard. Their focus is on the following:

- Defining the network, security and application software layers
- Providing interoperability and conformance testing specifications
- Promoting the ZigBee brand globally to build market awareness

- Managing the evolution of the technology

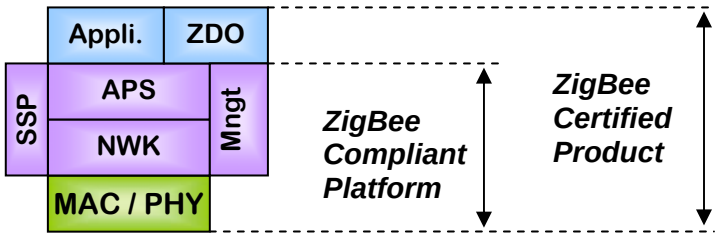
Product Certification

For a product to carry the ZigBee Alliance logo, it must first successfully complete the ZigBee Certification Program. This ensures that the product complies with the standards described in the ZigBee specification.

Only those products that pass ZigBee certification can display the ZigBee logo.
There are two ZigBee certified testing programs:

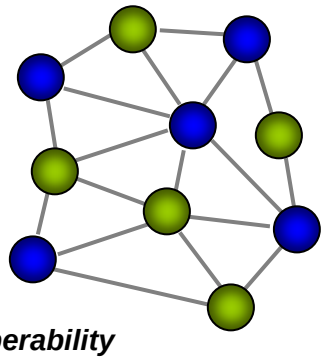
- ZigBee Compliant Platform (ZCP) The ZCP program applies to modules or platforms that are intended as building blocks for end products.

- ZigBee Certified Products This program applies to end products that are built upon a ZigBee Compliant Platform. After successful completion, these products can display the ZigBee logo.

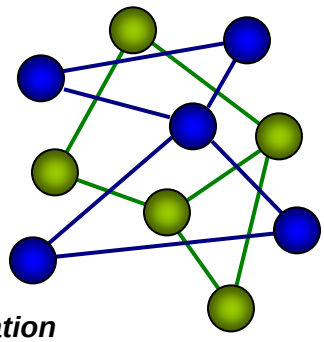


Products that use public application profiles are tested to ensure interoperability with other ZigBee end products.

Products that use manufacturer-specific profiles, which will operate as “closed systems”, are tested to ensure they can coexist with other ZigBee systems: that is, they do not adversely impact the operation of other ZigBee-certified products and networks.

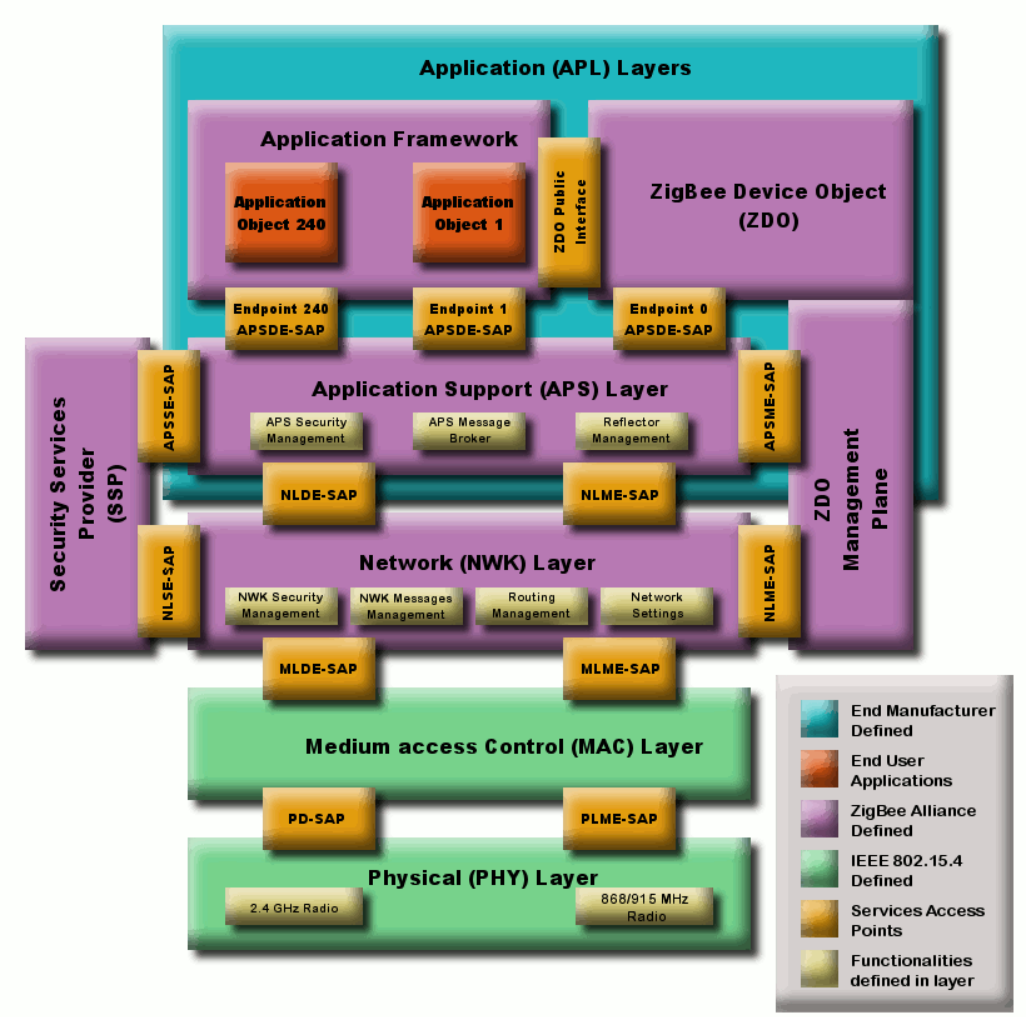


Interoperability



Cohabitation

1.1.2 ZigBee Stack



Application (APL) Layer

The top layer in the ZigBee protocol stack consists of the Application Framework, ZigBee Device Object (ZDO), and Application Support (APS) Sublayer.

Application Framework

Provides a description of how to build a profile onto the ZigBee stack (to help ensure that profiles can be generated in a consistent manner). It also specifies a range of standard data types for profiles, descriptors to assist in service discovery, frame formats for transporting data, and a key value pair construct to rapidly develop simple attribute-based profiles.

Application Objects

Software at an endpoint that controls the ZigBee device. A single ZigBee node supports up to 240 application objects. Each application object supports endpoints numbered between 1 and 240 (with endpoint 0 reserved for the ZigBee Device Object (ZDO)).

ZigBee Device Object (ZDO)

Defines the role of a device within the network (coordinator, router or end device), initiates and/or responds to binding and discovery requests, and establishes a secure relationship between network devices. It also provides a rich set of management commands defined in the ZigBee Device Profile (used in ZigBee commissioning). The ZDO is always endpoint zero.

ZDO Management Plane

Facilitates communication between the APS and NWK layers with the ZDO. Allows the ZDO to deal with requests from applications for network access and security using ZDP (ZigBee Device Profile) messages.

Application Support (APS) Sublayer

Responsible for providing a data service to the application and ZigBee device profiles. It also provides a management service to maintain binding links and the storage of the binding table itself.

Security Service Provider (SSP)

Provides security mechanisms for layers that use encryption (NWK and APS). Initialized and configured through the ZDO.

Network (NWK) Layer

Handles network address and routing by invoking actions in the MAC layer. Its tasks include starting the network (coordinator), assigning network addresses, adding and removing network devices, routing messages, applying security, and implementing route discovery.

IEEE 802.15.4

Medium Access Control (MAC) Layer

Responsible for providing reliable communications between a node and its immediate neighbors, helping to avoid collisions and improve efficiency. The MAC Layer is also responsible for assembling and decomposing data packets and frames.

Physical (PHY) Layer

Provides the interface to the physical transmission medium (e.g. radio). The PHY layer consists of two layers that operate in two separate frequency ranges. The lower frequency PHY layer covers both the 868MHz European band and the 915MHz band used in countries such as the US and Australia. The higher frequency PHY layer (2.4GHz) is used virtually worldwide.

2 Z-One Stack Organization

2.1 Introduction

The Z-One 2007 ZigBee Stack is composed of :

- a set of libraries, one for each type of device (Coord, Router, End Device) containing the non modifiable core of the stack.
- a set of .c files containing the functions the user can modify
- a set of .h files giving access to the functions and data from the core stack open to the user
- An IAR project allowing to build application for the different devices

The functions the user can use and/or modify act at different levels :

- ZigBee Device Object (ZDO) for high level network management
- Application Framework (AFW) for high level application management : Start, Data send/receive, endpoints management
- Application Support Layer for Security use and Tables management (Bindings, Groups, Addresses)
- Network Layer (NWK) for network data access
- Stack Support for the Security management, init and main functions, and different tools (RF Events indication, scheduler,…)
- Hardware Abstraction Library (HAL) gives access to the physical fuctions of the system (Serial, EEPROM, Interrupts, Timers,ADC)

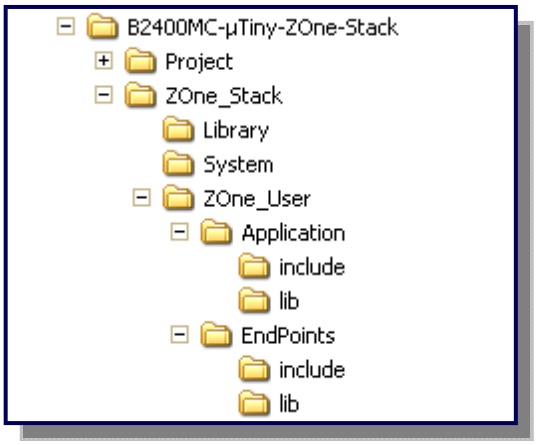
All these functions can be sorted in three types :

- Requests : called by the user to start a functionality, internally or through the radio link
- Indications : Informs the user of an incoming event
- Response : Gives to the user the result of a request.

Usually the Request type functions are part of the core stack and not modifiable by the user. The Response and Indication types are usually modifiable so the application can react to a specific event.

Most of the functions in the Zone Stack reflect primitives described in the ZigBee 2007 Specification. Please refer to this document for exact frames and descriptors format, as well as extended description of the use of the functions

2.2 Files organization



Project :

This folder contains the IAR project with all related files, objects, and final compiled files.

Zone_Stack :

This sub-tree contains all the files needed to build your project : libraries, configuration files and sources

Library :

Stores the different libraries of the Zone Stack, to be included in your project.

The files are different and depends on the type of device : Coordinator, Router or End Device.

System :

Different option and configuration settings to be used at compile or link time, each type of device using a different set of files

Zone_User :

Contains all the source files for your project.

The files provided can be used as a baseline for your project, as they contain the source code for the demonstration software. It includes the management of the serial endpoint, of the analog input endpoint and for the different digital inputs and outputs.

All the management functions are also provided, using the serial protocol described in the Democase manual. They can be modified to fit your own needs.

Application :

Upper layers functions, including Application Support, Hardware Abstraction and device and network management (ZDO)

Include : Header files

Lib : C files

EndPoints :

Endpoints management functions

Include : Header files

Lib : C files

2.3 IAR Project description

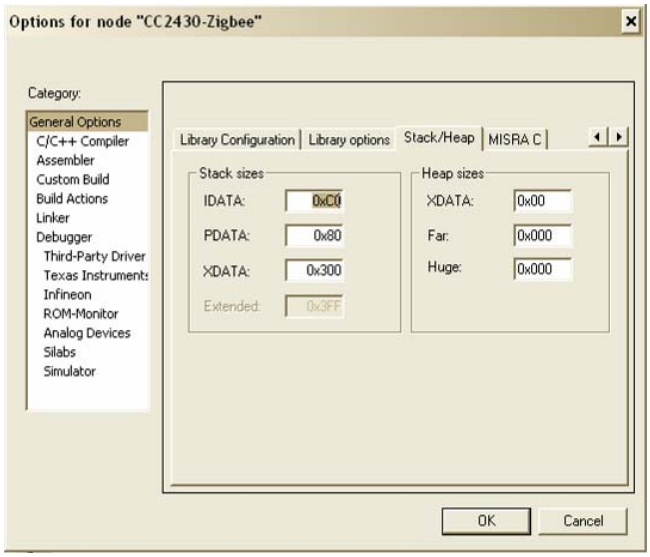
The existing projects have been developed and tested using the integrated environment tool "IAR Embedded Workbench IDE" version 7.40A

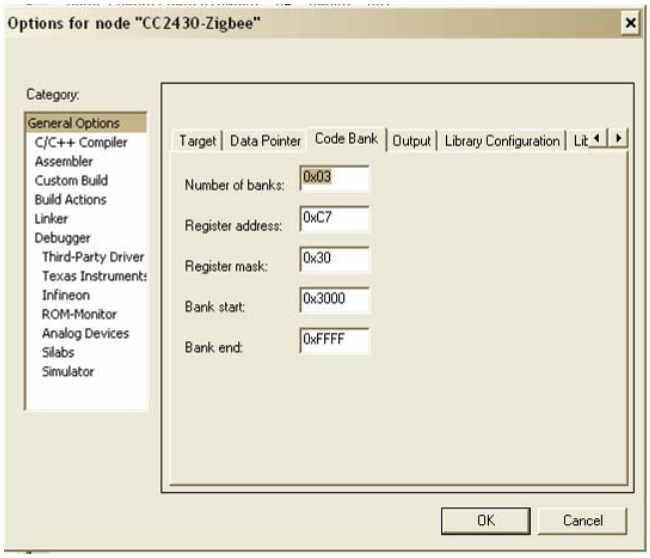
This needs a few parameters settings explained hereafter:

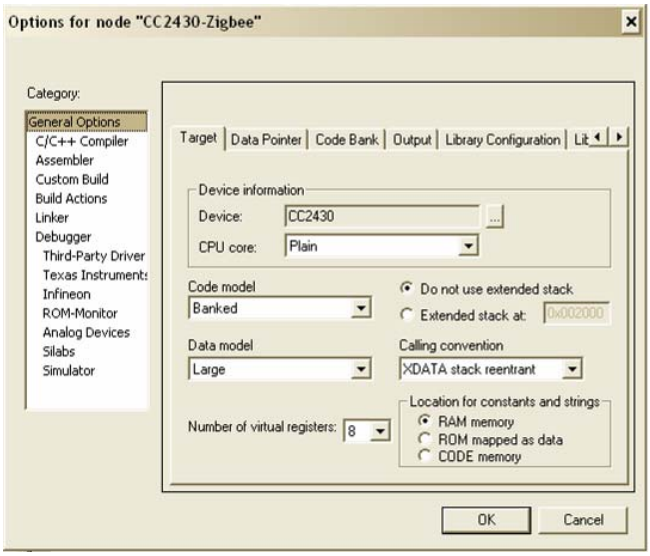
2.3.1 Project Parameters

All these parameters are already set in the provided application. The settings can be slightly different from one type of device to another, and from application to debug projects.

2.3.1.1 General

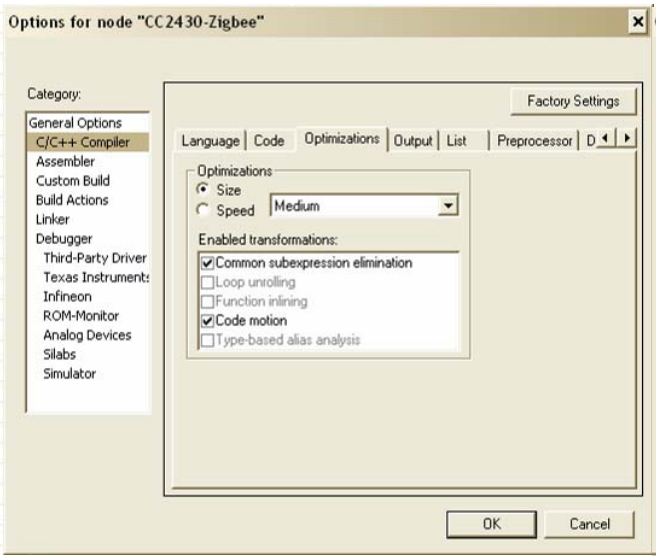


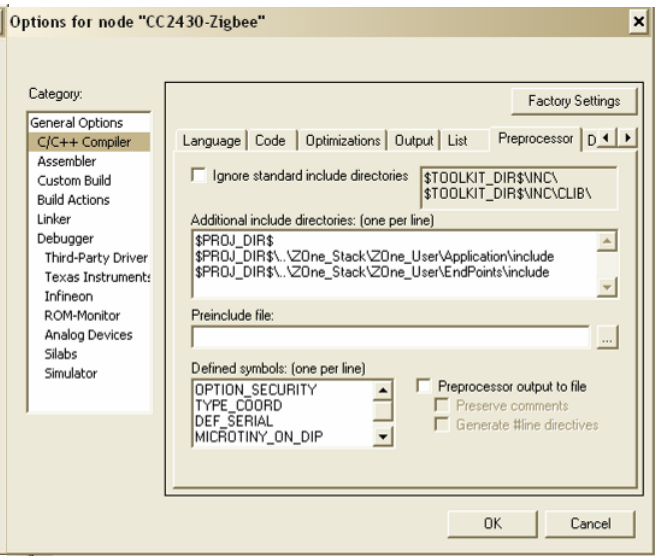




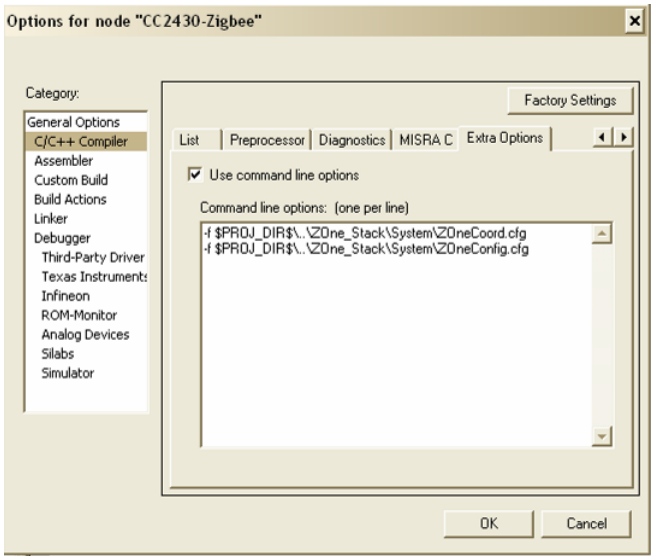
Sets the type of target (CC2430), the memory model, and the values for the stacks and heap.

2.3.1.2 Compiler





Defined Symbols: see 0 Compiler Defines for full list and explanations



The command lines options are device type dependant :

For Coordinator and End Devices :

-f $PROJ_DIR$\..\ZOne_Stack\System\ZOneCoord.cfg

-f $PROJ_DIR$\..\ZOne_Stack\System\ZOneConfig.cfg

For Routers:

-f $PROJ_DIR$\..\ZOne_Stack\System\ZOneRouter.cfg

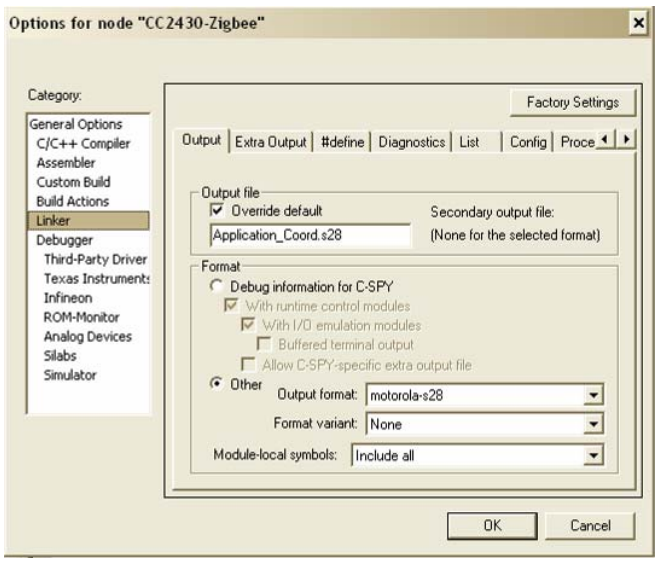
-f $PROJ_DIR$\..\ZOne_Stack\System\ZOneConfig.cfg

2.3.1.3 Compiler Defines

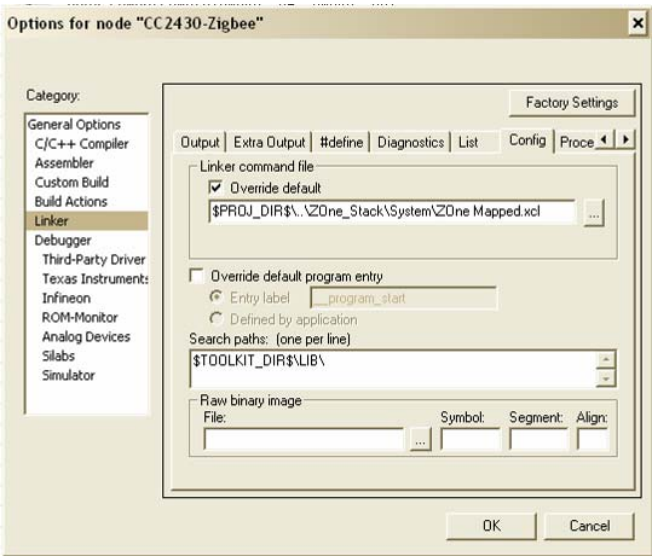
Some compiler defines must be set in the Preprocessor tab of the C/C++ compile Options window :

- TYPE_COORD Sets the module type as Coordinator
- TYPE_ROUTER Sets the module type as Router
- TYPE_RFD Sets the module type as End Device
- DEF_SERIAL Activates the serial link of the module
- MICROTINY_ON_DIP Defines the endpoints of the ZEXX-2.4 board in democase
- BOOTLOADER Compiles the project as a firmware flashed on top of the Bootloader
- MGT_INTERFACE Activates all the management functions to/from the serial link
- TEST_PROFILE Used for Compliance tests. Unused for applications

2.3.1.4 Linker



The output format must be set to Motorola-s28 for compatibility with ZTC and the bootloader.



The linker command file must be overrided for the different memory segments definition.

There are 4 different linker command files :

2 for applications :

- Coordinator and Router :

  "$PROJ_DIR$\..\ZOne_Stack\System\ZOne Mapped.xcl"

- End Devices :

  "$PROJ_DIR$\..\ZOne_Stack\System\ZOne-SleepingDevice Mapped.xcl"

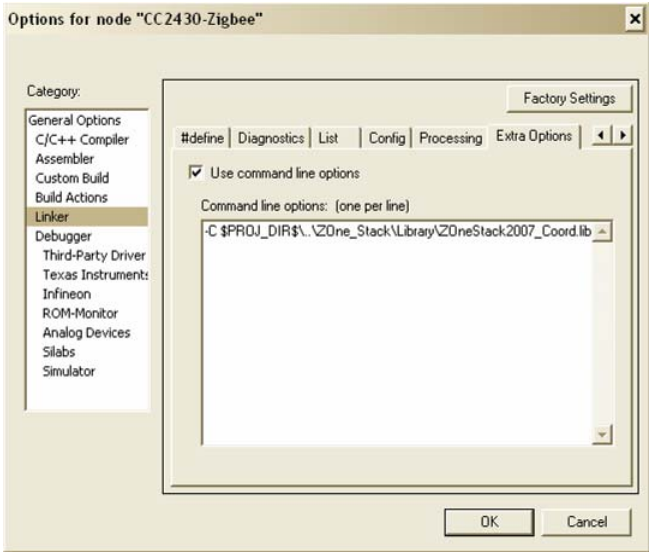
2 for debug, without the memory mapping incompatible with the debugger :

- Coordinator and Router :

  "$PROJ_DIR$\..\ZOne_Stack\System\ZOne.xcl"

- End Devices :

  "$PROJ_DIR$\..\ZOne_Stack\System\ZOne-SleepingDevice.xcl"



The command line option is very important, is the link to the used library file.

It can be any of the following lines:

3 for applications compatible with the Bootloader and ZTC :

- Coordinator:

  -C $PROJ_DIR$\..\ZOne_Stack\Library\ZOneStack2007_Coord.lib

- Router:

  -C $PROJ_DIR$\..\ZOne_Stack\Library\ZOneStack2007_Router.lib

- End Devices:

  -C $PROJ_DIR$\..\ZOne_Stack\Library\ZOneStack2007_EndDevice.lib

2 for debug, without BOOTLOADER compiler define:

- Coordinator:

  -C $PROJ_DIR$\..\ZOne_Stack\Library\ZOneStack2007_Debug_Coord.lib

- Router:

  -C $PROJ_DIR$\..\ZOne_Stack\Library\ZOneStack2007_Debug _Router.lib

- End Devices:

  -C $PROJ_DIR$\..\ZOne_Stack\Library\ZOneStack2007_Debug _EndDevice.lib

2.3.2 Bootloader and Debug projects

The ZEXX-2.4 boards are provided already flashed with a bootloader allowing to reflash the application software using a serial link or over the air.

This bootloader takes 4 kBytes of flash memory, no RAM, and redirects the different interrupt vectors to the application. This is why the interrupts are defined as standard functions.

The ZTC software is able to manage the reflashing of a module using both communication support.

However, the bootloader isn't compatible with the IAR Debugger tool, as the tool will try to erase the flash or to write over protected areas.

The Zone 2007 Stack is thus provided with two versions, one compiled without bootloader for debugging, and one compiled with the bootloader compatibility.

It is suggested to flash as few as possible boards without bootloader, as it will be impossible for the customer to reflash it afterward. Moreover, the bootloader is used to store the IEEE MAC address of the module, and it will be lost if it is erased. Only Telit RF Technologies can reflash a new bootloader/IEEE address.

In order to define the MAC Address of each module in Debug mode, a specific function is defined in ZOne.h in the "Debug" stack files : ***SetExtendedAddress***.

It must be called if the Debug stack is used !

3 Z-One Functions

3.1 Types and Tools

3.1.1 ZOneTypes.h

This header file contains all the definitions for the used types in the stack, as well as a few generic macro instructions.

3.1.2 ZOneTools.h / ZOneTools.c

Generic tools used by the stack to manipulate special types of variables, and to access some device-specific values like the IEEE MAC address, the stack revision, the acquired status …

3.1.2.1 CompareQword

Function	BOOL CompareQword(QWORD *pA, QWORD *pB);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Function	CompareQword	
Header file	ZOneTools.h	
C file	-	**Value**
Arguments	QWORD *pA, QWORD *pB	0x0000000000000000 - 0xFFFFFFFFFFFFFFFF
Return	BOOL	TRUE or FALSE

Compares two QWORD values. Returns TRUE if equivalent, and FALSE if different.

As a QWORD is defined as a table, the values are passed as pointers

3.1.2.2 ReverseQword

Function	void ReverseQword(QWORD *pA);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Function	ReverseQword	
Header file	ZOneTools.h	
C file	-	**Value**
Arguments	QWORD *pA,	0x0000000000000000 - 0xFFFFFFFFFFFFFFFF
Return	-	

Reverses the content of a QWORD to switch from Little Endian (used in radio transmissions) to Big Endian (used to communicate with user)

3.1.2.3 ReverseCopy

Function	void ReverseCopy(BYTE *pDestination, BYTE *pSource, UINT8 length);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments	BYTE *pDestination, BYTE *pSource, UINT8 length	Pointer Pointer 0x00-0xFF
Return	-	

Copy a source memory zone to a destination memory zone starting from the end. This is useful in case of superposition of the beginning of the destination zone with the end of the source zone.

3.1.2.4 MemCopyLE

Function	void MemCopyLE(BYTE *pDestination, BYTE *pSource, BYTE length);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments	BYTE *pDestination,	Pointer

	BYTE *pSource, UINT8 length	Pointer 0x00-0xFF
Return	-	

Copy a source memory zone to a destination memory zone in reverse order. This is useful to reverse Endianness when storing a data.

3.1.2.5 MemReverse

Function	void MemReverse(BYTE *pSource, UINT8 length);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments	BYTE *pSource, UINT8 length	Pointer 0x00-0xFF
Return	-	

Reverse the content of a memory zone beginning at *pSource* and of *length* size

3.1.2.6 GetRandomByte

Function	BYTE GetRandomByte(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments		
Return	BYTE	0x00-0xFF

Generates a random value and returns an unsigned char. The seeding of the random number generator is done at startup, before calling the *ZOneMain()* function.

3.1.2.7 getStackRevision

Function	WORD getStackRevision(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments		
Return	WORD	0x0000-0xFFFF

Returns the version number of the Zone Stack. The high byte gives the main revision value, the low byte the index in this revision. “0x010A” means “Version 01.10”

3.1.2.8 getCurrentChannel

Function	BYTE getCurrentChannel(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments		
Return	BYTE	0x0B-0x1A

Returns the value of the radio channel currently used. In the 2.4 GHz band, the channels are numbered from 11 to 26.

3.1.2.9 IsAcquired

Function	BOOL IsAcquired(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments		
Return	BOOL	TRUE - FALSE

Returns the acquired status of the device : TRUE if the device successfully joined or rejoined a network (or created one if the device is a coordinator), FALSE if the device isn't part of a network

3.1.2.10 setRxOnWhenIdle

Function	BOOL IsAcquired(void);		
Available for :	✗ **Coordinator**	✗ **Router**	✓ **End Device**

Function	setRxOnWhenIdle	
Header file	ZOneTools.h	
C file	-	**Value**
Arguments	BOOL aRxOnWhenIdle	TRUE : Device always ON FALSE : Sleeping device
Return	-	-

Sets the type of an End Device. Must be used before association. A device can't change this status once associated.

3.1.2.11 getRxOnWhenIdle

Function	BOOL getRxOnWhenIdle(void);		
Available for :	✗ **Coordinator**	✗ **Router**	✓ **End Device**

Function	getRxOnWhenIdle	
Header file	ZOneTools.h	
C file	-	**Value**
Arguments	-	-
Return	BOOL aRxOnWhenIdle	TRUE : Device always ON FALSE : Sleeping device

Returns the type of End Device

3.1.2.12 getIEEEAddress

Function	QWORD * getIEEEAddress(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	-	**Value**
Arguments		
Return	QWORD *	Pointer to an 8 bytes address

Returns a pointer to the IEEE Address of the module.

3.1.2.13 AdcInit

Function	void AdcInit (void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.h	
C file	ZOneTools.c	**Value**
Arguments	-	-
Return	-	-

Example of how to initialize the Analog-Digital converter.

3.1.2.14 CalculAnalogicIn_10bit

Function	WORD CalculAnalogicIn_10bit(BYTE channel);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTools.c	
C file	-	**Value**
Arguments		
Return	WORD	0x0000 – 03FF

Example of how to read an analog value from the Analog-Digital converter.

3.1.3 ZOneVectors.h / ZOneVectors.c

These files are needed for the linking of the project, as these functions are mapped in the memory at fixed places.

DO NOT MODIFY THESE FILES

The interrupt functions freely usable by the user are described in the next chapter "ZOneInterrupts.h / ZOneInterrupts.c"

3.1.4 ZOneInterrupts.h / ZOneInterrupts.c

These functions are given as exemples of how to manage the external interrupts called from the ZOneVectors procedures.

```
__near_func void ZOneP1_int(void);
__near_func void ZOneP2_int(void);
```

3.1.4.1 ZOneIntIO_Enable

Function	void ZOneIntIO_Enable(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneInterrupts.h	
C file	ZOneInterrupts.c	**Value**
Arguments	-	-
Return	-	-

Used to enable the interrupts coming from the ports P0 (pin IO5) and P1 (pin Standby).

3.1.4.2 ZOneIntIO_Disable

Function	void ZOneIntIO_Disable(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneInterrupts.h

C file	ZOneInterrupts.c	Value
Arguments	-	-
Return	-	-

Used to disable the interrupts on the external input pins.

3.1.4.3 ZOneP0_int

Function	void ZOneP0_int(void);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneInterrupts.h	
C file	ZOneInterrupts.c	Value
Arguments	-	-
Return	-	-

Called when an interrupt condition occurred on one of the Port 0 pins, if this has been enabled previously.

Used to manage IO3 (P0_2) to IO8 (P0_7) interrupts

3.1.4.4 ZOneP1_int

Function	void ZOneP1_int(void);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneInterrupts.h	
C file	ZOneInterrupts.c	Value
Arguments	-	-
Return	-	-

Called when an interrupt condition occurred on one of the Port 1 pins, if this has been enabled previously.

Used to manage IO1 (P1_0), IO2 (P1_1), IO9 (P1_6) and Standby (P1_7) interrupts

3.1.4.5 ZOneP2_int

Function	void ZOneP2_int(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneInterrupts.h	
C file	ZOneInterrupts.c	**Value**
Arguments	-	-
Return	-	-

Called when an interrupt condition occurred on one of the Port 2 pins, if this has been enabled previously.

Used to manage Prog (P2_0), DEBUG_D (P2_1) and DEBUG_C (P2_2) interrupts. As these I/O are used for other functions, **they must be set to High impedance at module's startup**, or it can lock itself in debug or bootloader mode.

3.1.4.6 ZOneADC_int, ZOneEnc_int, ZOneTimer1_int, ZOneWDT_int

Called when an interrupt condition occurred on one of the Analog Digital Interrupt, the encryption interrupt, the timer1 interrupt or the watchdog interrupt, if this has been enabled previously.

They are not used in the sample stack provided but can be filled if the application needs one of these functionalities.

3.2 Hardware Abstraction Library

3.2.1 ZOnePorts.h

This header file defines the correspondence between the CC2430 GPIO and the ZEXX-2.4 board's pinout.

3.2.2 ZOneEeprom.h / ZOneEeprom.c

The EEPROM is common and is used by both the stack and the user's application.

Part of it is freely available, and a set of functions is provided to access it. It is recommended to use them in order to avoid modification of parameters used for the ZigBee stack, as an unfortunate change can lock the system.

248 bytes in the EEPROM are dedicated to user's application, and can be accessed through the non-modifiable functions *ZOneEeprom_Read* and *ZOneEeprom_Write*.

The functions *ZOneEepromAllRead*, *ZOneEepromAllWrite* and *ZOneEeprom_Init* can be modified to fit the application's needs.

3.2.2.1 ZOneEepromRead

Function	BYTE ZOneEeprom_Read(WORD iAddress, BYTE * szData, WORD uiNumber);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneEeprom.h	
C file	-	**Value**
Arguments	WORD iAddress BYTE * szData WORD uiNumber	0x00 – 0xF7 Pointer to recipient memory block Block size to transfer to memory
Return	BYTE	1 : Success, 0 : Failed

Copies *uiNumber* bytes from the address *iAddress* of the EEPROM to the place pointed by *szData*. Returns 0 if a problem occurred when accessing the EEPROM, 1 if successful.

3.2.2.2 ZOneEeprom_Write

Function	BYTE ZOneEeprom_Write(WORD iAddress, BYTE * szData, WORD uiNumber);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneEeprom.h	
C file	-	**Value**
Arguments	WORD iAddress BYTE * szData WORD uiNumber	0x00 – 0xF7 Pointer to source memory block Block size to transfer to EEPROM
Return	BYTE	1 : Success, 0 : Failed

Copies *uiNumber* bytes from the place pointed by *szData* to the address *iAddress* of the EEPROM. Returns 0 if a problem occurred when accessing the EEPROM, 1 if successful.

3.2.2.3 zdoSetUserDescriptor

Function	void zdoSetUserDescriptor(BYTE * pDescriptor);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneEeprom.h	
C file	-	**Value**
Arguments	BYTE * pDescriptor	Pointer to a 16 bytes string
Return	-	-

Stores in the reserved part of the EEPROM the User Descriptor of the module. This will "name" the device when discovered by a network discovery tool

3.2.2.4 ZOneEeprom_Init

Function	void ZOneEeprom_Init(BYTE cReWrite);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneEeprom.h	
C file	ZOneEeprom.c	**Value**
Arguments	BYTE cReWrite	0 : Don't force EEPROM writing 1 : Force EEPROM write
Return	-	-

This function is given as an example.

In this case, it tests the application's version number and if it changed it writes the default values present in memory to EEPROM. Else, it copies the values saved in EEPROM to memory. This avoids wrong settings if the mapping of the EEPROM changed between the versions

3.2.2.5 ZOneEepromAllRead

Function	void ZOneEepromAllRead(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneEeprom.h	
C file	ZOneEeprom.c	**Value**

Arguments	-	-
Return	-	-

This function is given as an example.

In this case, it reads all the application parameters from EEPROM and stores them in memory

3.2.2.6 ZOneEepromAllWrite

Function	void ZOneEepromAllWrite(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneEeprom.h	
C file	ZOneEeprom.c	**Value**
Arguments	-	-
Return	-	-

This function is given as an example.

In this case, it writes all the application parameters stored in memory to the EEPROM

3.2.3 ZOneStandby.h

Defines the values and functions used to set the device in standby mode.

The Stack manages the standby for the sleeping End Devices.

The value ***gBySleepingTime*** defines the delay in minutes between two pollings from the end device. The variable ***cStandByExitMode*** is used to show how the device exited the sleeping mode : It can be from RTI (real Time Interrupt) after the given delay, or from a hardware interrupt on the StandBy input or another GPIO if it has been activated in the application.

3.2.3.1 StandBy_Mode

Function	void StandBy_Mode(void);		
Available for :	✘ **Coordinator**	✘ **Router**	✔ **End Device**

Header file	ZOneStandby.h	
C file	-	**Value**
Arguments	-	-
Return	-	-

Sets the End device in standby after activating the RTI interrupt for a sleeping time equal to ***gBySleepingTime.***

The function exits after awakening. On exit, the clock is already switched to the 16MHz oscillator, and the device already polled its parent to check for messages. The source of the awakening interrupt is indicated in ***cStandByExitMode***. It can be

StandByExitRTI (0x01) : Normal clock interrupt

StandByExitKBD (0x02) : GPIO interrupt

StandByExitHard (0x03) : Standby Input interrupt

3.2.4 ZOneTimer.h / ZOneTimer.c

In order to share efficiently the timer resources of the CC2430, some functions are provided to the application.

The main one is the ***ZOneTimerMillisecond_int*** function called by the Zone stack each millisecond.

As an exemple, two timing functions are included : a generic timer ***ZOneTimer*** and a serial timer ***ZOneTimer_Serial***. Both set a flag after a given number of milliseconds.

The sample application uses the serial timer to exit serial reception on timeout, and the generic timer to exit some loops if the intended action didn't happen to avoid being locked.

3.2.4.1 ZOneTimerMillisecond_int

Function	void ZOneTimerMillisecond_int(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimer.h	
C file	ZOneTimer.c	**Value**
Arguments	-	-
Return	-	-

Called by the Stack internal timer interrupt each millisecond.

Allows the application to do some periodic actions.

Used in the sample application to decrease some counting values and to set some flags after a defined number of milliseconds.

These flags are checked in the ***ZOneMain*** application loop.

3.2.4.2 ZOneTimer_Init

Function	void ZOneTimer_Init(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimer.h	
C file	ZOneTimer.c	**Value**
Arguments	-	-
Return	-	-

Initializes the application's timer values.

Used in the sample application to reset the counting variables and the timeout flags.

3.2.4.3 ZOneTimer

Function	void ZOneTimer(unsigned int iTime);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimer.h	
C file	ZOneTimer.c	**Value**
Arguments	unsigned int iTime	0x0000 – 0xFFFF
Return	-	-

Resets ***bZOneTimer_FlagT*** to FALSE and starts counting *iTime* milliseconds.

The value ***bZOneTimer_FlagT*** becomes TRUE when the counter reaches *iTime.*

3.2.4.4 ZOneTimer_Serial

Function	void ZOneTimer_Serial(unsigned int iTime);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimer.h	
C file	ZOneTimer.c	**Value**
Arguments	unsigned int iTime	0x0000 – 0xFFFF
Return	-	-

Resets ***bZOneTimer_FlagS*** to FALSE and starts counting *iTime* milliseconds.

The value ***bZOneTimer_FlagS*** becomes TRUE when the counter reaches *iTime.*

3.2.4.5 ZOneDelay

Function	void ZOneDelay(unsigned int iTime);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimer.h	
C file	ZOneTimer.c	**Value**
Arguments	unsigned int iTime	0x0000 – 0xFFFF
Return	-	-

Uses ***ZoneTimer*** to wait for *iTime* milliseconds.

Exits the functions at the end of the counting.

3.2.4.6 ZOneWait_MicroSec

Function	void ZOneWait_MicroSec(unsigned int iTime);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimer.h	
C file	ZOneTimer.c	**Value**
Arguments	-	-
Return	-	-

Generates a delay in micro seconds. The function exits after *iTime* µs

3.2.4.7 ZOneWait_100MicroSec

Function	void ZOneWait_100MicroSec(unsigned int iTime);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimer.h	
C file	ZOneTimer.c	**Value**

Arguments	-	-
Return	-	-

Generates a delay in micro seconds. The function exits after (100 x *iTime*) µs

3.2.5 ZOneSerial.h / ZOneSerial.c

The use of the UART is free, as only the ZOneMGTSerialInterface uses it, and the user can choose to modify or remove these functionalities from his application.

In order to ease the development, a set of serial link management tools is provided with the Zone stack.

The essential part is the five interrupt functions called by the stack when the corresponding event happens.

They allow the management of UART0 and UART1 through the Rx and Tx interrupts as well as DMA engine.

Other variables and functions are defined as sample, to be used if they fit application's need.

3.2.5.1 Variables

unsigned int cSerial_NbrRec

Gives the number of received bytes stored in *szSerial_BufferRec*

unsigned int cSerial_NbrSend

Gives the number of bytes stored in *szSerial_BufferSend* being currently sent

unsigned char szSerial_BufferRec[SERIAL_MAXI_BUFFERREC+1]

Reception buffer

unsigned char szSerial_BufferSend[SERIAL_MAXI_BUFFERSEND+1]

Transmission buffer

bZOneSerial_BeginRec

Value : TRUE or FALSE

Indicates that a reception started.

bZOneSerial_EndRec

Value : TRUE or FALSE

Indicates that a reception ended. Set when the timeout serial timer triggers.

bZOneSerial_BeginSend

Value : TRUE or FALSE

Indicates that a transmission started.

bZOneSerial_EndSend

Value : TRUE or FALSE

Indicates that a transmission ended.

3.2.5.2 ZOneUartTx0_int

Function	void ZOneUartTx0_int(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	-	-
Return	-	-

Called by the Stack internal serial interrupt when the UTX0 vector is triggered when a character has been sent.

Used when transmitting serial frame byte per byte.

This interrupt isn't used by the sample application. The DMA engine is activated to send serial frames, lessening the number of interrupts occurring in the system.

3.2.5.3 ZOneUartRx0_int

Function	void ZOneUartRx0_int(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	-	-
Return	-	-

Called by the Stack internal serial interrupt when the URX0 vector is triggered by a character reception. The application must read the Rx buffer before the next character is forced in it.

3.2.5.4 ZOneDma_int

Function	void ZOneDma_int(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	-	-
Return	-	-

Called by the Stack internal DMA interrupt.

Used in the sample application to indicates the end of a serial transmission on UART 0

3.2.5.5 ZOneUartTx1_int

Function	void ZOneUartTx1_int(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	-	-
Return	-	-

Called by the Stack internal serial interrupt when the UTX1 vector is triggered when a character has been sent.

Used when transmitting serial frame byte per byte.

The UART 1 in not used by the sample application. This function is defined in case the user's application uses this UART and acts as the interrupt vector for transmission, as all vectors are redirected by the stack.

3.2.5.6 ZOneUartRx1_int

Function	void ZOneUartRx1_int(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	-	-
Return	-	-

Called by the Stack internal serial interrupt when the URX1 vector is triggered by a character reception. The application must read the Rx buffer before the next character is forced in it.

The UART 1 in not used by the sample application. This function is defined in case the user's application uses this UART and acts as the interrupt vector for reception, as all vectors are redirected by the stack.

3.2.5.7 ZOneSerial_Init

Function	void ZOneSerial_Init(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	-	-
Return	-	-

Defines the position of UART0 pinout.

Initializes the UART0 registers in function of the stored values for baud rate, parity, character length and number of stop bytes.

Resets the serial flags and variables.

3.2.5.8 ZOneSerial_InitRec

Function	void ZOneSerial_InitRec(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	-	-
Return	-	-

Initializes the reception variables and flags to prepare the reception of the next frame. Called after a *ZOneSerial_Init()* and after each frame reception.

3.2.5.9 ZOneSerial_Send

Function	void ZOneSerial_Send(unsigned int cNbrData);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSerial.h	
C file	ZOneSerial.c	**Value**
Arguments	unsigned int cNbrData	0x01 – Buffer size
Return	-	-

Starts the transmission of *cNbrData* characters from the buffer *szSerial_BufferSend.*

The function returns immediately and the end of the transmission will be indicated by the flag *bZOneSerial_EndSend*.

3.3 Timed Tasks

3.3.1 Principle

A Timed Task is defined by :

- a function to call
- a delay in milliseconds before calling the function
- a value passed to the function (2 bytes)

Once it is set, the millisecond timer will decrease the value of the delay element, and when it reaches zero the function is called.

The function used in the Timed Task definition must be declared as follow :

*void TimedTaskFunction(volatile HAL_TIMEDTASK * pTTask);*

where *TimedTaskFunction* can be replaced by the fuction's name.

HAL_TIMEDTASK is a structure definition :

```
typedef struct {
    VFPTR pFunc;
    WORD  Data;
    WORD timeout;
    BYTE nextTTask;
    BYTE prevTTask;
    BOOL  occupied;
} HAL_TIMEDTASK;
```


The value defined as being passed to the function is available as *pTTask->Data* inside the function.

3.3.2 ZOneTimedTasks.h

The functions defined in ZOneTimedTasks.h are not modifiable by the user.

3.3.2.1 ttSetTTask

Function	BYTE ttSetTTask(VFPTR pFunc, WORD data, WORD timeout);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimedTasks.h	
C file	-	**Value**
Arguments	VFPTR pFunc WORD data WORD timeout	Pointer to a function Data to pass to *pFunc* Delay before calling *pFunc*
Return	BYTE	Timed Task Number

Define a Timed Task to call the function *pFunc* after *timeout* milliseconds with the value *data.*

The Timed Task number returned by the function can be used to cancel this scheduled task through *ttCancelTTask*.

3.3.2.2 ttCancelTTask

Function	BOOL ttCancelTTask(BYTE TTaskNumber);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTimedTasks.h	
C file	-	**Value**
Arguments	BYTE TTaskNumber	Number of the task to cancel
Return	BOOL	TRUE if success, FALSE if failed

Cancel a Timed Task identified by its number returned by *ttSetTTask.* If the task doesn't exist anymore, the function returns FALSE, else it removes the task and returns TRUE.

3.4 Network Layer Access

3.4.1 Principle

The user application need only a few access to the network layer, and mainly for management purpose. The Network layer access is thus limited to the functions and defines declared in ZOneNwkReq.h.

3.4.2 ZOneNwkReq.h

The functions defined in ZOneNwkReq.h are not modifiable by the user.

3.4.2.1 nlmeGetRequest

Function	BYTE nlmeGetRequest(NWK_IB_ATTR nibAttribute, void *pNibAttributeValue);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneNwkReq.h	
C file	-	**Value**
Arguments	NWK_IB_ATTR nibAttribute void *pNibAttributeValue	Attribute to read Pointer to result storage address
Return	BYTE	Status

Gets the value of a Network Information Base attribute.

The identifiers of the NWKIB are defined in the enumeration NWK_IB_ATTR in ZOneNwkReq.h.

The size of the attribute value written at *pNibAttributeValue* memory address is variable.

NWK_SEQUENCE_NUMBER : BYTE
NWK_PASSIVE_ACK_TIMEOUT : BYTE
NWK_MAX_BROADCAST_RETRIES : BYTE
NWK_MAX_CHILDREN : BYTE
NWK_MAX_DEPTH : BYTE
NWK_MAX_ROUTERS : BYTE
NWK_NETWORK_BROADCAST_DELIVERY_TIME : BYTE
NWK_REPORT_CONSTANT_COST : BYTE
NWK_ROUTE_DISCOVERY_RETRIES_PERMITTED : BYTE
NWK_SYM_LINK : BOOL
NWK_CAPABILITY_INFORMATION : BYTE
NWK_ADDR_ALLOC : BOOL

NWK_USE_TREE_ROUTING :	BOOL
NWK_TRANSACTION_PERSISTANCE_TIME :	WORD
NWK_PAN_ID :	WORD
NWK_MANAGER_ADDRESS :	WORD
NWK_UPDATE_ID :	BYTE
NWK_NETWORK_ADDRESS :	WORD
NWK_STACK_PROFILE :	BYTE
NWK_EXTENDED_PAN_ID :	QWORD
NWK_USE_MULTICAST :	BYTE
NWK_IS_CONCENTRATOR :	BYTE
NWK_UNIQUE_ADDR :	BYTE
NWK_SECURITY_LEVEL :	BYTE
NWK_ACTIVEKEY_SEQNUMBER :	BYTE
NWK_ALLFRESH :	BYTE
NWK_SECURE_ALLFRAMES :	BYTE

The other values return UNSUPPORTED_ATTRIBUTE, else the return is SUCCESS

3.4.2.2 nlmePermitJoiningRequest

Function	BYTE nlmePermitJoiningRequest(BYTE PermitDuration);		
Available for :	✔ **Coordinator**	✔ **Router**	✘ **End Device**

Header file	ZOneNwkReq.h	
C file	-	**Value**
Arguments	BYTE PermitDuration	Duration of permission or 0xFF
Return	BYTE	Status

Sets the permission for the device to accept join requests from other devices.

The type of permission is defined by the value of *PermitDuration*

0x00 : No permission

0x01 – 0xFE : Permission for *PermitDuration* seconds

0xFF : Allow associations without time limit

3.5 Application Support Layer

3.5.1 Principle

The user application need only a few access to the Application Support layer, and mainly for management purpose.

The Application Support layer access is thus limited to the functions and defines declared in ZOneApsReq.h and ZOneTables.h

The accesses to the APS provided in the Zone Stack deal mainly with the reading and writing of the information base.

The functions defined in ZOneTables.h complete the standard APS functions with a set of tables checking procedures for the Binding, Address and Group tables.

3.5.2 ZOneApsReq.h

The functions defined in ZOneNwkReq.h are not modifiable by the user.

3.5.2.1 apsmeBindRequest

Function	BYTE apsmeBindRequest(QWORD SrcAddress, BYTE SrcEndPoint, WORD ClusterId, BYTE DestAddrMode, ADDRESS DstAddress, BYTE DstEndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneApsReq.h	
C file	-	**Value**
Arguments	QWORD SrcAddress	Unused. Device's address
	BYTE SrcEndPoint	0x01 - 0xF0
	WORD ClusterId	0x0000 – 0xFFFF
	BYTE DestAddrMode	0x01 – 0x03
	ADDRESS DstAddress	Group, short or extended address
	BYTE DstEndPoint	0x01 – 0xF0
Return	BYTE	Status

Sets a binding from the device.

A message sent from the device's EndPoint *SrcEndPoint* and containing information identified by *ClusterId* will be addressed to the EndPoint *DstEndPoint* of the module(s) *DstAddress*.

The destination address can be of different types defined in ***ZOneApsReq.h*** :

APS_16BIT_GROUP_ADDR (0x01): Group address, 16 bits, sent as broadcast.

APS_16BIT_ADDR (0x02): Single device short address, sent unicast

APS_64BIT_ADDR (0x03): Single device extended IEEE address, sent unicast

The binding using an extended address can be defined even if the correspondence extended<->short addresses isn't defined in the Address Table. The extended address is converted to short only when a frame is handled to the APS data service access point.

Returns SUCCESS if the operation succeeded, or the failure cause otherwise (APS_ILLEGAL_REQUEST, APS_TABLE_FULL)

3.5.2.2 apsmeUnbindRequest

Function	BYTE apsmeUnbindRequest (QWORD SrcAddress, BYTE SrcEndPoint, WORD ClusterId, BYTE DestAddrMode, ADDRESS DstAddress, BYTE DstEndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneApsReq.h	
C file	-	**Value**
Arguments	QWORD SrcAddress	Unused. Device's address
	BYTE SrcEndPoint	0x01 - 0xF0
	WORD ClusterId	0x0000 – 0xFFFF
	BYTE DestAddrMode	0x01 – 0x03
	ADDRESS DstAddress	Group, short or extended address
	BYTE DstEndPoint	0x01 – 0xF0
Return	BYTE	Status

Removes an existing binding from the device.

Returns SUCCESS if the operation succeeded, or the failure cause otherwise (APS_ILLEGAL_REQUEST, APS_INV_BINDING)

3.5.2.3 apsmeGetRequest

Function	BYTE apsmeGetRequest(APS_IB_ATTR aibAttribute, void *paibAttributeValue);

Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneApsReq.h	
C file	-	Value
Arguments	APS_IB_ATTR aibAttribute void *paibAttributeValue	APSIB attribute idendification Pointer to the destination memory address
Return	BYTE	Status

Gets the value of a Application Support Information Base attribute.

The identifiers of the APSIB are defined in the enumeration APS_IB_ATTR in ZOneApsReq.h.

The size of the attribute value written at *paibAttributeValue* memory address is variable.

APS_DESIGNATED_COORDINATOR : BYTE
APS_CHANNEL_MASK : DWORD (4 Bytes)
APS_USE_EXTENDED_PAN_ID : QWORD (8 Bytes)
APS_USE_INSECURE_JOIN : BOOL
APS_INTERFRAME_DELAY : BYTE
APS_CHANNEL_TIMER : BYTE
APS_LAST_CHANNEL_ENERGY : BYTE
APS_LAST_CHANNEL_FAILURE_RATE : BYTE

Returns SUCCESS if the operation succeeded, or the failure cause otherwise (APS_UNSUPPORTED_ATTRIBUTE)

3.5.2.4 apsmeSetRequest

Function	BYTE apsmeSetRequest(APS_IB_ATTR aibAttribute, void *paibAttributeValue);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneApsReq.h	
C file	-	Value
Arguments	APS_IB_ATTR aibAttribute void *paibAttributeValue	APSIB attribute idendification Pointer to the source memory address
Return	BYTE	Status

Sets the value of a Application Support Information Base attribute.

The identifiers of the APSIB are defined in the enumeration APS_IB_ATTR in ZOneApsReq.h.

The size of the attribute value copied from *paibAttributeValue* memory address is variable.

APS_DESIGNATED_COORDINATOR :	BYTE
APS_CHANNEL_MASK :	DWORD (4 Bytes)
APS_USE_EXTENDED_PAN_ID :	QWORD (8 Bytes)
APS_USE_INSECURE_JOIN :	BOOL
APS_INTERFRAME_DELAY :	BYTE
APS_CHANNEL_TIMER :	BYTE
APS_LAST_CHANNEL_ENERGY :	BYTE
APS_LAST_CHANNEL_FAILURE_RATE :	BYTE

Returns SUCCESS if the operation succeeded, or the failure cause otherwise (APS_UNSUPPORTED_ATTRIBUTE, APS_INVALID_PARAM)

3.5.2.5 apsmeAddGroupRequest

Function	BYTE apsmeAddGroupRequest(WORD GroupAddress, BYTE EndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneApsReq.h	
C file	-	**Value**
Arguments	WORD GroupAddress BYTE EndPoint	0x0000 – 0xFFFF 0x01 - 0xF0
Return	BYTE	Status

Defines the endpoint *EndPoint* as being part of the group *GroupAddress*. A frame arriving to this group address will be directed to all endpoints listed in this group.

This is done by calling ***afwDataIndication*** as many time as needed with a different destination endpoint each.

3.5.2.6 apsmeRemoveGroupRequest

Function	BYTE apsmeRemoveGroupRequest(WORD GroupAddress, BYTE EndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneApsReq.h	
C file	-	**Value**
Arguments	WORD GroupAddress	0x0000 – 0xFFFF

	BYTE EndPoint	0x01 - 0xF0
Return	BYTE	Status

Removes the endpoint *EndPoint* from the group *GroupAddress*. If it was the last endpoint of the group, the group itself is removed from the group table

3.5.2.7 apsmeRemoveAllGroupsRequest

Function	BYTE apsmeRemoveAllGroupsRequest(BYTE EndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneApsReq.h	
C file	-	**Value**
Arguments	BYTE EndPoint	0x01 - 0xF0
Return	BYTE	Status

Removes the endpoint *EndPoint* from all the groups. If it was the last endpoint of a group, the group itself is removed from the group table.

3.5.3 ZOneTables.h

The functions defined in ZOneTables.h are not modifiable by the user.

3.5.3.1 tableBindingCount

Function	BYTE tableBindingCount(BYTE SrcEndPoint, WORD ClusterId);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTable.h	
C file	-	**Value**
Arguments	BYTE SrcEndPoint WORD ClusterId	0x01 – 0xF0 : Source Endpoint Cluster Identification
Return	BYTE	Number of bindings from *SrcEndPoint* to transmit *ClusterId* value

Counts the number of bindings from *SrcEndPoint* end point and concerning the value of *ClusterId* present in the Bindings Table.

3.5.3.2 tableBindingFindFirst

Function	TABLE_BINDING_FIND_RETURN tableBindingFindFirst(BYTE SrcEndPoint, WORD ClusterId);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTable.h	
C file	-	**Value**
Arguments	BYTE SrcEndPoint WORD ClusterId	Attribute to read Pointer to result storage address
Return	TABLE_BINDING_FIND_RETURN	Status

Search the Bindings Table for the first binding from *SrcEndPoint* end point and concerning the value of *ClusterId.* Returns all the binding's parameters in a structure :

```
typedef struct {
   BYTE  SrcEndPoint;
   WORD  ClusterId;
   ADDRESS DstAddress;
   BYTE  DstEndPoint;
} TABLE_BINDING_FIND_RETURN;
```

If no binding is found, the SrcEndPoint value in the returned structure will be set to 0xFF.

The following bindings will be obtained with the *tableBindingFindNext* function

3.5.3.3 tableBindingFindNext

Function	TABLE_BINDING_FIND_RETURN tableBindingFindNext(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTable.h	
C file	-	**Value**
Arguments	-	-
Return	TABLE_BINDING_FIND_RETURN	Status

Search the Bindings Table for the next binding from *SrcEndPoint* end point and concerning the value of *ClusterId. tableBindingFindFirst* should have been called before *tableBindingFindNext.* Returns all the binding's parameters in a structure :


```
typedef struct {
   BYTE  SrcEndPoint;
   WORD  ClusterId;
   ADDRESS DstAddress;
   BYTE  DstEndPoint;
} TABLE_BINDING_FIND_RETURN;
```

If no binding is found, the SrcEndPoint value in the returned structure will be set to 0xFF

3.5.3.4 tableGroupExist

Function	TABLE_GROUP_EXIST_RETURN tableGroupExist(WORD GroupAddress)		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTable.h	
C file	-	**Value**
Arguments	WORD GroupAddress	-
Return	TABLE_GROUP_EXIST_RETURN	Status

Checks if a group with the *GroupAddress* address exists in the Group Table.

Returns all the groups parameters in a structure, including the list of all linked EndPoints :

```
typedef struct {
   BYTE  EPCount;
   BYTE  EndPoint[TABLE_GROUP_MAX_EP_ENTRIES];
} TABLE_GROUP_EXIST_RETURN;
```

If no group is found, the EPCount value in the returned structure will be 0xFF

3.5.3.5 tableAddressShortFind

Function	WORD tableAddressShortFind(QWORD ExtendedAddress);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTable.h	
C file	-	**Value**
Arguments	QWORD ExtendedAddress	IEEE address of a module
Return	WORD	Short address if found, else 0xFFFF

Returns the short address of a device known only by its IEEE MAC address.

If the address couple isn't present in the Address Table, the function returns 0xFFFF.

3.5.3.6 tableAddressIEEEFind

Function	QWORD * tableAddressIEEEFind(WORD ShortAddress);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneTable.h	
C file	-	**Value**
Arguments	WORD ShortAddress	Short address of a module
Return	QWORD *	Pointer to the IEEE address if found, else NULL

Returns a pointer to the the IEEE MAC address of a device known only by its short address.

If the address couple isn't present in the Address Table, the function returns NULL

3.6 Application Framework Access

3.6.1 Principle

This will be the main access point from the Application to the Stack, on a functional point of view.

The *ZOneAfwReq.h* functions regroup the Network building functions (*afwStartNetwork*, *afwAssociation*, RFD Timer) and all the tools to build and send a data frame.

The *ZOneAfwRsp.h* contains the information sent by the Stack to the Application when a frame arrives (*afwdeDataIndication*) or has been requested to be sent (*afwdeDataconfirm*).

3.6.2 ZOneAfwReq.h

The functions defined in ZOneAfwReq.h are not modifiable by the user.

3.6.2.1 afwStartNetwork

Function	BYTE afwStartNetwork(void);		
Available for :	✔ **Coordinator**	✘ **Router**	✘ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	-	-
Return	BYTE	Status

Launches the network creation by the coordinator.

Returns the status of the operation as SUCCESS if the network is started, or AFW_ERROR_ACQ_COORD if either a network is already started or if the channels scan didn't detect any suitable channel.

3.6.2.2 afwAssociation

Function	BYTE afwAssociation(void);		
Available for :	✗ **Coordinator**	✓ **Router**	✓ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	-	-
Return	BYTE	Status

Try to associate the module to an existing network. If an extended PAN Id has been defined, only a module from this network can be selected as parent.

Returns the status of the operation as SUCCESS if the network is started, or AFW_ERROR_ACQ_ROUTER/ AFW_ERROR_ACQ_DEVICE if the module is already associated or AFW_ERROR_SCANNING if the channels scan didn't detect any suitable network.

3.6.2.3 afwStartRFDTimer

Function	void afwStartRFDTimer(void);		
Available for :	✗ **Coordinator**	✗ **Router**	✓ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	-	-
Return	BYTE	Status

This function is used for the Rx Off when Idle devices.

These devices are in Standby most of the time, with a settable wakeup interval which can be quite long. However, some actions need an answer after a delay, and the device needs to poll its parent to get it.

So a polling timer is defined and is launched by this function to send a data request each 2 seconds without going back to sleep. *afwCheckRFDTimer* must be called to check if the timer triggered and to poll the parent.

These functions are used in the sample application when an End Device Bind Request is sent, to wait for the answer of the coordinator.

3.6.2.4 afwCheckRFDTimer

Function	void afwCheckRFDTimer(void);		
Available for :	✗ **Coordinator**	✗ **Router**	✓ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	-	-
Return	BYTE	Status

Checks the RFD timer and sends a DataRequest to the parent if triggered. It restarts the timer for 2 seconds.

3.6.2.5 afwHeader

Function	BYTE afwHeader(BYTE DstAddrMode, ADDRESS aAddrAddressDest, WORD aWoGroupAddress, BYTE aByEPDest, BYTE aByEPSrc, WORD aByClusterID, WORD ProfileId, BYTE aTxOption, BYTE aRadius);		
Available for :	✓ **Coordinator**	✓ **Router**	✓ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	BYTE DstAddrMode	Addressing mode
	ADDRESS aAddrAddressDest	Destination Address
	WORD aWoGroupAddress	Group Address
	BYTE aByEPDest	Destination Endpoint

	BYTE aByEPSrc	Source Endpoint
	WORD aByClusterID	Cluster Identification
	WORD ProfileId	Profile Identification
	BYTE aTxOption	Transmission parameters
	BYTE aRadius	Max frame hops
Return	BYTE	Frame Handle

This function is used to create a new data packet and fill its header with the necessary values for transmission.

The destination address fields are filled differently in function of the value of *DstAddrMode*.

The possible values for *DstAddrMode* are defined in ***ZOneAfwReq.h***:

AFW_ADDR_NOT_PRESENT 0x00

The *aAddrAddressDest* and *aWoGroupAddress* fields are unused. The sending function will search the binding table to send one frame to each destination address/endpoint found for this endpoint/cluster/profile combination

AFW_16BIT_GROUP_ADDR 0x01

The *aAddrAddressDest* field is unused. The *aWoGroupAddress* field must be set to the destination group address.

AFW_16BIT_ADDR 0x02

The *aWoGroupAddress* field is unused. The *aAddrAddressDest* field must be set to the destination short address.

AFW_64BIT_ADDR 0x03

The *aWoGroupAddress* field is unused. The *aAddrAddressDest* field must be set to the destination IEEE address. The sending process will check if a short address exists in the Address Table for this 64 bits address, and return an error if not. The *afwHeader* function doesn't check the existence of the short address.

The different transmission options are defined in ***ZOneAfwReq.h***:

AFW_TX_OPT_SECURITY_ENABLE 0x01

Enables the security for this frame

AFW_TX_OPT_ACK_REQ 0x04

Requests an acknowledge from the destination. Not possible with group addressing.

AFW_TX_OPT_FRAGMENTATION 0x08

Allows fragmentation for this frame if the size is more than one packet can handle.

The Radius field defines the maximum number of hops the frame will do before reaching the destination. If not specified (0) the default will be set to twice the maximum depth of the network.

The function returns the handle used afterward to add data to the frame, send it, or to cancel the sending if needed.

3.6.2.6 afwAddElementToTransaction

Function	BOOL afwAddElementToTransaction(BYTE aPacketHandle, WORD ElementLength, BYTE * pElement, BOOL Invert);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	BYTE aPacketHandle WORD ElementLength BYTE * pElement BOOL Invert	Frame identifier Size of the added element Memory address of the element TRUE or FALSE
Return	BYTE	Status

Adds an element of size *ElementLength* copied from the memory address *pElement* to the frame identified by *aPacketHandle*. If the Invert parameter is set to TRUE, the byte order of the element is inverted in the frame.

The function returns FALSE if the element's length is longer than the space left in the frame, TRUE otherwise.

3.6.2.7 afwReserveBytes

Function	BOOL afwReserveBytes(BYTE aPacketHandle, WORD ReservedQuantity);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	BYTE aPacketHandle WORD ReservedQuantity	Frame identifier Size of the reserved space
Return	BOOL	TRUE if successful, else FALSE

Reserves a number *ReservedQuantity* of bytes in the frame identified by *aPacketHandle*.

Allows continuing to add elements and to write a value here afterward.

It is used by the sample application to set the frame length at the beginning of the frame, but the value is written at the end, when this length is known.

Can be called only once per frame.

The function returns FALSE if the element's length is longer than the space left in the frame, TRUE otherwise.

3.6.2.8 afwAddReservedElementToTransaction

Function	void afwAddReservedElementToTransaction(BYTE aPacketHandle, WORD ElementLength, BYTE * pElement, BOOL Invert);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	BYTE aPacketHandle WORD ElementLength BYTE * pElement BOOL Invert	Frame identifier Size of the added element Memory address of the element TRUE or FALSE
Return	-	-

Same action than *afwAddElementToTransaction*, but the element is copied in the previously reserved space in the frame.

3.6.2.9 afwHandleRelease

Function	void afwHandleRelease(BYTE aPacketRelease);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneAfwReq.h	
C file	-	**Value**
Arguments	BYTE aPacketRelease	Frame identifier
Return	-	-

Releases a frame without sending it.

3.6.2.10 afwdeDataRequest

Function	void afwdeDataRequest(BYTE aPacketHandle);

Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneAfwReq.h	
C file	-	Value
Arguments	BYTE aPacketHandle	Frame identifier
Return	-	-

Starts the sending process for the identified frame.

The result will be returned to the appliclation through the call of *afwDataConfirm*.

3.6.2.11 afwGetMaxPayloadSize

Function	BYTE afwGetMaxPayloadSize(void);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneAfwReq.h	
C file	-	Value
Arguments	-	-
Return	BYTE	Maximum payload size

Returns the maximum allowed size for a packet in function of the security parameters of the stack. This is the maximum size without fragmentation.

3.6.2.12 afwForceRouteDiscovery

Function	void afwForceRouteDiscovery (WORD DestAddr, BYTE Radius);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneAfwReq.h	
C file	-	Value
Arguments	WORD DestAddr BYTE Radius	Address to route to Max hops to destination
Return	-	-

Forces a route discovery to address DestAddr with a maximum number of hops of Radius.

This function is used only for tests purpose and should not be used in application.

3.6.3 ZOneAfwRsp.h / ZOneAfwRsp.c

The functions defined in ZOneAfwRsp.h are modifiable by the user and can be found in ***ZOneAfwRsp.c***

3.6.3.1 afwdeDataConfirm

Function	void afwdeDataConfirm(BYTE DstAddrMode, ADDRESS destAddr, BYTE DstEndPoint, BYTE SrcEndPoint, BYTE status);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneAfwRsp.h	
C file	ZOneAfwRsp.c	**Value**
Arguments	BYTE DstAddrMode ADDRESS destAddr BYTE DstEndPoint BYTE SrcEndPoint BYTE status	Addressing mode Destination Address Destination Endpoint Source Endpoint
Return	-	-

Gives the status of the frame sent to the endpoint *DstEndPoint* of the module *destAddr* from the endpoint *SrcEndPoint*.

This status can be :

AFW_SUCCESS 0x00
Frame sent and acknowledge received if asked for in the Tx Options

AFW_NO_ACK 0xA7
Frame sent, but acknowledge was asked for this frame and not received from destination.

AFW_NO_BOUND_DEVICE 0xA8
Frame sent with the DestAddressMode set to AFW_ADDR_NOT_PRESENT, but no binding was found in the bindings table

AFW_NO_SHORT_ADDRESS 0xA9
Frame sent with the DestAddressMode set to AFW_64BIT_ADDR, but no corresponding short address was found in the address table.

AFW_SECURITY_FAIL 0xAD
Encryption was asked for this frame, but an error occurred in the security process

3.6.3.2 afwdeDataIndication

Function	void afwdeDataIndication(BYTE aByDstAddrMode, ADDRESS aDstAddr, BYTE aByDstEndPoint, BYTE aBySrcAddrMode, ADDRESS aSrcAddr, BYTE aBySrcEndPoint, WORD aWoProfileId, WORD aWoClusterId, WORD aWoasduLength, BYTE *aByasdu, BYTE aBySecurity);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneAfwRsp.h	
C file	ZOneAfwRsp.c	**Value**
Arguments	BYTE aByDstAddrMode	0x01-0x03 (Group, 16b, 64b)
	ADDRESS aDstAddr	Destination address
	BYTE aByDstEndPoint	Destination EndPoint
	BYTE aBySrcAddrMode	0x02-0x03 (16b, 64b)
	ADDRESS aSrcAddr	Source address
	BYTE aBySrcEndPoint	Source EndPoint
	WORD aWoProfileId	Profile Id
	WORD aWoClusterId	Cluster Id
	WORD aWoasduLength	Payload length
	BYTE *aByasdu	Payload memory address
	BYTE aBySecurity	TRUE - FALSE
Return	-	-

Indicates that a frame has been received for the module.

The *aByDstAddrMode* indicates if the frame was group addressed, and if not the format 16 or 64 bits of the *aDstAddr*.

The *aBySrcAddrMode* indicates the format 16 or 64 bits of the *aSrcAddr* parameter.

The *aSrcAddr* Source address and *aBySrcEndPoint* Source EndPoint parameters can be used by application to send an answer or to identify the sender.

The *aByDstEndPoint* destination EndPoint, the *aWoProfileId* and *aWoClusterId* identify the exact destination in the module.

aWoasduLength gives the size of the frame's payload stores in memory at address *aByasdu*

aBySecurity indicates if the frame was secured or not.

3.7 Security Services Access

3.7.1 Principle

The Security Service Access let you manage the way security is handled in your network.

It allows to turn global security on or off, and to set the security keys at different levels.

3.7.2 ZOneSecurity.h

The functions defined in ZOneSecurity.h are not modifiable by the user.

3.7.2.1 setUseSecurity

Function	void setUseSecurity(BOOL aBoolUse);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	BOOL aBoolUse	TRUE - FALSE
Return	-	-

Sets the global network security on or off. If *aBoolUse* is set to TRUE, all frames will be secured at Network level

3.7.2.2 getUseSecurity

Function	BOOL getUseSecurity(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	-	-
Return	BOOL	TRUE - FALSE

Returns the global network security level

3.7.2.3 setPreconfiguredNwkKey

Function	void setPreconfiguredNwkKey(BOOL aBoolPreconfiguredNwkKey);		
Available for :	✘ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	BOOL aBoolPreconfiguredNwkKey	TRUE - FALSE
Return	-	-

This function must be called before association to set the way the module will get its Network Key.

If *aBoolPreconfiguredNwkKey* is set to TRUE, it will use the key defined using *setNwkKey*.

If it is set to FALSE, the module will obtain its Network Key through the security process, but the key will be sent once on the radio without encryption.

3.7.2.4 getPreconfiguredNwkKey

Function	BOOL getPreconfiguredNwkKey(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	-	-
Return	BOOL	TRUE - FALSE

Returns the module's way of getting the network key.

3.7.2.5 setNwkKey

Function	void setNwkKey(BYTE *aByNwkKey);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	BYTE* aByNwkKey	Points to a 16 bytes value
Return	-	-

Sets the global network security key to the value pointed by aByNwkKey. If the use of a network key is activated (*setUseSecurity(TRUE)*) all frames will be encrypted at network level with this key. Of course, a module using a wrong key will not be able to communicate with the network.

If the module is set to use preconfigured key at startup, it will try to associate using this key instead of getting a key over the air.

3.7.2.6 setPreconfiguredLinkKey

Function	void setPreconfiguredLinkKey(BOOL aBoolPreconfiguredLinkKey);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	BOOL aBoolUse	TRUE - FALSE
Return	-	-

Sets the link security use on or off. If *aBoolUse* is set to TRUE, the module will use a specific link key to encrypt the frames if a key exists for the destination.

This allows specific links to be set between nodes where the other nodes of the network can decrypt enough to route the message, but can't read the payload.

If preconfigured link keys are not allowed and security is asked for in a message, the module will try to get a Link Key from the Trust Center (Coordinator), but this link key will only be network encrypted when transmitted to the modules.

3.7.2.7 getPreconfiguredLinkKey

Function	BOOL getPreconfiguredLinkKey(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	BOOL aBoolUse	TRUE - FALSE
Return	-	-

Returns TRUE if the security status of the module allows it to use preconfigured link keys.

3.7.2.8 setLinkKey

Function	BOOL setLinkKey(QWORD *aIEEEAddress, BYTE *aByLinkKey);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneSecurity.h	
C file	-	**Value**
Arguments	QWORD *aIEEEAddress BYTE *aByLinkKey	Points to Destination IEEE address Points to 16 bytes Encryption key
Return	BOOL	TRUE - FALSE

Defines a preconfigured link key to a given address.

3.7.2.9 apsmeRequestKeyRequest

Function	void apsmeRequestKeyRequest(ADDRESS aAddDestAddress, BYTE aByKeyType, ADDRESS aAddPartnerAddress);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneApsReq.h	
C file	-	**Value**
Arguments	ADDRESS aAddDestAddress BYTE aByKeyType ADDRESS aAddPartnerAddress	The Extended address of its device. The Key Type: 0x01 : network Key 0x02 : application key If aByKeyType indicates the application key, this parameter shall indicate the Extended address of the partner for the link key
Return	BYTE	Status

Requests the network key shared by the entire network or an application key shared between two devices.

3.7.2.10 apsmeNetworkKeyUpdate

Function	void apsmeNetworkKeyUpdate(BYTE *aKey);		
Available for :	✔ **Coordinator**	✘ **Router**	✘ **End Device**

Header file	ZOneApsReq.h	
C file	-	**Value**
Arguments	BYTE *aKey	The new shared network key on 16 bytes.
Return	BYTE	Status

Updates the shared network key of all the devices of the entire network.

3.8 ZigBee Device Object (ZDO) Network Management

3.8.1 Principle

The ZigBee Device Object gives access to the management of the network. The services discovery, the network mapping and the remote definition of keys, bindings and other parameters are defined in the ZDO.

This allows the management of the network from any entry point.

3.8.2 ZOneZdoReq.h

The functions defined in ZOneZdoReq.h are not modifiable by the user.

They define the way the device will ask data from the network. The responses or requests coming from other devices are found in ZOneZdoRsp.h

3.8.2.1 zdoNwkAddrRequest

Function	void zdoNwkAddrRequest(ADDRESS aIEEEAddress, BYTE aRequestType, BYTE aStartIndex);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	ADDRESS aIEEEAddress BYTE aRequestType BYTE aStartIndex	Destination Extended Address 0x00 Single device, 0x01 Extended resp. Starting index in the list
Return	-	-

Requests a Network Address from a device identified by its IEEE Address.

The sender can request either a single device address (*aRequestType* = 0x00) or that the destination device joins the list of its children (*aRequestType* = 0x01) If the list is too long for the frame payload, the sender can specify the index of the first child address to join.

3.8.2.2 zdoIEEEAddrRequest

Function	void zdoIEEEAddrRequest(WORD aNWKAddrOfInterest, BYTE aRequestType, BYTE aStartIndex);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest BYTE aRequestType BYTE aStartIndex	Destination Network Address 0x00 Single device, 0x01 Extended resp. Starting index in the list
Return	-	-

Requests an Extended Address from a device identified by its Network Address.

The sender can request either a single device address (*aRequestType* = 0x00) or that the destination device joins the list of its children (*aRequestType* = 0x01) If the list is too long for the frame payload, the sender can specify the index of the first child address to join.

3.8.2.3 zdoNodeDescRequest

Function	void zdoNodeDescRequest(WORD aNWKAddrOfInterest);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest	Destination Network Address
Return	-	-

Asks the device identified by its Network Address to send back its Node Descriptor (See description in chapter 2.3.2.3. in the ZigBee 2007 specifications)

The request result will return as a call to *zdoNodeDescRsp*.

3.8.2.4 zdoPowerDescRequest

Function	void zdoPowerDescRequest(WORD aNWKAddrOfInterest);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest	Destination Network Address
Return	-	-

Asks the device identified by its Network Address to send back its Power Descriptor (See description in chapter 2.3.2.4. in the ZigBee 2007 specifications)

The request result will return as a call to *zdoPowerDescRsp*.

3.8.2.5 zdoSimpleDescRequest

Function	void zdoSimpleDescRequest(WORD aNWKAddrOfInterest, BYTE aEndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest BYTE aEndPoint	Destination Network Address EndPoint number
Return	-	-

Asks the device identified by its Network Address to send back its Simple Descriptor for the specified End Point (See description in chapter 2.3.2.5. in the ZigBee 2007 specifications)

The request result will return as a call to *zdoSimpleDescRsp*.

3.8.2.6 zdoActiveEPRequest

Function	void zdoActiveEPRequest(WORD aNWKAddrOfInterest);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest	Destination Network Address
Return	-	-

Asks the device identified by its Network Address to send back the list of its active end points.

The request result will return as a call to *zdoActiveEPRsp*.

3.8.2.7 zdoMatchDescRequest

Function	void zdoMatchDescRequest(WORD aNWKAddrOfInterest, WORD aProfileID, BYTE aNumInCluster, BYTE *apInClusterList, BYTE aNumOutCluster, BYTE *apOutClusterList);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest	Destination Network Address
	WORD aProfileID	Profile identifier
	BYTE aNumInCluster	Number of in clusters in list
	BYTE *apInClusterList	In Clusters List (Little Endian)
	BYTE aNumOutCluster	Number of out clusters in list
	BYTE *apOutClusterList	Out Clusters List (Little Endian)
Return	-	-

Asks the device identified by its Network Address to check if one of its End Points match the Profile Id and at least one of its in clusters matches one of the list's out clusters, or one of its out clusters matches one of the list's in clusters. (see ZigBee 2007 specifications §2.4.4.1.7 for a more detailed description)

The request result will return as a call to *zdoMatchDescRsp*.

3.8.2.8 zdoUserDescRequest

Function	void zdoUserDescRequest(WORD aNWKAddrOfInterest);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest	Destination Network Address
Return	-	-

Asks the device identified by its Network Address to send back its User Descriptor.

The request result will return as a call to *zdoUserDescRsp*.

3.8.2.9 zdoUserDescSet

Function	void zdoUserDescSet(WORD aNWKAddrOfInterest, BYTE aLength, BYTE *aUserDescriptor);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest BYTE aLength BYTE *aUserDescriptor	Destination Network Address 1 – 16 User Descriptor memory address
Return	-	-

Sends a command to the device identified by *aNWKAddrOfInterest* to set its User Descriptor to a new value of size *aLength* and copied from *aUserDescriptor* memory address

3.8.2.10 zdoEndDeviceAnnonceRequest

Function	void zdoEndDeviceAnnonce(WORD aNwkAddr, ADDRESS aIEEEAddress, BYTE aCapability);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNwkAddr ADDRESS aIEEEAddress BYTE aCapability	Source Network Address Source Extended Address Source Capability information
Return	-	-

Sends a message containing the Network Address *aNwkAddr*, the Extended Address *aIEEEAddress* and the capability information *aCapability* of the sender to inform the other device that it joined or rejoined the network, and giving its two addresses for them to update their tables.

3.8.2.11 zdoMgmtPermitJoiningRequest

Function	void zdoMgmtPermitJoiningRequest(WORD aNWKAddrOfInterest, BYTE aPermitDuration, BYTE aTC_Significance);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest BYTE aPermitDuration BYTE aTC_Significance	Destination Network Address 0x00 – 0xFF : Time in seconds 0x00 (Non used)
Return	-	-

Used from a management device or a commissioning tool to order a device to accept association for a given *aPermitDuration* time, where 0x00 means No Association, 0xFF means Permanent permit, and all other values give the permission period length in seconds.

3.8.2.12 zdoMgmtNwkUpdateRequest

Function	void zdoMgmtNwkUpdateRequest(WORD aNWKAddrOfInterest, DWORD ScanChannels, BYTE ScanDuration, BYTE ScanCount, WORD NwkMgrAddress);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddrOfInterest DWORD ScanChannels, BYTE ScanDuration BYTE ScanCount WORD NwkMgrAddress	Destination Network Address See following table
Return	-	-

Sends a NwkUpdaterequest message, where the ScanDuration parameter gives the type of order sent to the network :

	Scan Duration	Scan Channels	Scan Count	NwkManager Address	Reply
Channel Change	0xFE	Active Channel	-	-	N
Channels Mask & Nwk Manager Change	0xFF	New Channel mask	-	Y	N
Scan Channels	0x00 – 0x05	Scan mask	Y	-	Y

(-) means that the parameter is not used in the frame and any value can be used when the function is called.

The reply to the Scan Channels command returns as a NetworkUpdateNotify message sent by the remote device

3.8.2.13 zdoMgmtNwkUpdateNotifySend

Function	void zdoMgmtNwkUpdateNotifySend(BYTE cPacketHandle, DWORD ScanChannels, BYTE ScanDuration, BYTE ScanCount);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	BYTE cPacketHandle DWORD ScanChannels BYTE ScanDuration BYTE ScanCount	Packet handle or NO_AFW_PACKET Scanned Channels mask (16 bits)
Return	-	-

Sends a Network Update Notify message to the Network Manager, either by using an already reserved packet (cPacketHandle), or by reserving a new one if NO_AFW_PACKET is used.

The frame will contain the result of an energy detection scanning done using the parameters passed in the function call :

- ScanChannels : Channel mask of 16 bits, bit0 is channel 11, bit 16 is channel 26. If the bit is set (1), the channel will be scanned
- ScanDuration : How long the channel will be scanned to see the maximum RSSI value. Each step doubles the time.

- ScanCount : How many scanning are done and added to the frame

Used when the device detects a problem or at reception of a Networtk Update Request

3.8.2.14 zdoMgmtLeaveRequest

Function	void zdoMgmtLeaveRequest(ADDRESS aIEEEAddress, BOOL abRemoveChildren, BOOL abRejoin);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	ADDRESS aIEEEAddress BOOL abRemoveChildren BOOL abRejoin	Destination Extended Address TRUE/FALSE TRUE/FALSE
Return	-	-

Sends a command to a remote device to leave the network.

If *abRemoveChildren* is TRUE, the device will send a Leave Request to its children before leaving, using the same value for the *abRejoin* parameter.

If *abRejoin* is TRUE, the device will try to rejoin the same network after leaving.

3.8.2.15 zdoMgmtSwitchKeyRequest

Not used for application purpose. Dedicated to tests and certification

3.8.2.16 zdoMgmtBindRequest

Function	void zdoMgmtBindRequest(WORD aNWKAddr, BYTE aIndex);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aNWKAddr BYTE aIndex	-
Return	-	-

Asks the remote device identified by *aNWKAddr* to send as many elements as possible from its Binding Table, starting at index *aIndex*. The answer is received as a call to *zdoMgmtBindRsp*

3.8.2.17 zdoBindingsRequest

Function	void zdoBindingsRequest(WORD aBindingType, BYTE aDestAddressMode, ADDRESS aDestAddress, QWORD aIEEESrcAddress, BYTE aSrcEndPoint, WORD aClusterID, BYTE aDstAddrMode, ADDRESS aDstAddress, BYTE aDstEndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aBindingType	ZDO_CLUSTER_BIND_REQ / ZDO_CLUSTER_UNBIND_REQ
	BYTE aDestAddressMode	APS_16BIT_ADDR (0x02): short address APS_64BIT_ADDR (0x02): Extended address
	ADDRESS aDestAddress	Destination Address
	QWORD aIEEESrcAddress	Binding Source Extended address
	BYTE aSrcEndPoint	Binding Source Endpoint
	WORD aClusterID	Binding Cluster Id
	BYTE aBindDstAddrMode	Binding destination mode
	ADDRESS aBindDstAddress	Binding destination Address
	BYTE aDstEndPoint	Binding destination Endpoint
Return	-	-

The *aBindingType* value defines if the command sets or removes a binding from the remote device identified by *aDestAddress*:

ZDO_CLUSTER_BIND_REQ defines a new binding

ZDO_CLUSTER_UNBIND_REQ removes a binding if it exists.

The *aDestAddress* destination address mode can be APS_16BIT_ADDR (0x02) or APS_64BIT_ADDR (0x03).

If a binding is requested, it defines that a message sent from the remote device's EndPoint *SrcEndPoint* and containing information identified by *ClusterId* will be addressed to the EndPoint *DstEndPoint* of the module(s) *DstAddress*.

The binding destination address can be of different types defined in ***ZOneApsReq.h*** :

APS_16BIT_GROUP_ADDR (0x01): Group address, 16 bits, sent as broadcast.

APS_16BIT_ADDR (0x02): Single device short address, sent unicast

APS_64BIT_ADDR (0x03): Single device extended IEEE address, sent unicast

The binding using an extended address can be defined even if the correspondence extended<->short addresses isn't defined in the Address Table. The extended address is converted to short only when a frame is handled to the APS data service access point.

If an unbinding is asked for, the remote device will check the existence of a binding with the sent parameters and remove it from its table.

The reply to the Binding Request command returns as a Bindings Response message sent by the remote device.

3.8.2.18 zdoEndDeviceBindRequest

Function	void zdoEndDeviceBindRequest(WORD aBindingTarget, QWORD aSrcIEEEAddress, BYTE aSrcEndPoint, WORD aProfileID, BYTE aNumInCluster, BYTE *apInClusterList, BYTE aNumOutCluster, BYTE *apOutClusterList);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoReq.h	
C file	-	**Value**
Arguments	WORD aBindingTarget	Binding's source short address
	QWORD aSrcIEEEAddress	Binding's source IEEE address
	BYTE aSrcEndPoint	Binding's Source Endpoint
	WORD aProfileID	Profile Identifier
	BYTE aNumInCluster	Number of in clusters in list
	BYTE *apInClusterList	In Clusters List (Little Endian)
	BYTE aNumOutCluster	Number of out clusters in list
	BYTE *apOutClusterList	Out Clusters List (Little Endian)
Return	-	-

This command sends a message to the coordinator containing all the data to effectuate a match descriptor and a binding.

If a second message is received within 20 seconds on the coordinator from another device (or from another endpoint of the same device), the Coordinator checks the apInClusterList against the apOutClusterList of each device, and search a match. If one

is found, the Coordinator sends an Unbind request to the device with the collected data. If an error is returned (No binding to unbind), it sends a message defining a binding for each match found, addressed to the out cluster's device address, else it continues to unbind the other matches found.

3.8.3 ZOneZdoRsp.h / ZOneZdoRsp.c

The functions defined in ZOneZdoRsp.h are modifiable by the user, and the code is found in ZOneZdoRsp.c

3.8.3.1 zdoNWKIEEEAddrRsp

Function	void zdoNWKIEEEAddrRsp(BYTE aByStatus, ADDRESS aAddIEEEAddrRemoteDev, WORD aWoNWKAddrRemoteDev, BYTE aByNumAssoDev, BYTE aByStartIndex, BYTE *aByListOfAssociated, WORD aWoClusterID);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus	
	ADDRESS aAddIEEEAddrRemoteDev	remote device Extended Address
	WORD aWoNWKAddrRemoteDev	remote device Short Address
	BYTE aByNumAssoDev	Number of device's childs in list
	BYTE aByStartIndex	Index of first device in list
	BYTE *aByListOfAssociated	List of child addresses
	WORD aWoClusterID	Network or IEEE request
Return	-	-

Response to a *zdoNWKIEEEAddrReq*.

The *aWoClusterID* shows the type of request (Short or Extended Address Request).

The remote device addresses are returned in *aAddIEEEAddrRemoteDev* and *aWoNWKAddrRemoteDev* fields, and if asked for in the request, the list of the short addresses of the remote device's children is added to the response.

3.8.3.2 zdoNodeDescRsp

Function	void zdoNodeDescRsp(BYTE aByStatus, WORD aWoNWKAddrOfInterest, BYTE *aByNodeDescriptor);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus WORD aWoNWKAddrOfInterest BYTE *aByNodeDescriptor	SUCCESS or error code Remote device address Node Descriptor
Return	-	-

Response to a zdoNodeDescReq.

The Node Descriptor is organized as follows :

Field Name	Length (Bits)
Logical type	3
Complex descriptor available	1
User descriptor available	1
Reserved	3
APS flags	3
Frequency band	5
MAC capability flags	8
Manufacturer code	16
Maximum buffer size	8
Maximum incoming transfer size	16
Server mask	16
Maximum outgoing transfer size	16
Descriptor capability field	8

See ZigBee Specifications §2.3.2.3 for more information

3.8.3.3 zdoPowerDescRsp

Function	void zdoPowerDescRsp(BYTE aByStatus, WORD aAddNWKAddrOfInterest, BYTE *aBybufPowerDescriptor);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus WORD aAddNWKAddrOfInterest BYTE *aBybufPowerDescriptor	-
Return	-	-

Response to a zdoPowerDescReq.

The Power Descriptor is organized as follows :

Field Name	Length (Bits)
Current power mode	4
Available power sources	4
Current power source	4
Current power source level	4

See ZigBee Specifications §2.3.2.4 for more information

3.8.3.4 zdoSimpleDescRsp

Function	void zdoSimpleDescRsp(BYTE aByStatus, WORD aWoNWKAddrOfInterest, BYTE aByLenghtSimpleDescriptor, BYTE *aBySimpleDescriptor);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus WORD aWoNWKAddrOfInterest BYTE aByLenghtSimpleDescriptor BYTE *aBySimpleDescriptor	-
Return	-	-

Response to a zdoSimpleDescReq.

The Node Descriptor is organized as follows :

Field Name	Length (Bits)
Endpoint	8
Application profile identifier	16
Application device identifier	16
Application device version	4
Reserved	4
Application input cluster count	8
Application input cluster list	16*i (where *i* is the value of the application input cluster count)
Application output cluster count	8
Application output cluster list	16*o (where *o* is the value of the application output cluster count)

See ZigBee Specifications §2.3.2.5 for more information

3.8.3.5 zdoActiveEPRsp

Function	void zdoActiveEPRsp(BYTE aByStatus, WORD aWoNWKAddrOfInterest, BYTE aByActiveEPcount, BYTE *aBybufActiveEpList);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus WORD aWoNWKAddrOfInterest BYTE aByActiveEPcount BYTE *aBybufActiveEpList	-
Return	-	-

Response to a *zdoActiveEPReq*.

Returns the list of active endpoints for device *aWoNWKAddrOfInterest*, allowing the targeted requests of simple descriptors.

3.8.3.6 zdoMatchDescRsp

Function	void zdoMatchDescRsp(BYTE aByStatus, WORD aWoNWKAddrOfInterest, BYTE aByMatchLength, BYTE *aByMatchList);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus WORD aWoNWKAddrOfInterest BYTE aByMatchLength BYTE *aByMatchList	SUCCESS or error code Remote device Length of match list List of matching endpoints
Return	-	-

Response to a *zdoMatchDescReq*.

The status can be SUCCESS if the information is joined, ZDP_NO_DESCRIPTOR if the device of interest is a sleeping device and its parent doesn't possess the simple descriptors or ZDP_DEVICE_NOT_FOUND if no answer was received.

3.8.3.7 zdoUserDescRsp

Function	void zdoUserDescRsp(BYTE aByStatus, WORD aWoNWKAddrOfInterest, BYTE aByLenghtUserDescriptor, BYTE *aByUserDescriptor);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus WORD aWoNWKAddrOfInterest BYTE aByLenghtUserDescriptor BYTE *aByUserDescriptor	
Return	-	-

Response to a *zdoUserDescReq*.

Returns the User Descriptor as a string on maximum 16 bytes

3.8.3.8 zdoEndDeviceAnnonceIndication

Function	void zdoEndDeviceAnnonceIndication(WORD aWoNWKAddr, ADDRESS aAddIEEEAddr, BYTE aCapability);

Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	Value
Arguments	WORD aWoNWKAddr ADDRESS aAddIEEEAddr BYTE aCapability	-
Return	-	-

Indicates the reception of an End Device Announce message, and gives the Network and extended address of the device newly joined or rejoined to the network. When this function is called, the tables have already been updated.

3.8.3.9 zdoUserDescConf

Function	void zdoUserDescConf(BYTE aByStatus, WORD aWoNWKAddrOfInterest);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	Value
Arguments	BYTE aByStatus WORD aWoNWKAddrOfInterest	SUCCESS or error code Remote device address
Return	-	-

Confirms the setting of the remote User Descriptor.

See ZigBee Specifications §2.4.4.1.11 for more information

3.8.3.10 zdoSystemDiscoveryRsp

Function	void zdoSystemDiscoveryRsp(BYTE aByStatus, WORD aWoServerMask);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus WORD aWoServerMask	-
Return	-	-

Response to a *zdoUserDescReq*.

Returns the status of the answer (always SUCCESS) and the Server Mask of the remote device

3.8.3.11 zdoMgmtLeaveRsp

Function	void zdoMgmtLeaveRsp(BYTE aByStatus);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus	-
Return	-	-

Returns the status of a zdoMgmtLeaveRequest.

See ZigBee Specifications §2.4.4.3.5 for more information

3.8.3.12 zdoMgmtBindRsp

Function	void zdoMgmtBindRsp(BYTE aByStatus, BYTE aByBindingTableEntries, BYTE aByStartIndex, BYTE aByBindingTableCount, BYTE *apByBindingTableList, BYTE aByBindingTableListLenght);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus BYTE aByBindingTableEntries BYTE aByStartIndex BYTE aByBindingTableListCount BYTE *apByBindingTableList BYTE aByBindingTableListLenght	SUCCESS or error code
Return	-	-

Returns the response to a zdoMgmtBindRequest as a list of Binding Table elements.

Name	Size (Bits)	Valid Range	Description
SrcAddr	64	A valid 64-bit IEEE address	The source IEEE address for the binding entry.
SrcEndpoint	8	0x01 - 0xf0	The source endpoint for the binding entry.
ClusterId	16	0x0000 - 0xffff	The identifier of the cluster on the source device that is bound to the destination device.
DstAddrMode	8	0x00 - 0xff	The addressing mode for the destination address. This field can take one of the non-reserved values from the following list: 0x00 = reserved 0x01 = 16-bit group address for DstAddr and DstEndpoint not present 0x02 = reserved 0x03 = 64-bit extended address for DstAddr and DstEndp present 0x04 – 0xff = reserved
DstAddr	16/64	As specified by the DstAddrMode field	The destination address for the binding entry.
DstEndpoint	0/8	0x01 - 0xf0, 0xff	This field shall be present only if the DstAddrMode field has a value of 0x03 and, if present, shall be the destination endpoint for the binding entry.

aByBindingTableEntries contains the number of entries in the remote binding table, and aByBindingTableListCount the number of entries sent in this message.

See ZigBee Specifications §2.4.4.3.4 for more information

3.8.3.13 zdoMgmtPermitJoiningRsp

Function	void zdoMgmtPermitJoiningRsp(BYTE aByStatus);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus	SUCCESS or ZDP_INV_REQUESTTYPE
Return	-	-

Returns the status of a zdoMgmtPermitJoiningReq.

ZDP_INV_REQUESTTYPE if the remote device is an End Device, or if the security setting doesn't allow it to execute the command. Else the status will be SUCCESS.

3.8.3.14 zdoMgmtNwkUpdateNotify

Function	void zdoMgmtNwkUpdateNotify(BYTE aByStatus, DWORD ScannedChannels, WORD TotalTx, WORD TxFailures, BYTE ScanListCount, BYTE* ScanList);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus	SUCCESS or error code
	DWORD ScannedChannels	Channels bit mask 1 ch=1 bit
	WORD TotalTx	Total number of transmissions
	WORD TxFailures	Number of transmission failures
	BYTE ScanListCount	Number of scans in list
	BYTE* ScanList	Energy levels table

Return	-	-

Received on the Network Manager, either as a response to a *zdoMgmtNwkUpdateRequest*, or spontaneously sent by a device when the level of errors reaches 25% of the frames sent, indicating a probability of noise on the used channel.

The managing application is then free to ask other energy scans in other parts of the network, and to change the current channel if needed.

The scan list contains the energy level on each scanned channel defined in the *ScannedChannels* parameter.

3.8.3.15 zdoBindingsRsp

Function	void zdoBindingsRsp(WORD aWoTypeBindings, BYTE aByStatus);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	WORD aWoTypeBindings BYTE aByStatus	ZDO_CLUSTER_BIND_REQ / ZDO_CLUSTER_UNBIND_REQ SUCCESS or error code
Return	-	-

Response to a *zdoBindingsRequest*.

The status can be SUCCESS, AFW_INV_BINDING if an unbind is requested for a non existing binding, or AFW_TABLE_FULL if a binding is requested and the remote table is full.

3.8.3.16 zdoEndDeviceBindRsp

Function	void zdoEndDeviceBindRsp(BYTE aByStatus);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus	SUCCESS or error code
Return	-	-

Response to a *zdoEndDeviceBindRequest*

aByStatus can return SUCCESS if a match has been found. In this case the binding requests from the coordinator should be received after if an out cluster was matched.

Else, the status can be ZDP_TIMEOUT if no second device has sent an End Device Bind Request in time, ZDP_NO_MATCH if no matching cluster was found, or ZDP_NOT_SUPPORTED if an End Device Binding process is already running in the coordinator.

3.8.3.17 zdoUnBindRequestIndication

Function	void zdoUnBindRequestIndication(BYTE aByStatus, QWORD SrcAddress, BYTE SrcEndPoint, WORD ClusterId, BYTE DestAddrMode, ADDRESS DstAddress, BYTE DstEndPoint);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus QWORD SrcAddress BYTE SrcEndPoint WORD ClusterId BYTE DestAddrMode ADDRESS DstAddress BYTE DstEndPoint	All these information are informative and reflect the request received by the device
Return	-	-

This function informs the management that the device received an unbind request. All the unbinding process has been done in the device before calling this function, so it is only informative and can be ignored. Can be used for user interface.

3.8.3.18 zdoBindRequestIndication

Function	void zdoBindRequestIndication(BYTE aByStatus, QWORD SrcAddress, BYTE SrcEndPoint, WORD ClusterId, BYTE DestAddrMode, ADDRESS DstAddress, BYTE DstEndPoint);

Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneZdoRsp.h	
C file	ZOneZdoRsp.c	**Value**
Arguments	BYTE aByStatus QWORD SrcAddress BYTE SrcEndPoint WORD ClusterId BYTE DestAddrMode ADDRESS DstAddress BYTE DstEndPoint	All these information are informative and reflect the request received by the device
Return	-	-

This function informs the management that the device received a bind request. All the unbinding process has been done in the device before calling this function, so it is only informative and can be ignored. Can be used for user interface.

3.9 Management interface

3.9.1 Principle

The ZOneMGTInterface and ZOneMGTISerialInterface provide a way to access the management functions from the ZDO and AFW layers from the outside through the serial link.

All these functions are given as example of how to use the management procedures, and can be modified to suit the user's interface.

3.9.2 ZOneMGTSerialInterface.h / ZOneMGTSerialInterface.c

The functions defined in ZOneMGTSerialInterface.h are modifiable by the user.

They define a way for the Management functions to send data on the serial link.

These serial data are formatted as follow :

Byte 0	Byte 1	Byte 2	...	Byte N
Length (N)	Command code	Data 1	...	Data N-1

These functions are called from the stack, and must be defined even if empty. If you don't need them, just don't define the MGT_INTERFACE compilation symbol.

3.9.2.1 Init_MGT_Serial

Function	void Init_MGT_Serial(BYTE Command);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneMGTSerialInterface.h	
C file	ZOneMGTSerialInterface.c	**Value**
Arguments	BYTE Command	
Return	-	-

Reserves the Length field and starts to fill the serial buffer with the command code *Command*.

Must be called at the beginning of each serial packet building.

3.9.2.2 Add_MGT_Serial

Function	void Add_MGT_Serial(WORD aLength, BYTE * aValue, BYTE aReverse);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneMGTSerialInterface.h	
C file	ZOneMGTSerialInterface.c	**Value**
Arguments	WORD aLength BYTE * aValue BYTE aReverse	
Return	-	-

Copies an element of size *aLength* from the memory address *aValue* to the serial buffer. It is added at the end of the current frame being built.

3.9.2.3 Send_MGT_CommandFrame

Function	void Send_MGT_CommandFrame(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneMGTSerialInterface.h

C file	ZOneMGTSerialInterface.c	Value
Arguments	-	-
Return	-	-

Gives the order to send the frame currently in the serial buffer.

3.9.3 ZOneMGTSerialInterface.h / ZOneMGTSerialInterface.c

The functions defined in ZOneMGTSerialInterface.h are modifiable by the user so he can modify the way the system reacts when they are called

These functions are called from the stack, and must be defined even if empty. If you don't need them, just don't define the MGT_INTERFACE compilation symbol.

3.9.3.1 zmiSerialFrameReception

Function	BYTE zmiSerialFrameReception(void);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneMGTInterface.h	
C file	ZOneMGTInterface.c	Value
Arguments	-	-
Return	BYTE	

Called by the main function when a serial frame is received and the module has been set in Control mode, either by hardware (PROG pin) or by software (+++ received).

Manages the different command frames in function of the command code received. Calls the corresponding ZDO or lower layer function after extracting the parameters values from the serial frame.

3.9.3.2 mgt_nlmeLeaveConfirm

Function	void mgt_nlmeLeaveConfirm(QWORD ExtendedAddress, BYTE Status);		
Available for :	✔ Coordinator	✔ Router	✔ End Device

Header file	ZOneMGTInterface.h	
C file	ZOneMGTInterface.c	Value
Arguments	QWORD ExtendedAddress	

	BYTE Status	
Return	-	-

Called by the stack when the device receives the answer to a Leave Request.

ExtendedAddress identify the leaving module, and *Status* is equal to SUCCESS if the module left the network, or any status value from the Network or MAC layers explaining the error

3.9.3.3 mgt_nlmeLeaveIndication

Function	void mgt_nlmeLeaveIndication(QWORD ExtendedAddress, BYTE Rejoin);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneMGTInterface.h	
C file	ZOneMGTInterface.c	**Value**
Arguments	QWORD ExtendedAddress BYTE Rejoin	Leaving device extended address TRUE - FALSE
Return	-	-

Indicates that the device identified by *ExtendedAddress* left the network. The *Rejoin* option shows if the device will attempt to rejoin the network at the end of the leave process.

3.9.3.4 mgt_nlmeJoinIndication

Function	void mgt_nlmeJoinIndication(WORD ShortAddress, QWORD ExtendedAddress, BYTE CapabilityInfo, BOOL SecureJoin);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneMGTInterface.h	
C file	ZOneMGTInterface.c	**Value**
Arguments	WORD ShortAddress QWORD ExtendedAddress BYTE CapabilityInfo BOOL SecureJoin	Child's network address Child's extended address Capability Info bitfield TRUE-FALSE : Joining secured ?
Return	-	-

Indicates the device identified by its IEEE Address *ExtendedAddress* joined the network on this device. Its Network Address is *ShortAddress* and it is characterized by the information in *CapabilityInfo*:

bits: 0	1	2	3	4-5	6	7
Alternate PAN coordinator	Device type	Power source	Receiver on when idle	Reserved	Security capability	Allocate address

3.9.3.5 mgt_nlmeJoinConfirm

Function	void mgt_nlmeJoinConfirm(WORD ShortAddress, WORD PanID, BOOL HaveNwkKey, BYTE status) ;		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneMGTInterface.h	
C file	ZOneMGTInterface.c	**Value**
Arguments	WORD ShortAddress WORD PanID BOOL HaveNwkKey BYTE status	
Return	-	-

Answers to a call to nlmeJoinRequest. If the *Status* is set to SUCCESS, the values of *ShortAddress* and *PanID* identifie the device and the network. If the device received the NetworkKey, *HaveNwkKey* is set to TRUE.

Else *Status* will indicate the cause of the failure.

3.9.3.6 mgt_apsmeTransportKeyIndication

Not used for application purpose. Dedicated to tests and certification

3.9.3.7 mgt_apsmeRemoveDeviceIndication

Not used for application purpose. Dedicated to tests and certification

3.9.3.8 mgt_apsmeUpdateDeviceIndication

Not used for application purpose. Dedicated to tests and certification

3.9.3.9 mgt_apsmeRequestKeyIndication

Not used for application purpose. Dedicated to tests and certification

3.9.3.10 mgt_apsmeSwitchKeyIndication

Not used for application purpose. Dedicated to tests and certification

3.9.3.11 mgt_epsFrameReception

Not used for application purpose. Dedicated to tests and certification

3.9.3.12 mgt_AFDirectConfirm

Not used for application purpose. Dedicated to tests and certification

3.9.3.13 mgt_AFGroupConfirm

Not used for application purpose. Dedicated to tests and certification

3.9.3.14 mgt_zdoChangePanId

Function	void mgt_zdoChangePanId(WORD * PanId);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneMGTInterface.h	
C file	ZOneMGTInterface.c	**Value**
Arguments	WORD * PanId	
Return	-	-

Called to set a new PAN Id in EEPROM. This function must be called as the user application doesn't have access to the reserved part of the EEPROM.

The manual setting of a PAN Id should be used only for tests purpose, as the IEEE 802.15.4 specifies that the PAN ID is randomly generated and the Zone Stack 2007 follows these specifications

3.10 ZCL functions

3.10.1 Principle

The ZOneZCLFondation provides some functions to build or separate the different fields of the ZCL header.

3.10.2 ZOneZCLFondation.h / ZOneZCLFondation.c

The functions defined in ZOneZCLFondation.c are modifiable by the user and are not validated.

3.10.2.1 zclCommand

Function	BYTE zclCommand(BYTE *Buffer);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZCLFondation.h	
C file	ZOneZCLFondation.c	**Value**
Arguments	BYTE *Buffer	Must have the asdu pointer
Return	BYTE	The zcl command

Called to know the zcl command starting from the asdu buffer.

3.10.2.2 zclTransactionSequenceNumber

Function	BYTE zclTransactionSequenceNumber(BYTE *Buffer);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZCLFondation.h	
C file	ZOneZCLFondation.c	**Value**
Arguments	BYTE *Buffer	Must have the asdu pointer
Return	BYTE	The zcl transaction sequence number

Called to know the zcl transaction sequence number starting from the asdu buffer.

3.10.2.3 zclRead_ReadAttributes

Function	WORD zclRead_ReadAttributes(BYTE *Buffer, BYTE aByIndex);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZCLFondation.h	
C file	ZOneZCLFondation.c	**Value**
Arguments	BYTE *Buffer BYTE aByIndex	Must have the asdu pointer Index of the Read Attributes to read
Return	WORD	The Read Attribute corresponding at index argument

Called to read the zcl Read attributes starting from the asdu buffer.

3.10.2.4 zclGeneralCommandReadResponse

Function	Void zclGeneralCommandReadResponse(BYTE cPacketHandle, WORD aWoAttributes, BYTE aByStatus, BYTE aByDataType, BYTE *aByData);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZCLFondation.h	
C file	ZOneZCLFondation.c	**Value**
Arguments	BYTE cPacketHandle WORD aWoAttributes BYTE aByStatus BYTE aByDataType BYTE *aByData	Handle of the packet Attributes of the Read Command Status of the Read Command Type of the Data Data
Return	-	-

Called to add in the packet the zcl Read Response command.

This function manages 3 kinds of Data:

ZCL_DATA_TYPE_LOGICAL_BOOLEAN	Boolean	1 Byte
ZCL_DATA_TYPE_GENERALDATA_8BIT	unsigned char - 8 Bits	1 Byte
ZCL_DATA_TYPE_LOGICAL_UNSIGNED_INT_16BIT	Unsigned int – 16 Bits	2 Bytes

3.10.2.5 zclGeneralCommandWriteResponse

Function	void zclGeneralCommandWriteResponse(BYTE cPacketHandle, WORD aWoAttributes, BYTE aByStatus);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZCLFondation.h	
C file	ZOneZCLFondation.c	**Value**
Arguments	BYTE cPacketHandle WORD aWoAttributes BYTE aByStatus	Handle of the packet Attributes of the Write Command Status of the Write Command
Return	-	-

Called to add in the packet the zcl Write Response command.

3.10.2.6 zclGeneralCommandReadAttributes

Function	void zclGeneralCommandReadAttributes(BYTE cPacketHandle, BYTE aByNmbArg, WORD aWoAttributeIdentifier1, ...);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOneZCLFondation.h	
C file	ZOneZCLFondation.c	**Value**
Arguments	BYTE cPacketHandle BYTE aByNmbArg WORD aWoAttributeIdentifier1 ...	Handle of the packet Number of following arguments First Attribute Identifier Next Attribute Identifier
Return	-	-

Called to add to the frame some attribute.

3.11 Main Functions

3.11.1 Principle

The Zone module provide the initialization functions, some utilities, and the main function of the application called at the beginning.

3.11.2 ZOne.h / ZOne.c

The functions defined in ZOne.c are modifiable by the user.

3.11.2.1 SetExtendedAddress

Function	Void SetExtendedAddress(BYTE * pAddressLo);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOne.h	
C file	-	**Value**
Arguments	BYTE * pAddressLo	Points to a 4 bytes table containing the low part of the MAC address for the device
Return	-	-

Debug stacks only !

Sets a value for the lower part of the Extended Address of the module. To avoid conflict with other manufacturers, the high part is always set to the One RF Technology identifier
0x00 0x15 0x4F 0x00

This function allows setting a different address to each module in debug mode. The default values will be :

- Coordinator : 0x00 0x00 0xBE 0xEF
- Router : 0x00 0x00 0xCA 0xFE
- End device : 0x00 0x00 0xAB 0xCD

3.11.2.2 ZOneInit

Function	void ZOneInit(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOne.h	
C file	ZOne.c	**Value**
Arguments	-	-
Return	-	-

Initializes the user part of the stack. All the internal initialization has been done before calling ZOneMain.

In Our sample application, this function sets the default values for the GPIO, and calls the EEPROM and serial init functions.

3.11.2.3 ZOneMain

Function	void ZOneMain(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOne.h	
C file	ZOne.c	**Value**
Arguments	-	-
Return	-	-

Function called by the stack to run the application. Equivalent to the main() function in c code.

It calls the initialization function (ZOneInit) and loops infinitely on the check of the serial or GPIO events flags. The radio events are managed directly from the stack and appropriate functions called.

This function puts also the device in Stand By mode if needed.

3.11.2.4 LaunchBootloader

Function	void LaunchBootloader(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOne.h	
C file	ZOne.c	**Value**
Arguments	-	-
Return	-	-

Puts the device in reflash mode. When called by this function, the device doesn't exit the Bootloader mode until reflashed or reset.

3.11.2.5 StackReset

Function	void StackReset(BOOL aBoolResetType);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOne.h	
C file	ZOne.c	**Value**

Arguments	BOOL aBoolResetType	ZONE_HARD_RESET / ZONE_SOFT_RESET
Return	-	-

If called as a Soft reset, initializes only the stack functional tables and values to default, as if the device never joined a network, but keeps the global parameters like channels mask or Extended PAN Id.

A Hard reset fully reinitializes the device as provided from factory

3.11.2.6 IsTaskDefined

Function	BOOL IsTaskDefined(void);		
Available for :	✔ **Coordinator**	✔ **Router**	✔ **End Device**

Header file	ZOne.h	
C file	ZOne.c	**Value**
Arguments	-	-
Return	BOOL	TRUE if a task is scheduled

Checks if a task is still scheduled. Useful to put the device in Stand By mode only when all the processes are stopped and no action need to be done anymore on this session

3.11.2.7 RFStartReceptionIndication, RFStopReceptionIndication, RFStartTransmissionIndication, RFStopTransmissionIndication

These functions are called directly from the MAC layer to indicates the starts and stops of transmissions and receptions.

Used for example to light LEDs showing activity on a USB dongle.

4 Document Change Log

Revision	Date	Changes
ISSUE#0	17/04/09	First Release