Instruction

Z-Wave 400 Series Developer's Kit v6.02.00 Contents

Document No.:	INS12035
Version:	2
Description:	Describes the contents and sample applications user guides of the Z-Wave 400 Series Developer's Kit v6.0x
Written By:	JFR
Date:	2012-05-25
Reviewed By:	CHL;BBR
Restrictions:	Partners Only

Approved by:

Date	CET	Initials	Name	Justification
2012-05-25	11:27:54	NTJ	Niels Thybo Johansen	

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20090906	JFR	ALL	Initial draft
2	20091229	VVI	5.1.1	Added description of Z-Wave programmer firmware source files.
3	20100414	JFR	3	Software components updated
4	20100423	JFR	3.2.4.11	Alternative external non-volatile memory chip select pin
5	20100521	JFR	-	<appl>_ZW040x_y_devmode.hex replaces <appl>_ZW040x_y_devmode_OTP.hex
6	20100608 20100617	JFR EFH	0 4.10.2.5	Changed pin to initiate production test mode Added Paragraph "Serial API Node List"
7	20100624 20100702	JFR EFH	4.5 & 4.6 4.9	Update FLiRS wakeup time for Door Bell and Door Lock Update user interface for Prod_Test_Gen
8	20101111	JFR	3.3 & 4	Removed Prod_Test_DUT sample application, use instead ApplicationTestPoll.
8	20101203	SSE	3.3.1.13	Added new JP hex file for the production test generator
8	20110114	JFR	3.4.9 3.4.8	Added ZWaveProgrammer USB driver supporting Windows XP/2003/Vista(32/64)/7(32/64). Added Micro RF Link diagnostic programs.
9	20110126	JFR	3	Added description of Linux applications
9	20110127	EFH	3.2.1 & 4	Updated description of common makefiles for applications
10	20110419	JFR	3.5.1 & 3.5.2.1.5	Setup Information file for installation of a USB VCP driver.
10	20110623	JFR	3.4.9	Added SD3402 crystal calibration firmware for calibration box.
11	20111004	JFR	3	Removed Z/IP Router
12	20120112	JFR	4.10 3.1 & 3.4.13	Added Serial API Power Management Added uVision project generator
13	20120523	JFR	3.4.7 3.4.9	Added Micro PVT tool Added ZWaveProgrammer source code

Table of Contents

Table of Figures

1 ABBREVIATIONS

Abbreviation	Explanation
ACK	Acknowledge
AES	The Advanced Encryption Standard is a symmetric block cipher algorithm. The AES is a NIST-standard cryptographic cipher that uses a block length of 128 bits and key lengths of 128, 192 or 256 bits. Officially replacing the Triple DES method in 2001, AES uses the Rijndael algorithm developed by Joan Daemen and Vincent Rijmen of Belgium.
ANZ	Australia/New Zealand
API	Application Programming Interface
ASIC	Application Specific Integrated Circuit
DLL	Dynamic Link Library
DUT	Device Under Test
EOOS	Execution out of SRAM
ERTT	Enhanced Reliability Test tool
EU	Europe
GNU	An organization devoted to the creation and support of Open Source software
HK	Hong Kong
HW	Hardware
IN	India
JP	Japan
JP_DK	Japan using a lower LBT RSSI threshold
LBT	Listen Before Talk
MY	Malaysia
NVM	Non-volatile memory
OTP	One Time Programmable memory
R&D	Research and Development
RF	Radio Frequency
RSSI	Received Signal Strength Indicator
SDK	Z-Wave Software Developer's Kit includes software and related software documentation.
UPnP	Universal Plug and Play
US	United States
VCP	Virtual COM Port
XML	eXtensible Markup Language
ZDK	Z-Wave Developer's Kit includes hardware, software and related software documentation.

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to describe the contents on the Z-Wave Developer's Kit. Finally, a description of all embedded sample applications including user guide or reference to relevant document.

2.2 Audience and prerequisites

The audience is Z-Wave Partners.

3 SOFTWARE COMPONENTS

The Z-Wave development software packet consists of a protocol part, sample applications and a number of tools used for developing and building the sample code.

3.1 Directory Structure

The development software is organized in the following directory structure:

```
/
        - PC
                - Bin
                        - ZW040x_USB_VCP_PC_Driver
                        - ZWaveDll
                        - ZWaveInstaller
                        - ZWavePCController
                        - ZWaveSecurityPCController
                        - ZWaveUPnPBridge
                - Source
                        - Libraries
                              - WinFormsUI
                              - ZensysFramework
                              - ZensysFrameworkUI
                              - ZensysFrameworkUIControls
                              - ZW040x_USB_VCP_PC_Driver
                              - ZWaveCommandClasses
                              - ZWaveDll
                              - ZWaveHAL
                              - ZWaveSecurity
                        - SampleApplications
                              - ZWaveInstaller
                              - ZWavePCController
                              - ZWaveSecurityPCController
                              - ZWaveUPnPBridge
```

- Product
    - Bin
        - Bin_Sensor
        - Bin_Sensor_Sec
        - Bin_Sensor_Battery
        - Bin_Sensor_Battery_Sec
        - dev_ctrl
        - dev_ctrl_AVR_Sec
        - DoorBell
        - DoorLock
        - DoorLock_Sec
        - LED_Dimmer
        - LED_Dimmer_Sec
        - Prod_Test_Gen
        - SerialAPI_Controller_Bridge
        - SerialAPI_Controller_Installer
        - SerialAPI_Controller_Portable
        - SerialAPI_Controller_Static
        - SerialAPI_Controller_Static_Norep
        - SerialAPI_Controller_Static_Single
        - SerialAPI_Slave_Enhanced
        - SerialAPI_Slave_Enhanced_232
        - SerialAPI_Slave_Routing
    - Bin_Sensor
    - dev_ctrl
    - dev_ctrl_AVR_Sec
    - DoorBell
    - DoorLock
    - LED_Dimmer
    - MyProduct
    - Prod_Test_Gen
    - SerialAPI
    - Util_Func

- Tools
    - ERTT
        - PC
        - Z-Wave_Firmware
    - FixPatchCRC
    - HexTools
    - IncDep
    - Make
    - Mergehex
    - Micro_PVT
    - Micro_RF_Link
    - Programmer
        - PC
            - Source
        - SD3402_Calibration
        - ZDP0xA_Firmware
            - Source
    - PVT_and_RF_regulatory
    - Python
    - TextTools
    - uVisionProjectGenerator
    - XML_Editor
        - PC
    - Zniffer
        - PC
            - FileConverter
        - Z-Wave_Firmware

```
- Z-Wave
        - Common
        - include
        - IO_defines
        - lib
                - controller_bridge_ZW040x
                - controller_bridge_ZW040x_devmode
                - controller_bridge_ZW040x_3CH
                - controller_bridge_ZW040x_3CH_devmode
                - controller_installer_ZW040x
                - controller_installer_ZW040x_devmode
                - controller_installer_ZW040x_3CH
                - controller_installer_ZW040x_3CH_devmode
                - controller_portable_ZW040x
                - controller_portable_ZW040x_devmode
                - controller_portable_ZW040x_3CH
                - controller_portable_ZW040x_3CH_devmode
                - controller_static_norep_ZW040x
                - controller_static_norep_ZW040x_devmode
                - controller_static_norep_ZW040x_3CH
                - controller_static_norep_ZW040x_3CH_devmode
                - controller_static_single_ZW040x
                - controller_static_single_ZW040x_devmode
                - controller_static_single_ZW040x_3CH
                - controller_static_single_ZW040x_3CH_devmode
                - controller_static_ZW040x
                - controller_static_ZW040x_devmode
                - controller_static_ZW040x_3CH
                - controller_static_ZW040x_3CH_devmode
                - ext_nvm
                - init_vars
                - rf_freq
                - slave_enhanced_232_ZW040x
                - slave_enhanced_232_ZW040x_devmode
                - slave_enhanced_232_ZW040x_3CH
                - slave_enhanced_232_ZW040x_3CH_devmode
                - slave_enhanced_ZW040x
                - slave_enhanced_ZW040x_devmode
                - slave_enhanced_ZW040x_3CH
                - slave_enhanced_ZW040x_3CH_devmode
                - slave_prodtest_gen_ZW040x
                - slave_prodtest_gen_ZW040x_devmode
                - slave_prodtest_gen_ZW040x_3CH
                - slave_prodtest_gen_ZW040x_3CH_devmode
                - slave_routing_ZW040x
                - slave_routing_ZW040x_devmode
                - slave_routing_ZW040x_3CH
                - slave_routing_ZW040x_3CH_devmode
```

This directory structure contains all the tools and sample applications needed, except the recommended Keil software, which must be purchased separately. More information about where and how to buy the Keil software development components are described in paragraph 6.1.

Note! Recommending leaving the directory structure as is due to compiler and linker issues.

The majority of the above mentioned Z-Wave specific tools and sample application are briefly described in the following sections.

3.2 Z-Wave

The Z-Wave header files and libraries are the software files needed for building a Z-Wave enabled product. The files are organized in directories used for building Z-Wave controllers and slaves respectively.

3.2.1 Common

The Common directory contains a set of standard make files needed for building the sample applications. The directory contains the following files:

- Makefile.common
- Makefile.common_ZW0x0x
- Makefile.common_ZW0x0x_appl

3.2.1.1 Application Makefile

Every sample application has a main Makefile describing what can be built. It also gives the developer an opportunity to limit what is built to a subset of this. The main Makefile includes a set of common makefiles from Z-Wave\Common directory, which defines how to build the target.

Targets can be built in lots of variants with 5 varying parameters:

- FREQUENCY
- CODE_MEMORY_MODE
- LIBRARY
- HOST_INTERFACE
- SENSOR_TYPE
- UVISION

Not all of these parameters are relevant for all applications, but the irrelevant ones are set to a default value in the applications Makefile.

For every one of these parameters, there are 3 different ways to set which one you want. This is described in the Makefile for the application. You can leave parameters unspecified. Then make will build targets for all combinations of these parameters.

The applications main Makefile defines a list of modules, which are specific for the application, and which shall be included in the build.

The applications main Makefile also defines CDEFINES, which are specific for the application.

3.2.1.2 Makefile.common

Makefile.common is included by the applications main Makefiles.

Makefile.common defines lists of the different parameter values to build with:

- LIST_OF_FREQUENCIES
- LIST_OF_CODE_MEMORY_MODES
- LIST_OF_LIBRARIES
- LIST_OF_HOST_INTERFACES
- LIST_OF_SENSOR_TYPES

You can specify subsets of values for these lists in the applications Makefile. This will override the lists specified in Makefile.common. Some of our sample applications use this technique.

Makefile.common contains the heart of the recursion engine for make. For every parameter that is not defined, a list of values will be walked through.

Makefile.common includes Makefile.common_ZW0x0x_appl and Makefile.common_ZW0x0x.

3.2.1.3 Makefile.common_ZW0x0x

This common makefile contains the linker rules, and common CDEFINES for the targets built.

3.2.1.4 Makefile.common_ZW0x0x_appl

This common makefile contains all the rules that create the build directory and compile rules for the c files and assembly .a51 files. It also contains the compiler and assembler options.

It also contains the CLASSES specification for the linking process for defining the memory layout for the target.

3.2.1.5 Makefile.common_ZW0x0x_uvision

This common makefile enable generation of uVision project files when building embedded sample application.

3.2.2 Include

The include directory contain all the header files ZW_xxx_api.h with declarations of API calls etc. For further detail, refer to [19].

Warning: Disabled linker warning L25 'DATA TYPES DIFFERENT' to allow ZW_classcmd.h updates as device and command class development progress.
Refer to Makefile.common_ZW0x0x_appl files in Common directory regarding linker parameters.

3.2.3 I/O Defines

The Product\IO_defines directory contains hardware definition files needed for building an application e.g. the development controller sample application.

AppRFSetup.a51	This file defines the normal and low power transmission levels. Change levels here if necessary.
ZW_evaldefs.h	This file contains definitions of the connector pins on the controller board.
ZW_L51_BANK.a51	This file enables code bank switching.
ZW_patchable_footer.a51	Patch system used in development mode
ZW_patchable_header.a51	Patch system used in development mode
ZW_pindefs.h	This file contains definitions of the connector pins on the Z-Wave module, and macros for accessing the I/O pins. Refer to [19] regarding a detail description.
ZW_portdefs.h	This file contains I/O port initialization vectors on the Z-Wave ASIC.
ZW_segment_tail.a51	This file enables use of XDATA located in SRAM part of the code area in development mode.

3.2.4 Libraries

The lib directory structure contains all the supported libraries.

3.2.4.1 Bridge Controller

The lib\controller_bridge_ZW040x directory contains all files needed for building a Z-Wave bridge controller application. The directory contains the following files:

ZW_controller_bridge_zw040x.lib **ZW_controller_bridge_zw040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave bridge controller application should be linked together with. JP uses the 3CH variant.
ZW_controller_bridge_zw040x_devmode.lib **ZW_controller_bridge_zw040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

The extern_eep.hex file is used to initialize the external non-volatile memory. The 32-bit home ID (xxxxxxxx) is located in byte 8, 9, 10 and 11 (when counting from 0) in the file. Byte 8 is the most significant byte and byte 11 is the least significant.

```
:200000005A654E7359730000xxxxxxxx000000000000000000000000000000000000000093
```

The Z-Wave protocol will automatically generate a new random home ID in case home ID is 0x00000000 in the external non-volatile memory. Random home ID interval is from 0xC0000000 to 0xFFFFFFFE.

3.2.4.2 Installer Controller

The lib\controller_installer_ZW040x directory contains all files needed for building a Z-Wave installer controller application. The directory contains the following files:

ZW_controller_installer_ZW040x.lib **ZW_controller_installer_ZW040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave installer controller application should be linked together with. JP uses the 3CH variant.
ZW_controller_installer_ZW040x_devmode.lib **ZW_controller_installer_ZW040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

3.2.4.3 Portable Controller

The lib\controller_ZW040x directory contains all files needed for building a Z-Wave controller application. The directory contains the following files:

ZW_controller_portable_zw040x.lib **ZW_controller_portable_zw040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave portable controller application should be linked together with. JP uses the 3CH variant.
ZW_controller_portable_zw040x_devmode.lib **ZW_controller_portable_zw040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

3.2.4.4 Static Controller

The lib\controller_static_norep_ZW040x directory contains all files needed for building a Z-Wave static controller application. The directory contains the following files:

ZW_controller_static_zw040x.lib **ZW_controller_static_zw040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave static controller application should be linked together with. JP uses the 3CH variant.
ZW_controller_static_zw040x_devmode.lib **ZW_controller_static_zw040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

3.2.4.5 Static Controller without repeater functionality

The lib\controller_static_norep_ZW040x directory contains all files needed for building a Z-Wave static controller application without repeater functionality. The directory contains the following files:

ZW_controller_static_norep_zw040x.lib **ZW_controller_static_norep_zw040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave static controller application without repeater functionality should be linked together with. JP uses the 3CH variant.
ZW_controller_static_norep_zw040x_devmode.lib **ZW_controller_static_norep_zw040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

3.2.4.6 Static Controller Single

The lib\controller_static_single_ZW040x directory contains all files needed for building a Z-Wave static single controller application. ERTT application uses this library because it supports suppression of retransmission.

WARNING: Do not use this library in product applications

The directory contains the following files:

ZW_controller_static_single_zw040x.lib **ZW_controller_static_single_zw040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave static controller single application should be linked together with. JP uses the 3CH variant.
ZW_controller_static_single_zw040x_devmode.lib **ZW_controller_static_single_zw040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

3.2.4.7 Enhanced Slave

The lib\slave_enhanced_ZW040x directory contains all files needed for building a Z-Wave enhanced slave node application. The directory contains the following files:

ZW_slave_enhanced_ZW040x.lib **ZW_slave_enhanced_ZW040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave enhanced slave application should be linked together with. JP uses the 3CH variant.
ZW_slave_enhanced_ZW040x_devmode.lib **ZW_slave_enhanced_ZW040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

3.2.4.8 Enhanced 232 Slave

The lib\slave_enhanced_232_ZW040x directory contains all files needed for building a Z-Wave enhanced slave node application. The directory contains the following files:

ZW_slave_enhanced_232_ZW040x.lib **ZW_slave_enhanced_232_ZW040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave enhanced 232 slave application should be linked together with. JP uses the 3CH variant.
ZW_slave_enhanced_232_ZW040x_devmode.lib **ZW_slave_enhanced_232_ZW040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).
extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.

3.2.4.9 Production Test Generator

The lib\slave_prodtest_ZW040x directory contains all files needed for building a production test generator application on a Z-Wave module. The directory contains the following files:

ZW_slave_prodtest_gen_ZW040x.lib **ZW_slave_prodtest_gen_ZW040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module modules that a Z-Wave production test generator application should be linked together with.
ZW_slave_prodtest_gen_ZW040x_devmode.lib **ZW_slave_prodtest_gen_ZW040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).

3.2.4.10 Routing Slave

The lib\slave_routing_ZW040x directory contains all files needed for building a Z-Wave routing slave node application on a Z-Wave module. The directory contains the following files:

ZW_slave_routing_ZW040x.lib **ZW_slave_routing_ZW040x_3CH.lib**	These files are the compiled Z-Wave protocol and API library hosted in OTP (normal mode) for a 400 Series based module that a Z-Wave routing slave application should be linked together with. JP uses the 3CH variant.
ZW_slave_routing_ZW040x_devmode.lib **ZW_slave_routing_ZW040x_3CH_devmode.lib**	These files are the same as above but used during application development (development mode).

3.2.4.11 External non-volatile memory

The ext_nvm directory contains external non-volatile memory (NVM) drivers using SPI1 and pin P2.5 as chip select:

- ZW_at25128a_spi_if.obj – Atmel SPI Serial EEPROM AT25128A
- ZW_m25pe10_spi_if.obj – STMicroelectronics Serial Flash M25PE10 (default)

The libraries support both types of external NVM. The driver adapt automatically to the external NVM in question. It is possible to overrule default in library by linking one of the above object files by modifying the makefile from:

```
# Export the variables declared above to the other makefiles.
export BASEDIR ZWLIBROOT UNDERSTAND_C
```

to the following:

```
# Set another type of the non-volatile memory
NVM_TYPE:=at25128a
# Export the variables declared above to the other makefiles.
export BASEDIR ZWLIBROOT UNDERSTAND_C NVM_TYPE
```

Drivers using alternative pins as chip select are also available:

- ZW_at25128a_p0_4_spi_if.obj, enter `NVM_TYPE:=at25128a_p0_4`
- ZW_at25128a_p1_4_spi_if.obj, enter `NVM_TYPE:=at25128a_p1_4`
- ZW_at25128a_p3_0_spi_if.obj, enter `NVM_TYPE:=at25128a_p3_0`
- ZW_at25128a_p3_4_spi_if.obj, enter `NVM_TYPE:=at25128a_p3_4`
- ZW_at25128a_p3_5_spi_if.obj, enter `NVM_TYPE:=at25128a_p3_5`
- ZW_at25128a_p3_6_spi_if.obj, enter `NVM_TYPE:=at25128a_p3_6`
- ZW_at25128a_p3_7_spi_if.obj, enter `NVM_TYPE:=at25128a_p3_7`
- ZW_m25pe10_p0_4_spi_if.obj, enter `NVM_TYPE:=m25pe10_p0_4`
- ZW_m25pe10_p1_4_spi_if.obj, enter `NVM_TYPE:=m25pe10_p1_4`
- ZW_m25pe10_p3_0_spi_if.obj, enter `NVM_TYPE:=m25pe10_p3_0`
- ZW_m25pe10_p3_4_spi_if.obj, enter `NVM_TYPE:=m25pe10_p3_4`
- ZW_m25pe10_p3_5_spi_if.obj, enter `NVM_TYPE:=m25pe10_p3_5`
- ZW_m25pe10_p3_6_spi_if.obj, enter `NVM_TYPE:=m25pe10_p3_6`
- ZW_m25pe10_p3_7_spi_if.obj, enter `NVM_TYPE:=m25pe10_p3_7`

The external NVM is accessed through the SPI1 interface. Refer to Z-Wave Memory API for details about the NVM API interface.

3.2.4.12 Variable initialization

The init_vars directory contains an init_vars.obj object file, which replaces the standard Keil initialization procedure. This reduces the time to detect whether a wakeup beam is present or not by postponing initialization. Initialization happens only in case the wakeup beam is addressed to the node in question.

3.2.4.13 RF frequency

The rf_freq directory contains all the possible RF initialization object files ZW_rf_040x_xx.obj:

- ZW_rf_040x_ALL.obj – Contains all frequencies (Used by Zniffer and Production Test Generator)
- ZW_rf_040x_ANZ.obj – Australia/New Zealand
- ZW_rf_040x_EU.obj – Europe
- ZW_rf_040x_HK.obj – Hong Kong
- ZW_rf_040x_IN.obj – India
- ZW_rf_040x_JP.obj – Japan using 32.005 MHz crystal
- ZW_rf_040x_MY.obj – Malaysia
- ZW_rf_040x_US.obj – US

3.3 Product

The Product directory contains Z-Wave sample applications for a number of different product examples. Both source code and precompiled files ready for download are supplied.

Each directory contains the necessary files for creating ANZ, EU, HK, IN, JP, MY, and US, products. JP_DK used for testing purposes due to a LBT RSSI threshold lower than -75 dBm as required by the Japanese authorities.

Hex files containing JP_32MHZ and JP_DK_32MHZ are temporary solutions supporting a 32MHz crystal.

3.3.1 Bin

The Product\Bin directory structure contains the precompiled code of the Z-Wave sample applications and the hex files needed to download to the Z-Wave ASIC via the Z-Wave Programmer.

3.3.1.1 Bin_Sensor

The Product\Bin\Bin_Sensor directory contains all files needed for running a binary sensor sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
Bin_Sensor_ZW040x_y.hex	The compiled and linked binary sensor sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
Bin_Sensor_ZW040x_y_ starter_devmode.hex **Bin_Sensor_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
Bin_Sensor_ZW040x_y_devmode.hex **Bin_Sensor_ZW040x_y_devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].

3.3.1.2 Bin_Sensor_Sec

Secure binary sensor sample application binaries not distributed due to export restrictions. Contact support via support@zen-sys.com for further information.

3.3.1.3 Bin_Sensor_Battery

The Product\Bin\Bin_Sensor_Battery directory contains all files needed for running a battery operated binary sensor sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize

	only external non-volatile memory once by downloading this file.
Bin_Sensor_Battery_ZW040x_y.hex	The compiled and linked battery operated binary sensor sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
Bin_Sensor_Battery_ZW040x_y_ starter_devmode.hex Bin_Sensor_Battery_ZW040x_y_ starter_devmode_patch_RAM.hex	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
Bin_Sensor_Battery_ZW040x_y_ devmode.hex Bin_Sensor_Battery_ZW040x_y_ devmode_patch_RAM.hex	Sample application hex files when working in development mode [23].

3.3.1.4 Bin_Sensor_Battery_Sec

Secure battery operated binary sensor sample application binaries not distributed due to export restrictions. Contact support via support@zen-sys.com for further information.

3.3.1.5 Dev_Ctrl

The Product\Bin\Dev_Ctrl directory contains all files needed for running a development controller sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
dev_ctrl_ZW040x_y.hex	The compiled and linked development controller sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
dev_ctrl_ZW040x_y_ starter_devmode.hex dev_ctrl_ZW040x_y_ starter_devmode_patch_RAM.hex	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
dev_ctrl_ZW040x_y_devmode.hex dev_ctrl_ZW040x_y_devmode_patch_RAM.hex	Sample application hex files when working in development mode [23].

3.3.1.6 Dev_Ctrl_AVR_Sec

Secure development controller sample application binaries not distributed due to export restrictions. The sample application uses an AVR ATmega128 as host on a ZDP03A Development module [14]. Configure the Z-Wave module on the ZDP03A Development module with a serial API based portable controller sample application. Contact support via support@zen-sys.com for further information.

3.3.1.7 DoorBell

The Product\Bin\DoorBell directory contains all files needed for running a bell sample application on a Z-Wave module. The development controller application is used as button in the doorbell application. The directory contains the following files:

doorbell_bell_ZW040x_y.hex	The compiled and linked bell sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
doorbell_bell_ZW040x_y_ starter_devmode.hex **doorbell_bell_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
doorbell_bell_ZW040x_y_ devmode.hex **doorbell_bell_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].

3.3.1.8 DoorLock

The Product\Bin\DoorLock directory contains all files needed for running a doorlock sample application on a Z-Wave module. The directory contains the following files:

doorlock_ZW040x_y.hex	The compiled and linked doorlock sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
doorlock_ZW040x_y_ starter_devmode.hex **doorlock_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
doorlock_ZW040x_y_ devmode.hex **doorlock_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].

3.3.1.9 DoorLock_Sec

The Product\Bin\DoorLock_Sec directory contains all files needed for running a secure doorlock sample application on a Z-Wave module. The directory contains the following files:

doorlock_ZW040x_y_SCHEME_0.hex	The compiled and linked secure doorlock sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
doorlock_ZW040x_y_SCHEME_0_ starter_devmode.hex doorlock_ZW040x_y_SCHEME_0_ starter_devmode_patch_RAM.hex	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
doorlock_ZW040x_y_SCHEME_0_ devmode.hex doorlock_ZW040x_y_SCHEME_0_ devmode_patch_RAM.hex	Sample application hex files when working in development mode [23].

NOTICE: Secure door lock sample application binaries not distributed due to export restrictions. Contact support via support@zen-sys.com for further information.

3.3.1.10 LED_Dimmer

The Product\Bin\LED_Dimmer directory contains all files needed for running a LED dimmer sample application on a Z-Wave module. The directory contains the following files:

leddimmer_ZW040x_y.hex	LED dimmer sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
leddimmer_ZW040x_y_ starter_devmode.hex leddimmer_ZW040x_y_ starter_devmode_patch_RAM.hex	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
leddimmer_ZW040x_y_devmode.hex leddimmer_ZW040x_y_devmode_patch_RAM.hex	Sample application hex files when working in development mode [23].
leddimmer_ZM4102_y.hex	LED dimmer sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4102 based module mounted on ZDP03A.
leddimmer_ZM4102_y_ starter_devmode.hex leddimmer_ZM4102_y_ starter_devmode_patch_RAM.hex	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
leddimmer_ZM4102_y_devmode.hex leddimmer_ZM4102_y_devmode_patch_RAM.hex	Sample application hex files when working in development mode [23].

3.3.1.11 LED_Dimmer_Sec

Secure LED dimmer sample application binaries not distributed due to export restrictions. Contact support via support@zen-sys.com for further information.

3.3.1.12 MyProduct

No hexadecimal files available.

3.3.1.13 Prod_Test_Gen

The Product\Bin\Prod_Test_Gen directory contains all files needed for running a production test generator sample application on a Z-Wave module. The directory contains the following files:

prod_test_gen_ZW040x_ALL.hex	The compiled and linked production test generator sample application hosted in OTP (normal mode) for all frequencies versions running on a ZM4125 (ZM4101) module mounted on ZDP03A (except JP frequency).
prod_test_gen_ZW040x_ALL_3CH.hex	The compiled and linked production test generator sample application hosted in OTP (normal mode) for JP frequency running on a ZM4125 (ZM4101) module mounted on ZDP03A.
prod_test_gen_ZW040x_ALL_starter_devmode.hex **prod_test_gen_ZW040x_ALL_starter_devmode_patch_RAM.hex** **prod_test_gen_ZW040x_ALL_3CH_starter_devmode.hex** **prod_test_gen_ZW040x_ALL_3CH_starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
prod_test_gen_ZW040x_ALL_devmode.hex **prod_test_gen_ZW040x_ALL_devmode_patch_RAM.hex** **prod_test_gen_ZW040x_ALL_3CH_devmode.hex** **prod_test_gen_ZW040x_ALL_3CH_devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].

3.3.1.14 SerialAPI_Controller_Bridge

The Product\Bin\SerialAPI_Controller_Bridge directory contains all files needed for running a serial API based bridge controller sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_controller_bridge_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_controller_bridge_ZW040x_y_ starter_devmode.hex **serialapi_controller_bridge_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP. Non starter hex files not present due to code space shortage.
SupportedFunc_serialapi_controller_bridge.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.1.15 SerialAPI_Controller_Installer

The Product\Bin\SerialAPI_Controller_Installer directory contains all files needed for running a serial API based installer controller sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_controller_installer_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_controller_installer_ZW040x_y_ starter_devmode.hex **serialapi_controller_installer_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
serialapi_controller_installer_ZW040x_y_ devmode.hex **serialapi_controller_installer_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].
SupportedFunc_serialapi_controller_installer.txt	Show enabled (1) and disabled (0) serial API

calls of released sample application.

3.3.1.16 SerialAPI_Controller_Portable

The Product\Bin\SerialAPI_Controller_Portable directory contains all files needed for running a serial API based portable controller sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_controller_portable_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_controller_portable_ZW040x_y_ starter_devmode.hex **serialapi_controller_portable_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
serialapi_controller_portable_ZW040x_y_ devmode.hex **serialapi_controller_portable_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].
SupportedFunc_serialapi_controller_portable.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.1.17 SerialAPI_Controller_Static

The Product\Bin\SerialAPI_Controller_Static directory contains all files needed for running a serial API based static controller sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_controller_static_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_controller_static_ZW040x_y_ starter_devmode.hex serialapi_controller_static_ZW040x_y_ starter_devmode_patch_RAM.hex	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP. Non starter hex files not present due to code space shortage.
SupportedFunc_serialapi_controller_static.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.1.18 SerialAPI_Controller_Static_Norep

The Product\Bin\SerialAPI_Controller_Static_Norep directory contains all files needed for running a serial API based static controller sample application without repeater functionality on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_controller_static_norep_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_controller_static_norep_ZW040x_y_ starter_devmode.hex serialapi_controller_static_norep_ZW040x_y_ starter_devmode_patch_RAM.hex	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP. Non starter hex files not present due to code space shortage.
SupportedFunc_serialapi_controller_static_norep.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.1.19 SerialAPI_Controller_Static_Single

The Product\Bin\SerialAPI_Controller_Static_Single directory contains all files needed for running an ERTT based serial API static controller sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_controller_static_single_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_controller_static_single_ZW040x_y_ starter_devmode.hex **serialapi_controller_static_single_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
serialapi_controller_static_single_ZW040x_y_ devmode.hex **serialapi_controller_static_single_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].
SupportedFunc_serialapi_controller_single.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.1.20 SerialAPI_Slave_Enhanced

The Product\Bin\SerialAPI_Slave_Enhanced directory contains all files needed for running a serial API based enhanced slave sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_slave_enhanced_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_slave_enhanced_ZW040x_y_ starter_devmode.hex **serialapi_slave_enhanced_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
serialapi_slave_enhanced_ZW040x_y_ devmode.hex **serialapi_slave_enhanced_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].
SupportedFunc_serialapi_slave_enhanced.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.1.21 SerialAPI_Slave_Enhanced_232

The Product\Bin\SerialAPI_Slave_Enhanced_232 directory contains all files needed for running a serial API based enhanced 232 slave sample application on a Z-Wave module. The directory contains the following files:

extern_eep.hex	This file contains the external non-volatile memory data on the ZM4125 module. Initialize only external non-volatile memory once by downloading this file.
serialapi_slave_enhanced_232_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_slave_enhanced_232_ZW040x_y_ starter_devmode.hex **serialapi_slave_enhanced_232_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
serialapi_slave_enhanced_232_ZW040x_y_ devmode.hex **serialapi_slave_enhanced_232_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].
SupportedFunc_serialapi_slave_enhanced_232.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.1.22 SerialAPI_Slave_Routing

The Product\Bin\SerialAPI_Slave_Routing directory contains all files needed for running a serial API based routing slave sample application on a Z-Wave module. The directory contains the following files:

serialapi_slave_routing_ZW040x_y.hex	The compiled and linked production serial API sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions running on a ZM4125 (ZM4101) module mounted on ZDP03A.
serialapi_slave_routing_ZW040x_y_ starter_devmode.hex **serialapi_slave_routing_ZW040x_y_ starter_devmode_patch_RAM.hex**	Sample application hex files when working in starter development mode [23]. Starter indicates that only library is present in OTP.
serialapi_slave_routing_ZW040x_y_ devmode.hex **serialapi_slave_routing_ZW040x_y_ devmode_patch_RAM.hex**	Sample application hex files when working in development mode [23].
SupportedFunc_serialapi_slave_routing.txt	Show enabled (1) and disabled (0) serial API calls of released sample application.

3.3.2 Binary Sensor

The Product\Bin_Sensor directory contains sample source code for a non-secure/secure binary sensor and non-secure/secure battery operated binary sensor application.

3.3.3 Development Controller

The Product\Dev_Ctrl directory contains sample source code for the development controller application used on the ZM12xxRE Module mounted on the Z-Wave Development module. For further information refer to section 4.3 and reference [8].

3.3.4 Secure Development Controller based on serial API and using an AVR as host

The Product\Dev_Ctrl_AVR_Sec directory contains sample source code for the secure development controller. The sample application uses an AVR ATmega128 as host on a ZDP02A/ZDP03A Development module. Configure the Z-Wave module on the ZDP02A/ZDP03A Development module with a serial API based portable controller sample application. For further information refer to section 4.4 and reference [11].

3.3.5 Doorbell

The Product\DoorBell directory contains sample source code for the doorbell application used on the Z-Wave Interface module. Use the Development Controller application to control the doorbell application. For further information, refer to section 4.5.

3.3.6 Door Lock

The Product\DoorLock directory contains sample source code for the non-secure and secure door lock application on a Z-Wave module. Use the Secure Development Controller application to control the door lock application. For further information, refer to section 4.6.

3.3.7 LED Dimmer

The Product\LED_Dimmer directory contains sample source code for the non-secure and secure dimmer application on a Z-Wave module, which uses the LEDs to simulate a light switch with a built in dimmer. For further information, refer to section 4.7.

3.3.8 MyProduct

The Product\MyProduct directory contains sample source code for a routing slave application on a Z-Wave module. For further information, refer to section 4.8.

3.3.9 Production Test Generator

The Product\Prod_Test_Gen directory contains sample source code for a production test generator application on a Z-Wave module. For further information, refer to section 4.9.

3.3.10 Serial API

The Product\SerialAPI directory contains sample source code for the Serial API sample applications. For further information about the Serial API, refer to section 4.10.

3.3.11 Utilities

The Product\util_func directory contains some helpful functions that are used by several of the sample applications.

AES_module.h	This header file contains definitions for implementing secure communication using AES as encrypting/decrypting engine.
association.c **association.h**	The files contain sample code that shows how association between nodes could be implemented on a Z-Wave module. This sample code holds all associations in RAM and the number of nodes/groupings possible using this implementation is limited. Applications using this collection of functions must implement three functions (ApplicationStoreAll, ApplicationInitAll, ApplicationClearAll). These should handle the storage in nonvolatile memory if this is desired.
battery.c **battery.h**	The files contain sample code that shows how battery operated devices may implement power down, wake up notification and network update requests. Applications using this collection must call the following functions at their appropriate location: *UpdateWakeupCount* – call from ApplicationInitSW to update the wakeup counter which determines the wakeup interval on application level (200-series) – Only called when Wakeupreason is WUT-Kicked. *InitRTCActionTimer* – call from ApplicationInitSW, to activate the RTC timer. (100-Series) *HandleWakeupFrame* – call from ApplicationCommandHandler to handle incoming COMMAND_CLASS_WAKE_UP is received. Handles WAKE_UP_INTERVAL_GET/SET/NO_MORE_INFORMATION. *SetDefaultBatteryConfiguration* – is called from ApplicationInitHW when node is reset, and from SetDefaultConfiguration. Sets the default values for powerdown timeout, sleep time and networkupdate. *LoadBatteryConfiguration* – call from LoadConfiguration. Loads the battery related information from EEPROM and make them available for the running application. *SaveBatteryConfiguration* – call from SaveConfiguration. Saves the battery related information to EEPROM. *StartPowerDownTimer* – call from ApplicationInitSW and set as callback function ZW_SEND_DATA methods after which the node should enter sleep mode. Please refer to the BatterySensor sample application for an example on how this can be implemented.
ctrl_learn.h **ctrl_learn.c**	The files contain sample code for how to handle learn mode on controller nodes.
one_button.c **one_button.h**	Enables easy use of a button. The functions detect whether a button has been pressed shortly or is being held. To initialise the button detection, run OneButtonInit() from ApplicationInitSW. And call OneButtonLastAction when button information is needed (e.g. in ApplicationPoll()).
self_heal.c	Support functions to implement Lost / Self Heal functionality. This file is

self_heal_non_zero_vars.c **self_heal.h**	mandatory if the battery helper functions are used and ZW_SELF_HEAL is defined. See the battery.c and bin_sensor.c source files for help on using the functions.
slave_learn.h **slave_learn.c**	The files contain sample code for how to handle learn mode on slave nodes. These two files are used by all slave based sample code in the SDK. The sample application should just call StartLearnModeNow() to enter learnmode and transmit nodeinformation. Inclusion uses normal power. The sample application should then wait for the BOOL learnState to go FALSE before doing transmissions.
ZW_AES128.h	This header file contains definitions for the security solution on application level.
ZW_FLiRS.c **ZW_FLiRS.h**	The files contain sample code for how to handle FLiRS nodes.
ZW_Security_AES_module.c **ZW_Security_AES_module.h**	The files contain sample code for the functionality supporting secure communication using AES as encryption/decryption mechanism.
ZW_TransportLayer.h	Transport layer type selector
ZW_TransportNative.h	Implements functions for transporting frames over the native Z-Wave Network.
ZW_TransportSecurity.h **ZW_TransportSecurity.c**	Implements functions for transporting frames over the secure Z-Wave Network.
ZWZip6lowPanIphc.h **ZWZip6lowPanIphc.c**	Implements functions for IPv6 to 6lowPAN data conversions.

3.4 Tools

The Tools directory contains various tools needed for building and debugging the sample applications. All tools in this directory can freely be used for building Z-Wave applications.

3.4.1 ERTT

This directory contains the PC software and the embedded code for the Enhanced Reliability Test Tool (ERTT). Notice that the PC based Controller now supports the ERTT functionality. For further details, refer to [4].

The ERTT directory contains the following files:

PC\setup.exe **PC\ZWaveControllerSetup.msi**	PC application.
Z-Wave_Firmware\extern_eep.hex	This file contains the external NVM data on the Z-Wave module. Initialize only external NVM once.
serialapi_controller_static_single_ZW040x_y.hex	Static controller single based serial API (COM port) sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions for a 400 Series based module.
serialapi_controller_static_single_ZW040x_USB_y.hex	Static controller single based serial API (USB announcing itself as a virtual COM port) sample application hosted in OTP (normal mode) for y = ANZ, EU, HK, IN, JP, MY and US frequency versions for a 400 Series based module.

3.4.2 FixPatchCRC

This directory contains a tool used when building patchable sample applications.

3.4.3 HexTools

This directory contains a tool used when building patchable sample applications.

3.4.4 IncDep

This directory contains a python script that is used for making dependency files when building the sample applications.

3.4.5 Make

This directory contains a DOS/Windows version of the GNU make utility. The make utility is used for building the sample applications.

3.4.6 Mergehex

This directory contains a tool used for merging two files in Intel hex format. The tool is used for building external non-volatile memory files in the sample code.

3.4.7 Micro PVT

This Micro_PVT k directory contains embedded programs to check the RF performance on a device regardless of the contents in the OTP. This makes the tool suitable to investigate the RF performance of un-programmed or already programmed devices and when performing PVT measurements. The only requirement for using the tool is that the programming interface to the chip must be available and the UART0 interface for communication.This the 400 Series chip must be programmed in EOOS mode allowing execution of code out of the 4K SRAM. For further details, refer to [25].

The Micro_RF_Link directory contains the following files:

micro_pvt_rx_9K6_calval.hex	Testing RF Rx communication at 9.6kbit/s on SD3402 or ZM4101
micro_pvt_rx_9K6_ZM4102_calval.hex	Testing RF Rx communication at 9.6kbit/s on ZM4102
micro_pvt_rx_40K_calval.hex	Testing RF Rx communication at 40kbit/s on SD3402 or ZM4101
micro_pvt_rx_40K_ZM4102_calval.hex	Testing RF Rx communication at 40kbit/s on ZM4102
micro_pvt_rx_100K_calval.hex	Testing RF Rx communication at 100kbit/s on SD3402 or ZM4101
micro_pvt_rx_100K_ZM4102_calval.hex	Testing RF Rx communication at 100kbit/s on ZM4102
micro_pvt_tx_9K6_calval.hex	Testing RF Tx communication at 9.6kbit/s on SD3402 or ZM4101
micro_pvt_tx_9K6_ZM4102_calval.hex	Testing RF Tx communication at 9.6kbit/s on ZM4102
micro_pvt_tx_40K_calval.hex	Testing RF Tx communication at 40kbit/s on SD3402 or ZM4101
micro_pvt_tx_40K_ZM4102_calval.hex	Testing RF Tx communication at 40kbit/s on ZM4102
micro_pvt_tx_100K_calval.hex	Testing RF Tx communication at 100kbit/s on SD3402 or ZM4101
micro_pvt_tx_100K_ZM4102_calval.hex	Testing RF Tx communication at 100kbit/s on ZM4102

3.4.8 Micro RF Link

This Micro_RF_Link directory contains embedded programs to check the RF link on a device regardless of the contents in the OTP. This requires that the 400 Series chip must be programmed in EOOS mode allowing execution of code out of the 4K SRAM. For further details, refer to [22].

The Micro_RF_Link directory contains the following files:

micro_rf_link_9K6_ZW040x.hex	Testing RF link at 9.6kbit/s
micro_rf_link_40K_ZW040x.hex	Testing RF link at 40kbit/s
micro_rf_link_100K_ZW040x.hex	Testing RF link at 100kbit/s

3.4.9 Programmer

This Programmer directory contains the PC software and ATMega128 firmware for the non-volatile memory programming of the Z-Wave 100/200/300/400 Series Chips. The Z-Wave Programmer also supports programming of the external EEPROM on the Z-Wave modules. Finally, the Z-Wave Programmer can also be used to configure transmission power and RF settings on the Z-Wave modules and lock bits. For further details, refer to [7].

The Programmer directory contains the following files:

PC\setup.exe	Programmer application.
PC\CP210x_VCP_Win_XP_S2K3_Vista_7.exe	The CP210x USB to UART Bridge Virtual COM Port (VCP) driver. This driver supports Windows XP/2003/Vista(32/64)/7(32/64).
PC\Source\...	ZWaveProgrammer PC source code providing Windows GUI and interface to ATMega128 situated on the ZDP02A/ZDP03A Development Platform. For further details regarding communication protocol interface, refer to [12].
ZDP0xA_Firmware\ATMega128_Firmware.hex	The compiled and linked Z-Wave Programmer bootloader for the ATMega128 situated on the ZDP02A/ZDP03A Development Platform.
ZDP0xA_Firmware\ZWaveProgrammer_FW.hex	The compiled and linked Z-Wave Programmer firmware for the ATMega128 situated on the ZDP02A/ZDP03A Development Platform.
ZDP0xA_Firmware\Source\...	Z-Wave Programmer firmware source code for the ATMega128 situated on the ZDP02A/ZDP03A Development Platform. For further details regarding communication protocol interface, refer to [12].
SD3402_Calibration\SD3402_Calibration.hex	SD3402 crystal calibration firmware used by the calibration box, refer to [24]. ZM4101 and ZM4102 are already calibrated during production.

3.4.10 PVT and RF Regulatory

This directory contains software used in connection with PVT and RF regulatory measurements on the hardware. All programs run in "Development Mode" and reside in the SRAM part using polling instead of interrupt. The programs requires download of a hex file in the OTP part for wanted frequency before they operates correct. For a guideline how to carry out the measurements, refer to [4].

The PVT_and_RF_regulatory directory contains the following 400 Series related files:

ZW0401_y_OTP.hex	This hex file must be downloaded into OTP using y = ANZ, EU, HK, IN, JP, MY and US frequency versions of the 400 Series based products. Hex file contains a jump vector to below sample applications and RF constants. Preprogrammed calibration value in chip is not affected.
ZW0401_rx_100kbps_y.hex	400 Series configured to receive 100 kbps signal using y = ANZ, EU, HK, IN, MY and US frequency versions of the 400 Series based products.
ZW0401_rx_40kbps_y.hex	400 Series configured to receive 9.6/40 kbps signal using y = ANZ, EU, HK, IN, MY and US frequency versions of the 400 Series based products.
ZW0401_rx_100kbps_JP_y.hex	400 Series configured to receive 100kbps modulated signal on y = ch0(950.9514MHz), ch1(954.5508MHZ) and ch2(955.3508MHz)
ZW0401_TXcar_100kbps_y.hex **ZW0401_TXcar_40kbps_y.hex** **ZW0401_TXcar_9k6bps_y.hex**	400 Series constantly transmits a carrier y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22 MHz (IN) or 868.10MHz (MY) or 908.42MHz (US).
ZW0401_TXcar_100kbps_JP_y.hex	400 Series constantly transmits a carrier y = ch0(950.9514MHz), ch1(954.5508MHZ) and ch2(955.3508MHz)
ZW0401_TXmod_100kbps_y.hex **ZW0401_TXmod_40kbps_y.hex** **ZW0401_TXmod_9k6bps_y.hex**	400 Series constantly transmits a modulated signal y +/-25kHz, where y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22 MHz (IN) or 868.10MHz (MY) or 908.42MHz (US).

ZW0401_TXmod_100kbps_JP_y.hex	400 Series configured to constantly transmit a 100kbps modulated signal at y = ch0(950.9514MHz), ch1(954.5508MHZ) and ch2(955.3508MHz)
ZW0401_TXmod_Transient_100kbps_y.hex **ZW0401_TXmod_Transient_40kbps_y.hex** **ZW0401_TXmod_Transient_9k6bps_y.hex**	400 Series configured to constantly transmit a 100kbps modulated signal at y = 868.42MHz (EU), when pressing IO pin P11.
ZW0401_TXmod_Transient_100kbps_JP_y.hex	400 Series configured to constantly transmit a 100kbps modulated signal at y = ch0(950.9514MHz), ch1(954.5508MHZ) and ch2(955.3508MHz), when pressing IO pin P11.

In addition, the PVT_and_RF_regulatory directory contains the following 200/300 Series related files:

ZW0201_rx_y.hex	Puts the ZW0201 in receive mode for y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0201 based products.
ZW0201_TXcar_y.hex	ZW0201 constantly transmits a carrier y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) or 869.0MHz (RU) or 908.42MHz (US).
ZW0201_TXmod_y.hex **ZW0201_TXmod_40kbps_y.hex**	ZW0201 constantly transmits a modulated signal y +/- 25kHz, where y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) or 869.0MHz (RU) or 908.42MHz (US). Hex file ZW0201_TXmod_y.hex covers 9.6kbit/s.
ZW0301_rx_y.hex	Puts the ZW0301 in receive mode. y = ANZ, EU, HK, IN, MY, RU and US frequency versions of the ZW0301 based products.
ZW0301_TXcar_y.hex	ZW0301 constantly transmits a carrier y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) 868.42MHz or 908.42MHz (US).
ZW0301_TXmod_y.hex **ZW0301_TXmod_40kbps_y.hex**	ZW0301 constantly transmits a modulated signal y +/- 25kHz, where y = 921.42MHz (ANZ) or 868.42MHz (EU) or 919.82MHz (HK) or 865.22MHz (IN) or 868.10MHz (MY) 868.42MHz or 908.42MHz (US). Hex file ZW0301_TXmod_y.hex covers 9.6kbit/s.

3.4.11 Python

This directory contains a python scripting language interpreter. Python is used for various purposes in the sample code build process.

3.4.12 TextTools

This directory contains the sed stream editor used to modify text strings during the make process.

3.4.13 uVisionProjectGenerator

This directory contains uVision Project Generator program; the program generate uVision projects when running the makefile from a DOS prompt.

The uVisionProjectGenerator directory contains the following files:

__init__py	uVision Project Generator application files.
j.py	
MakePatch.bat	
och51.bat	
uv-find-segment-end.bat	
uVisionProjectGenerator.exe	

3.4.14 XML Editor

This directory contains the XML Editor program; the program can be used to define approved Z-Wave device and command classes used by the application layer of the Z-Wave protocol. The XML file can be used by the Zniffer for interpretation of the device and command classes. The customer can also define device and command classes under development or proprietary command class structures enabling interpretation by the Zniffer.

Beside a XML file containing all the information, it is also possible to generate a C# class file and C header file as foundation for Z-Wave application development. For further details refer to [10].

The XML Editor directory contains the following files:

PC\setup.exe	XML Editor application.
PC\Setup.msi	

3.4.15 Zniffer

This directory contains the Zniffer program; the program is a development tool for capturing Z-Wave RF communication and presenting the frames in a graphical user interface on a PC. The tool shows the node ID of the Source and Destination for the communication, the type of frame being sent, and the application content, i.e. the specific command, which is being sent.

The Zniffer tool is a passive "listener" to the Z-Wave network traffic, and will only display the RF communications taking place within direct RF range. Be also aware that Zniffer can occasionally miss RF communication even from Z-Wave nodes within direct range.

The tool consists of two parts, an embedded part that should be downloaded to a Z-Wave module and a PC application that should run on a PC attached to the Z-Wave module via the serial interface. For further details refer to [5].

The Zniffer directory contains the following files:

PC\setup.exe **PC\ZnifferSetup.msi**	Zniffer application supporting Windows XP/2003/Vista(32/64)/7(32/64)
PC\FileConverter\setup.exe **PC\FileConverter\FileConverterSetup.msi**	FileConverter enable Zniffer to automatically convert old file formats *.znf to latest *.zlf when opening file.
Z-Wave_Firmware\sniffer_ZW040x.hex	Zniffer application supporting ANZ, EU, HK, IN, JP, MY and US versions on a 400 Series based module.
Z-Wave_Firmware\sniffer_ZW030x_y.hex	Zniffer application supporting y = ANZ, EU, HK, IN, MY and US frequency versions on a ZW0301 based module.
Z-Wave_Firmware\sniffer_ZW020x_y.hex	Zniffer application supporting y = ANZ, EU, HK, IN, MY and US frequency versions on a ZW0201 based module.

3.5 PC

The PC directory contains three PC sample applications demonstrating the use of the Z-Wave DLL and Serial API.

3.5.1 Bin

The PC\Bin directory contains the program or installation executables of the PC sample applications.

ZW040x_USB_VCP_PC_Driver\uzb.inf	Setup Information file used by Microsoft Windows for installation of a USB VCP driver.
ZWaveDll\setup.exe **ZWaveDll\ZWaveSetup.msi**	Installation executables of the Z-Wave DLL framework. This framework simplifies development of PC sample applications. Installation includes also a help file describing Z-Wave DLL architecture, namespaces and how to create a Z-Wave DLL based PC application.
ZWaveInstaller\setup.exe **ZWaveInstaller\ZWaveInstallerToolSetup.msi**	Installation executables of the Z-Wave Installer Tool sample application.
ZWavePCController\setup.exe **ZWavePCController\ZWaveControllerSetup.msi**	Installation executables of the Z-Wave Non-secure PC Controller sample application.
ZWaveSecurityPCController\setup.exe **ZWaveSecurityPCController\ZWaveSecurityControllerSetup.msi**	Installation executables of the Z-Wave Secure PC Controller sample application. Binaries not distributed due to export restrictions. Contact support via support@zen-sys.com for further information.
ZWaveUPnPBridge\setup.exe **ZWaveUPnPBridge\ZWaveUPnPBridgeSetup.msi**	Installation executables of the Z-Wave to UPnP Bridge sample application.

3.5.2 Source

The PC\Source directory contains the C# source code of the PC sample applications using the Microsoft Visual Studio 2008 environment.

3.5.2.1 Libraries

The PC\Source\Libraries directory contains various libraries used by the PC sample applications.

3.5.2.1.1 WinForms UI

The PC\Source\Libraries\WinFormsUI directory contains C# source code of the windows docking library.

WinFormsUI.csproj	Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

3.5.2.1.2 Zensys Framework

The PC\Source\Libraries\ZensysFramework directory contains C# source code of the additional functions, formatters, helpers.

ZensysFramework.csproj	Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

3.5.2.1.3 Zensys Framework UI

The PC\Source\Libraries\ZensysFrameworkUI directory contains C# source code of the completed Z-Wave UI elements that can be reused in applications:

- Associations View Control;
- Bridged UPnP Device View Control;
- Controller View Control;
- Node View Control;
- UPnP Binary Light Device View Control;
- UPnP Device Scaner View Control;
- UPnP Media Renderer View Control.

ZensysFrameworkUI.csproj	Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

3.5.2.1.4 Zensys Framework UI Controls

The PC\Source\Libraries\ZensysFrameworkUIControls directory contains C# source code of the additional UI elements such as:

- ListDataView;
- TreeDataView;
- BitBox;
- ThreadSafeLabel.

ZensysFrameworkUIControls.csproj	Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

3.5.2.1.5 ZW040x USB VCP PC Driver

The PC\Source\Libraries\ ZW040x_USB_VCP_PC_Driver directory contains the Setup Information file used by Microsoft Windows for installation of a USB VCP driver. It maps an USB port into a virtual COM port.

uzb.inf	Setup Information file used by Microsoft Windows for installation of a USB VCP driver.

3.5.2.1.6 Z-Wave Command Class

The PC\Source\Libraries\ZWaveCommandClasses directory contains C# source code for the XML parser, which enables parsing of Z-Wave frames by the Zniffer and generating frames by the PC based applications.

ZWaveCommandClasses.csproj	Microsoft Visual Studio 2008 project file containing information at the project level and used to build the project.

3.5.2.1.7 Z-Wave DLL

The PC\Source\Libraries\ZWaveDll directory contains C# source code of the dynamic link library used by the PC application to communicate with a 400 Series based module via the serial API interface. Refer to [6] for further details.

SerialZWaveDll.sln	Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

3.5.2.1.8 Z-Wave HAL

The PC\Source\Libraries\ZWaveHAL directory contains C# source code of the Z-Wave High-level Application Layer in terms of Z-Wave Dll architecture. It contains common functions that are used in Z-Wave enabled PC applications: ZWavePCController, ZWaveProgrammer, ZWaveUPnPBridge etc. Refer to [6] for further details.

SerialZWaveHAL.sln	Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

3.5.2.2 Sample Application

The PC\Source\SampleApplications contains the various the PC applications

3.5.2.2.1 Z-Wave Installer

The PC\Source\SampleApplications\ZWaveInstaller directory contains C# sample source code for a PC based installer tool using the Z-Wave DLL etc. Further reading on how to use the PC based Installer Tool see [2].

ZWaveInstallerTool.sln	Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

3.5.2.2.2 Z-Wave PC Controller

The PC\Source\SampleApplications\ZWavePCController directory contains C# sample source code for a PC based controller using the Z-Wave DLL etc. Further reading on how to use the PC based Controller see [1].

ZWaveController.sln	Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

3.5.2.2.3 Z-Wave Security PC Controller

The PC\Source\SampleApplications\ZWaveSecurityPCController directory contains C# sample source code for a Secure PC based controller using the Z-Wave DLL etc. Further reading on how to use the PC based Controller see [1].

ZWaveSecurityController.sln	Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

3.5.2.2.4 Z-Wave UPnP Bridge

The PC\Source\SampleApplications\ZWaveUPnPBridge directory contains C# sample source code for a PC based Z-Wave Bridge using the Z-Wave DLL etc. Further readings on how to use the Z-Wave UPnP Bridge see [3].

ZWaveUPnPBridge.sln	Microsoft Visual Studio 2008 solutions file containing information at the project level and used to build the project.

4 APPLICATION SAMPLE CODE

The Z-Wave Developer's Kit includes several sample applications: a serial controller application, a LED dimmer application, a binary sensor and a battery operated binary sensor application for the Z-Wave module. The sample application realizes a light control system to help the developer to understand how the various components can interact. In addition the Z-Wave Developer's Kit also comprises of a number of PC centric sample applications for illustrating advanced functionalities of the Z-Wave protocol:

1. How a Z-Wave Module can be controlled from a PC.
2. Installation including display of network topology.
3. Bridging to and from other networks.

The 400 Series build environment is different compared to previous 100/200/300 Series because the ASIC contains 64KB OTP instead of 32KB Flash. However, the ASIC support a development mode enabling application development. Refer to [23] for further details about the development environment.

4.1 Binary Sensor Sample Code

The Developer's Kit contains sample code for a non-secure and secure binary sensor. This device is in effect a binary sensor where the sensor input is the pin also used as a button input on the device module. The Bin_Sensor_Sec will on every button release transmit a basic set frame to any associated devices. If the button is held for a little while instead a nodeInfo frame will be transmitted. A static controller such as the one described in [1] can control, configure and assign routes to the Bin_Sensor_Sec.

The Bin_Sensor_Sec is a binary sensor that supports the association command class described in the device class specification (see ref [1]). This device complies with the specific device class named routing binary sensor device class (4.1). When included non-secure the Secure Binary Sensor application lists the following supported command classes in the Node Information Frame:

Non-Secure Included

- Binary Sensor command class
- Association command class
- Version command class
- Manufacturer Specific command class
- Security command class

Secure Included

When included secure the Secure Binary Sensor application lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Binary Sensor command class
- Association command class
- Manufacturer Specific command class
- Version command class

The Basic command class is secure because application does not list it in Node Information Frame.

The Bin_Sensor_Sec is a slave device based on the enhanced slave API. During initialization, the Bin_Sensor_Sec will initialize the mounted button, the 4 LED's and a timer function that handles the button input and sensor input (in this example the same as the button input). It will also get stored data from the NVM. After the initialization the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the Bin_Sensor_Sec main function. The **ApplicationPoll** function checks if the button or the sensor input has changed state and then acts accordingly to the current state the Bin_Sensor_Sec is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the Bin_Sensor_Sec. This function checks the command and acts according to the command. When transmitting the Bin_Sensor_Sec will, if routes have been assigned use these.

The Bin_Sensor_Sec implements Lost functionality and network topology maintenance by using a series of methods. If the device is unsuccessful in sending a message a predefined count it will enter lost state, and attempt to find a SUC in the network, and if successful ask the SUC for routes to the failing devices.

At regular intervals the Bin_Sensor_Sec will transmit a Static Route Request, which asks the SUC for any updates done to the network.

The functions for AES128 encryption/decryption use the built in AES engine in the 400 Series avoiding 3rd party products subjected to intellectual property (IP) rights and licensing issues.

4.1.1 Network Wide Inclusion

By default the node will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The node will stay in NWI mode for 4 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the node enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.3.1 regarding implementation details.

4.1.2 User Interface

The following table defines the functionality of the button on the Z-Wave module.

	Button Triple Clicked	**Button Clicked**
In Network	Node Info Frame / Enter learn mode	Basic Set (Broadcast)
Not in Network	Node Info Frame / Enter learn mode	None
Associated	Node Info Frame / Enter learn mode	Basic Set (to associated nodes)

Learn mode is now activated by pressing the button three times within 1.5 seconds to avoid unintentional inclusion/exclusion of the node.

4.1.3 Bin_Sensor Files

The Product\Bin_Sensor directory contains sample source code for a non-secure/secure binary sensor and a non-secure/secure battery powered binary sensor slave application on a Z-Wave module. The application uses also a number of utility functions described in section 3.3.11.

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK "FREQUENCY=EU"** in command prompt.

Makefile / Makefile.SecureTargets

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

eeprom.h

This header file defines the addresses where application data are stored in the external non-volatile memory.

Bin_Sensor.h / Bin_Sensor.c

These files contain the source code for the binary sensor application state machine. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll, ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

Bin_Sensor_patch.c

This file contains the patched source code of Bin_Sensor.c

Bin_Sensor_ZW040x_....Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

4.1.3.1 Macros for accessing the LED's

LED_ON(led)

Turn LED on.

Parameter:
    led - LED number

Example:
```
    PIN_OUT(LED1);  /* define LED1 as an output pin */
    LED_ON(1);      /* turn LED 1 on */
```

LED_OFF(led)

Turn LED off.

Parameter:
    led - LED number

Example:
```
    LED_OFF(1);     /* turn LED 1 off */
```

LED_TOGGLE(led)

Toggle the LED OFF if the LED was ON and ON if the LED was OFF.

Parameter:
    led - LED number

Example:
```
  LED_TOGGLE(1);    /* toggle LED 1 */
```

4.2 Binary Sensor Battery Sample Code

The Developer's Kit contains sample code for a non-secure and secure battery powered binary sensor. This device is in effect a binary sensor where the sensor input is the pin also used as a button input on the device module. When the binary sensor is inactive the ASIC will be powered down. The binary sensor will power up when the button is pressed or the RTC / WUT[1] is fired. Upon wakeup, be it button press or RTC/WUT a Wakeup Notification Frame is sent either as broadcast or as singlecast to the device that configured the wakeup settings. If the devie has any associations it will transmit a basic set to the associated devices. If the button is held for a longer time a Node Information Frame is transmitted. A static controller such as the one described in **[1]** can control, configure and assign routes to the Bin_Sensor_Battery_Sec.

The Bin_Sensor_Battery_Sec is a binary sensor that supports the association command class and the Wake Up command class described in the device class specification (see ref [1]). This device complies with the specific device class named routing binary sensor device class (4.1). When included non-secure the secure battery-operated Binary Sensor application lists the following supported command classes in the Node Information Frame:

Non-Secure Included

- Binary Sensor command class
- Wake Up command class
- Association command class
- Version command class
- Manufacturer Specific command class
- Security command class

Secure Included

When included secure the secure battery-operated Binary Sensor application lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Binary Sensor command class
- Wake Up command class
- Association command class
- Version command class
- Manufacturer Specific command class

The Basic command class is secure because application does not list it in Node Information Frame.

[1] RTC is used in ZW0102 and WUT is used in ZW0201.

The Bin_Sensor_Battery_Sec is a slave device based on the enhanced slave API. During initialization, the Bin_Sensor_Battery_Sec will initialize the mounted button, the 4 LED's and a timer function that handles the button input and sensor input (in this example the same as the button input). It will also get stored data from the NVM. After the initialization will go in power down mode and it will wakeup again either when the button is pressed or when the RTC timer / WUT is fired. While the Bin_sensor_Battery_Sec is wake the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the Bin_Sensor_Battery_Sec main function. The **ApplicationPoll** function checks if the button or the sensor input has changed state and then acts accordingly to the current state the Bin_Sensor_Battery_Sec is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the Bin_Sensor_Battery_Sec. This function checks the command and acts according to the command. When transmitting the Bin_Sensor_Battery_Sec will, if routes have been assigned use these. If the Bin_sensor_Battery was waked by the sensor input or button activity, then it will power down again it is done executing any event caused by the sensor input or the button. If the binary sensor is woken up by RTC timer / WUT and the wakeup time interval is expired then it will send wake notification frame and wait for 5 second before powering down again.

The Bin_Sensor_Battery_Sec implements Lost functionality and network topology maintenance by using a series of methods. If the device is unsuccessful in sending a message a predefined count it will enter lost state, and attempt to find a SUC in the network, and if successful ask the SUC for routes to the failing devices. At regular intervals the Bin_Sensor_Battery_Sec will transmit a Static Route Request, which asks the SUC for any updates done to the network.

Note that the wakeup notification frame will only be sent when the Bin_sensor_Battery_Sec has been assigned a node ID.

On the 400 Series some of the uninitialized RAM bytes are used to keep track of the WUT timer. See also [19].

The Bin_Sensor_Sec and Bin_Sensor_Battery_Sec share the same code base. They are distinquished between by defining BATTERY when compiling which will also enable use of the utility function file battery.c/h.

The functions for AES128 encryption/decryption use the built in AES engine in the 400 Series avoiding 3rd party products subjected to intellectual property (IP) rights and licensing issues.

4.2.1 Network Wide Inclusion

By default the node will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The node will stay in NWI mode for 4 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the node enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.3.1 regarding implementation details.

4.2.2 User Interface

The following table defines the functionality of the button on the Z-Wave module.

	Button Triple Pressed	**Button Clicked**
In Network	Node Info Frame / Enter learn mode	Basic Set (Broadcast)
Not in Network	Node Info Frame / Enter learn mode	None
Associated	Node Info Frame / Enter learn mode	Basic Set (to associated nodes)
Wakeup Node set	Node Info Frame / Enter learn mode	Wake Up Notifications (to Wake up node)
Wakeup Node not set	Node Info Frame / Enter learn mode	Wake Up Notifications (Broadcast)

4.2.3 Bin_Sensor Files

Refer to chapter 4.1.3.

4.3 Development Controller Sample Code

The Developer's Kit contains sample code that demonstrates how the basic tasks of adding, removing and controlling devices in a Z-Wave network using the Z-Wave portable controller API.

The Application complies with the Generic Controller command class [20]. When included the Development Controller application lists the following supported command classes in the Node Information Frame:

- Controller Replication command class
- Version command class

The Development Controller controls the following command classes:

- Controller Replication command class
- Basic command class
- Association command class

Controlled command classes not listed in the Node Information Frame in this sample application because it is optional to list.

For details regarding functionality supported by the development controller and user interface, refer to [8].

The Z-Wave basis software continually calls the **ApplicationPoll** function. The **ApplicationPoll** function contains a state machine, which initiates actions from user input. The **ApplicationCommandHandler** function is called when the Z-Wave basis software receives a frame. This could be a Basic Get Command to obtain the dim level of a multilevel switch.

4.3.1 Network Wide Inclusion

By default the controller will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included or have included other nodes itself. The controller will stay in NWI mode for 4 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the controller enter NWI mode again is by doing hardware reset by either remove and reapply the power or press the reset button on the side of the board.

The flow diagram below show how the node request NWI, which is implemented on application level. The ctrl_learn.c file contains the implementation situated in the util_func directory.

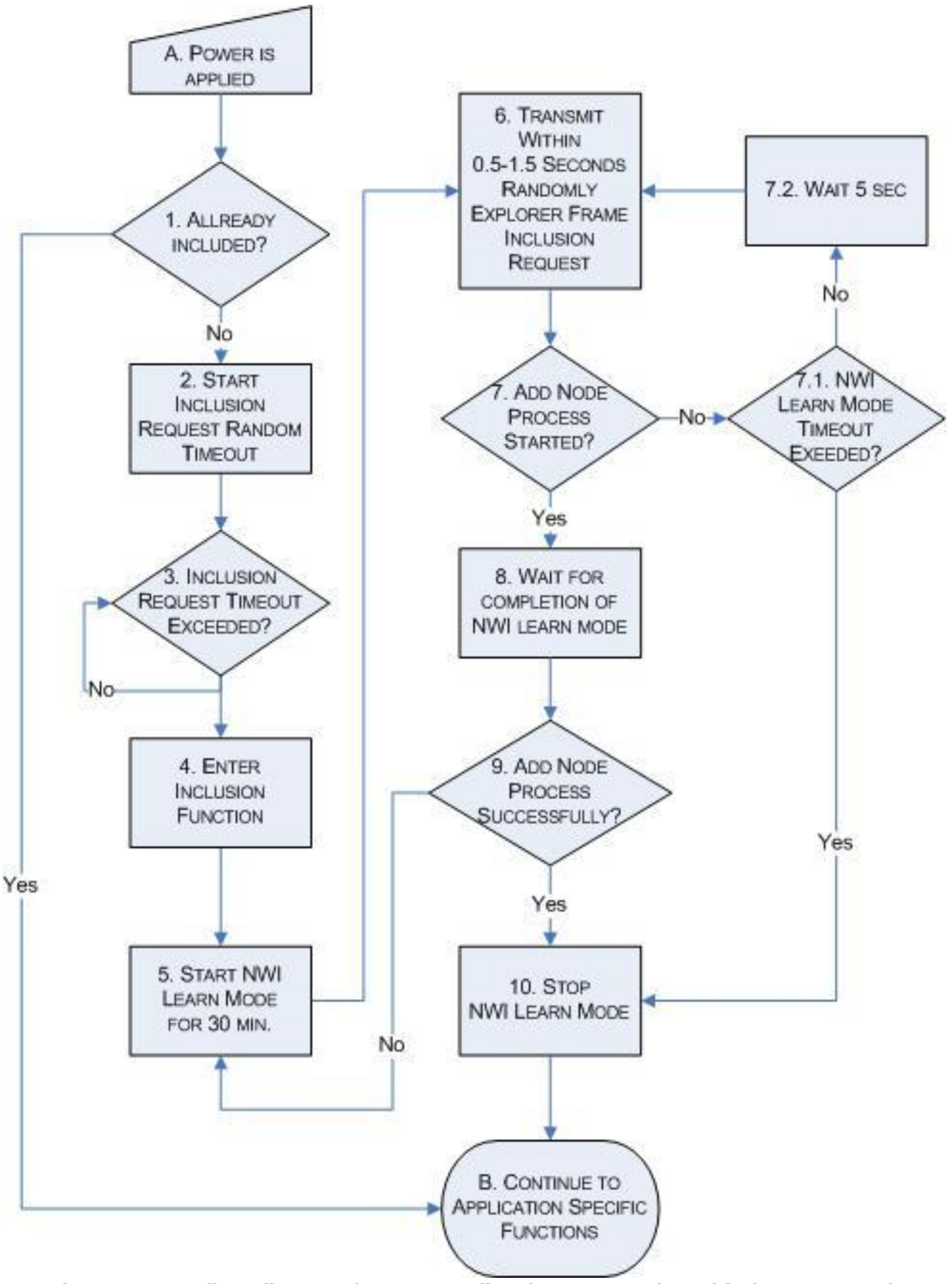

A. POWER IS APPLIED
1. ALLREADY INCLUDED?
No
Yes
2. START INCLUSION REQUEST RANDOM TIMEOUT
3. INCLUSION REQUEST TIMEOUT EXCEEDED?
No
4. ENTER INCLUSION FUNCTION
5. START NWI LEARN MODE FOR 30 MIN.
6. TRANSMIT WITHIN 0.5-1.5 SECONDS RANDOMLY EXPLORER FRAME INCLUSION REQUEST
7.2. WAIT 5 SEC
No
7. ADD NODE PROCESS STARTED?
No
7.1. NWI LEARN MODE TIMEOUT EXEEDED?
Yes
8. WAIT FOR COMPLETION OF NWI LEARN MODE
9. ADD NODE PROCESS SUCCESSFULLY?
Yes
No
Yes
10. STOP NWI LEARN MODE
B. CONTINUE TO APPLICATION SPECIFIC FUNCTIONS

Figure 1, NWI flow diagram for a controller that want to be added to a network

Finally, the controller will also accept network wide inclusion requests when used as primary/inclusion controller adding nodes into its network.

4.3.2 Production test mode

To initiate production test mode short-circuits J16-pin3 to J17-pin1 (ground) on ZDP03A.

After resetting the ZDP03A in production test mode the following happens:

1. Initializes RF ready to receive NOP frames and acknowledge them. Use node ID equal to 0x01 in NOP frame and home ID value is ignored in production test mode. The Production test generator can now be used to test RF link and remember to change node ID to 0x01.

2. Radio start to send constant unmodulated signal on channel 0 by pressing push button on Z-Wave module hosting 400 Series chip/module once.

3. Radio start to send constant modulated signal on channel 0 by pressing push button on Z-Wave module once.

4. Radio start to send constant unmodulated signal on channel 1 by pressing push button on Z-Wave module once.

5. Radio start to send constant modulated signal on channel 1 by pressing push button on Z-Wave module once.

6. Radio start to send constant unmodulated signal on channel 2 by pressing push button on Z-Wave module once (3 channels system only, for 2 channel systems it jumps back to point 2).

7. Radio start to send constant modulated signal on channel 2 by pressing push button on Z-Wave module once (3 channels system only).

8. Jump back to point 2 by pressing push button on Z-Wave module.

The production test mode application is located in the **ApplicationTestPoll** function.

4.3.3 Dev_Ctrl Files

The Product\Dev_ctrl directory contains the source code for the controller application. The application uses also a number of utility functions described in section 3.3.11.

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK "FREQUENCY=EU"** in command prompt.

Makefile

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

eeprom.c / eeprom.h

These files contain functions and define for accessing the application data in the external non-volatile memory.

dev_ctrl_if.h

This file defines how the IO connections on the Z-Wave module are connected to the ZDP03A Module.

dev_ctrl.c / dev_ctrl.h

These files contain the source code for the development controller application state machine. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll, ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

dev_ctrl_patch.c

This file contains the patched source code of dev_ctrl.c

p_button.c / p_button.h

These files contain functions and define for detecting Push button presses. This includes de-bounce checking.

p_button_patch.c

This file contains the patched source code of p_button.c

dev_ctrl_ZW040x_....Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

4.4 Secure Development Controller (ATmega) Sample Code

The SDK contains sample code that demonstrates how the basic tasks of adding, removing and controlling devices in a Z-Wave network can be accomplished using a host processor to control a Serial API based portable controller application. The application is a security updated Development Controller application. The Z-Wave Development Platform ZDP03A [14] is used for this purpose. The host processor is an AVR ATmega128 and software is builded by environment below:

- IAR Embedded Workbench for Atmel AVR (v. 4.30A)
- IAR C/C++ Compiler for AVR 4.30A/W32 (4.30.1.5)

The AVR ISP In-System Programmer programs the AVR Atmega128.

When included non-secure the Development Controller application lists the following supported command classes in the Node Information Frame:

Non-Secure Included

- Controller Replication command class
- Version command class
- Security command class

Secure Included

When included secure the Development Controller lists the following supported command classes in the Node Information Frame:

- Version command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Controller Replication command class
- Version command class

The Basic command class is secure because application does not list it in Node Information Frame.The Development Controller controls the following command classes:

- Controller Replication command class
- Basic command class
- Association command class

Controlled command classes not listed in the Node Information Frame in this sample application because it is optional to list.

For further information about the features of the Secure Development Controller using an AVR as host, see [11].

4.4.1 Dev_Ctrl_AVR_Sec Files.

The Product\dev_ctrl_AVR_Sec directory contains the source code for the controller application. Only selected files in the directory structure is described below.

Portable.dep /.ewd /.ewp /.eww

Project files used to build AVR based sample application.

include\ZW_Security_AES_module.h

Header file used to implement security on application level.

include\AES_module.h

This header file contains definitions for implementing secure communication using AES as encrypting/decrypting mechanism.

src\ZW_Security_AES.c

These files contain shared data and functions for AES128 and functions for AES128 encryption/decryption. Files are not distributed on the Developer's Kit CD due to export restrictions. Contact support via support@zen-sys.com for further information.

Alternatively, implement the functions based on an Atmel's Application Note "AVR231: AES Bootloader":

http://www.atmel.com/dyn/resources/prod_documents/doc2589.pdf

4.5 Door Bell Sample Code

The developer's Kit contains sample code for a Door Bell sample application. This device an example of how a battery operated chime in a doorbell system could be build. The Door Bell uses the frequently listening mode where it powers up the radio for a short period every 250ms@2-ch and 1000ms@3-ch and if it receives a command it will power up entirely and turn on the LED's.

The Door Bell based on the routing slave library and it has its generic device class set to Binary Switch and the specific device class set to none. The Door Bell supports the following command classes:

- Binary Switch command class
- Version command class

NOTE: This node will fail certification because when its level is set to on with a binary set command it will toggle its state back to off again after a timeout to emulate the behavior of a doorbell.

4.5.1 Network Wide Inclusion

By default the node will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The node will stay in NWI mode for 4 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the node enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.3.1 regarding implementation details.

4.5.2 User interface

The following list defines the functionality of the button on the Z-Wave module.

Press shortly	Wake up for 2 sec.
Press 3 times within 1.5 sec.	Enter learn mode and timeout after 3 sec.

The LEDs on the Z-Wave module has the following meaning:

LED D1	LED D2	LED D3	Description
Off	Off	Off	The door bell is in powerdown mode (Frequently listening mode)
On	Off	Off	The node was woken up by button press or reset
Off	On	Off	The node was woken up by an RF beam
Off	Off	On	The node is in learn mode
On	On	On	Bell was turned on by Binary or Basic set command

4.5.3 Door Bell Files

The Product\DoorBell directory contains the source code and makefiles for the application. The application uses also a number of utility functions described in section 3.3.11.

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK "FREQUENCY=EU"** in command prompt.

Makefile

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

Bell.h / Bell.c

This file contains the source code for the Door Bell sample application

Bell_patch.c

This file contains the patched source code of Bell.c

DoorBell_ZW040x_....Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

4.6 Door Lock Sample Code

The SDK contains sample code for a non-secure and secure Door Lock sample application. This device shows an example of how a door lock system could be build.

A controller such as the Development Controller and secure Development Controller (Atmega) can control the Door Lock. The Door Lock uses the frequently listening mode (FLiRS) where it powers up the radio for a short period every 1000ms@2-ch and 1000ms@3-ch and in case a wakeup beam for this particular node is detected then it stay awake to receive a command. It is now possible to turn the LED on/off indicating lock/unlock status. After receiving one command, it returns to frequently listening mode again to conserve battery consumption.

The Door Lock is based on the enhanced slave library and it has its generic device class set to Entry Control and the specific device class set to Door Lock. When included non-secure the Door Lock application lists the following supported command classes in the Node Information Frame:

Non-Secure Included

- Lock command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class
- Security command class (not used by a non-secure Door Lock application)

Secure Included

When included secure the Door Lock lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Lock command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class

The Basic command class is secure because application does not list it in Node Information Frame.

During initialization, the Door Lock will initialize the mounted button and one LED. It will also get stored data from the NVM. After the initialization the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the Door Lock main function. The **ApplicationPoll** function checks button activation and act according to the state the Door Lock is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the Door Lock. This function checks the command and acts according to the command.

The functions for AES128 encryption/decryption use the built in AES engine in the 400 Series avoiding 3rd party products subjected to intellectual property (IP) rights and licensing issues.

4.6.1 Network Wide Inclusion

By default the node will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The node will stay in NWI mode for 4 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the node enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.3.1 regarding implementation details.

4.6.2 User Interface

The following table defines the functionality of the button on the Z-Wave module.

	Button Triple Pressed	**Button Clicked**
In Network	Node Info Frame / Enter learn mode	Toggle on/off status
Not in Network	Node Info Frame / Enter learn mode	Toggle on/off status

Learn mode is now activated by pressing the button three times within 1.5 seconds to avoid unintentional inclusion/exclusion of the node.

4.6.3 Door Lock Files

The Product\DoorLock directory contains the source code and makefiles for the application. The application uses also a number of utility functions described in section 3.3.11.

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK "FREQUENCY=EU"** in command prompt.

Makefile

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

eeprom.h

This header file contains the address definitions in the external non-volatile memory used to store application data.

DoorLock.h / DoorLock.c

This file contains the source code for the non-secure and secure Door Lock sample application

DoorLock_patch.c

This file contains the patched source code of DoorLock.c

DoorLock_ZW040x_....Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

4.6.3.1 Macros for accessing the Lock/Unlock

PIN_ON(pin)

Set output pin to 1.

Parameter:

```
pin - Z-Wave pin name
```

Example:

```
PIN_ON(TRIACpin);        /* turn TRIACpin on */
```

PIN_OFF(pin)

Set output pin to 0.

Parameter:

```
pin - Z-Wave pin name
```

Example:

```
PIN_OFF(TRIACpin);       /* turn TRIACpin off */
```

PIN_GET(pin)

Read pin value.

Parameter:

```
pin - Z-Wave pin name
```

Example:

```
PIN_GET(SSN);            /* Read pin SSN value*/
```

PIN_IN(pin, pullup)

Set I/O pin as input.

Parameter:

pin - Z-Wave pin name
pullup - if not zero activate the internal pullup resistor

Example:

```
PIN_IN(SSN, 0);          /* Set I/O pin SSN as input and activate the internal pullup resistor */
```

4.7 LED Dimmer Sample Code

The Developer's Kit contains sample code for a non-secure and secure LED Dimmer. This device is in effect a light switch with a built in dimmer where the light bulb is substituted with 3 LED's when using 400 Series. A controller such as the secure or non-secure Development Controller can control the LED Dimmer.

The LED Dimmer is a multilevel switch that supports the all switch command class, the protection command class and the powerlevel command class described in the device class specification (see ref [1]). This device complies with the specific device class named multilevel power switch device class. The LED Dimmer does not support the optional Clock command class. When included non-secure the secure LED Dimmer application lists the following supported command classes in the Node Information Frame:

Non-Secure Included

- Multilevel Switch command class
- All Switch command class
- Protection command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class
- Security command class

Secure Included

When included secure the secure LED Dimmer lists the following supported command classes in the Node Information Frame:

- Version command class
- Manufacturer Specific command class
- Security command class

The following listed in the Security Commands Supported Report frame:

- Multilevel Switch command class
- All Switch command class
- Protection command class
- Powerlevel command class
- Version command class
- Manufacturer Specific command class

The Basic command class is secure because application does not list it in Node Information Frame.

The secure LED Dimmer is a slave device based on the slave/enhanced slave API. During initialization, the secure LED Dimmer will initialize the mounted button and the 3 LED's. It will also get stored data from the NVM. After the initialization, the Z-Wave basis software will continually call the **ApplicationPoll** function, which contains the Secure LED Dimmer main function. The **ApplicationPoll** function checks button activation and act according to the state the secure LED Dimmer is in. The other main function is the **ApplicationCommandHandler** function that is called every time a command has been received, destined for the secure LED Dimmer. This function checks the command and acts according to the command.

The functions for AES128 encryption/decryption use the built in AES engine in the 400 Series avoiding 3rd party products subjected to intellectual property (IP) rights and licensing issues.

4.7.1 Network Wide Inclusion

By default the node will enter network wide inclusion (NWI) to be added to a network when it is powered up and have not already been included. The node will stay in NWI mode for 4 minutes or until it has been included into the network. Any key press will terminate the NWI mode and the only way to make the node enter NWI mode again is by doing hardware reset either by remove and reapply the power or press the reset button on the side of the board. Refer to section 4.3.1 regarding implementation details.

4.7.2 User Interface

The following table defines the functionality of the button on the Z-Wave module.

	Button Triple Pressed	**Button Clicked**	**Button is held**
In Network	Node Info Frame / Enter learn mode	Toggle on/off status	Dim up/down
Not in Network	Node Info Frame / Enter learn mode	Toggle on/off status	Dim up/down

Learn mode is now activated by pressing the button three times within 1.5 seconds to avoid unintentional inclusion/exclusion of the node.

4.7.3 Production test mode

To initiate production test mode short-circuits J17-pin10 to J17-pin1 (ground) on ZDP03A.

After resetting the ZDP03A in production test mode the following happens:

1. Initializes RF ready to receive NOP frames and acknowledge them. Use node ID equal to 0x00 in NOP frame and home ID value is ignored in production test mode. The Production test generator can now be used to test RF link because directly default node ID used is equal to 0x00.

2. Radio start to send constant unmodulated signal on channel 0 by pressing push button on Z-Wave module hosting 400 Series chip/module once.

3. Radio start to send constant modulated signal on channel 0 by pressing push button on Z-Wave module once.

4. Radio start to send constant unmodulated signal on channel 1 by pressing push button on Z-Wave module once.

5. Radio start to send constant modulated signal on channel 1 by pressing push button on Z-Wave module once.

6. Radio start to send constant unmodulated signal on channel 2 by pressing push button on Z-Wave module once (3 channels system only, for 2 channel systems it jumps back to point 2).

7. Radio start to send constant modulated signal on channel 2 by pressing push button on Z-Wave module once (3 channels system only).

8. Jump back to point 2 by pressing push button on Z-Wave module.

The production test mode application is located in the **ApplicationTestPoll** function.

4.7.4 Secure_LED_Dimmer Files

The Product\LED_Dimmer directory contains sample source code for a slave application on a Z-Wave module. The application uses also a number of utility functions described in section 3.3.11.

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK "FREQUENCY=EU"** in command prompt.

Makefile / Makefile.SecureTargets

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

eeprom.h

This header file defines the addresses where application data are stored in the external non-volatile memory.

LEDdim.h / LEDdim.c

This file contains the source code for the LED dimmer application state machine. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll, ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

LEDdim_patch.c

This file contains the patched source code of LEDdim.c. LED1 (D0 on ZDP03A) is inverted in the patched code to differ between patchable and patched source code.

LED_Dimmer_ZW040x_....Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

4.7.4.1 Macros for accessing the LED's

LED_ON(led)

Turn LED on.

Parameter:
  led - LED number

Example:
```
PIN_OUT(LED1);  /* define LED1 as an output pin */
LED_ON(1);      /* turn LED 1 on */
```

LED_OFF(led)

Turn LED off.

Parameter:
  led - LED number

Example:
```
LED_OFF(1);     /* turn LED 1 off */
```

LED_TOGGLE(led)

Toggle the LED OFF if the LED was ON and ON if the LED was OFF.

Parameter:
  led - LED number

Example:
```
LED_TOGGLE(1);      /* toggle LED 1 */
```

4.8 MyProduct Sample Code

The My Product contains the minimum framework to begin developing a slave application. To realize the application in question it is often easier to modify the existing sample code applications than build one from scratch based on MyProduct.

4.8.1 MyProduct Files

The Product\MyProduct directory contains sample source code for a routing slave application on a Z-Wave module. The application uses also a number of utility functions described in section 3.3.11.

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK "FREQUENCY=EU"** in command prompt.

Makefile

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

MyProduct.h / MyProduct.c

This file contains the source code for the MYProduct. The common API functions such as **ApplicationInitHW**, **ApplicationInitSW**, **ApplicationNodeInformation**, **ApplicationPoll, ApplicationSlaveUpdate** and **ApplicationCommandHandler** are defined here.

MyProduct_patch.c

This file contains the patched source code of MyProduct.c

MyProduct_ZW040x_....Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

4.9 Production Test Generator

The Developer's Kit contains sample code that demonstrates how the basic tasks of testing devices in a Z-Wave network can be accomplished using the Z-Wave API. The Z-Wave generator is used to verify the TX / RX circuits on Z-Wave enabled products.

A simple generator consists of a ZW040x Interface Module and a ZMxx20 Z-Wave Module.

On the Interface module there are 6 LED diodes, which have these assignments in the Prod_Test_Gen sample application:

LED #	Colour	Description
D6	Green	Power on
D1	Red	Error
D2	Red	Success
D3	Red	Send (flashes during transmission)
D4	Red	-
D5	Red	Indication of Push button

The push button on the ZMxx20 Z-Wave Module is the "Test" button.

After connection to power, the red "Error" LED 'D1' on the Interface module will be on.

When the push button is pressed, 10 NOP's will be transmitted. A device under test (application in production test mode executing ApplicationTestPoll) is expected to verify the reception of each NOP with an ACK. During transmission, the red LED 'D3' will flash.

If all NOP's are replied correctly, the red "Error" LED 'D1' will turn off and the red "Success" LED 'D2' will turn on and stay on until the next test is conducted. If the DUT does not reply correctly, the red "Error" LED 'D1' will turn on and stay on until the next test is conducted.

The Z-Wave basis software continually calls the **ApplicationPoll** function. The **ApplicationPoll** function contains a state machine, which initiates actions from user input. The **ApplicationCommandHandler** function is only called when the Z-Wave basis software receives information for the application.

The Production Test Generator sample application is based on the ZW_slave_prodtest_gen library.

The application is controlled via RS232 (115200,8,N,1) or button with fixed timings:
Device will resopond to any char received with an ASCII SPACE followed by a command answer or error '!' followed by error information:
Following ASCII commands are implemented.
Received:

'U':
Frequency US is selected sending 9.6kbps on channel 1
Response is: ' US'

'E':
Frequency EU is selected sending 9.6kbps on channel 1
Response is: ' EU'

'Z':
Frequency ANZ is selected sending 9.6kbps on channel 1
Response is: ' ANZ'

'M':
Frequency MY is selected sending 9.6kbps on channel 1
Response is: ' MY'

'I':
Frequency IN is selected sending 9.6kbps on channel 1
Response is: ' IN'

'J':
Frequency JP is selected sending 100kbps on channel 1
Response is: ' JP'

'n' (where n = 0..9):
Any frequency is selected from the table of defined frequencies. The input shall be 2 decimal digits.
Response for the first digit is: ' n'
Response for the second digit is: ' n 0xnn' (where nn is the selected hexadecimal index in the table of defined frequencies.

'S':
Start test
Response is ' ST'

'C':
Set the number of NOPs to send to 1000
Response: ' CO'

'N':
Set the destination node ID.
Response: ' NI'

'R':
Reset the hardware
Response: ' RS'

On Unknown:
'!' 'received Char'

4.9.1 Production Test Generator Files

The Product\Prod_Test_Gen directory contains the source code for the Production Test Generator sample application.

MK.BAT

Make bat file for building the sample application in question.

Makefile

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

prod_test_gen.c

This file contains the main source code for the sample application. Both **ApplicationPoll** and **ApplicationCommandHandler** are defined in this file.

Prod_test_gen_patch.c

This file contains the patched source code of prod_test_gen.c

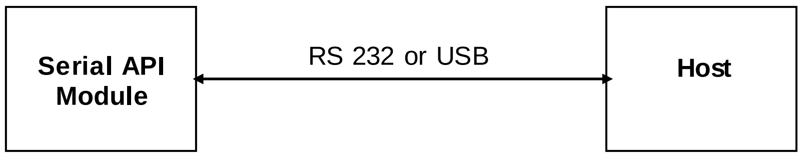
Prod_Test_Gen_ZW040x….Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

4.10 Serial API Embedded Sample Code

The purpose of the Serial API embedded sample code is to show how a 400 Series Z-Wave module can be controlled via the RS 232 or USB port by a host. The following host based PC applications are available on the Developer's Kit CD:

- The PC based Controller application showing the available functionality in a Serial API based on a static controller API.
- The PC based Installer Tool application showing the available functionality in a Serial API based on an installer API.
- The PC based Z-Wave Bridge application showing the available functionality in a Serial API based on a bridge controller API.



The Serial API can be used as it is or it can be changed to fit specific needs. The UART on the Z-Wave Module is initialized for 115200 baud, no parity, 8 data bits and 1 stop bit.

4.10.1 Supported API Calls

Only a subset of the API calls is available via the serial interface. In [19] each API call has a description regarding Serial API support and the corresponding frame format and flow.

4.10.2 Implementation

The Serial API embedded sample code is provided on the Z-Wave Developer's Kit. Be aware that altering the function ID's and frame formats in the Serial API embedded sample code can result in interoperability problems with the Z-Wave DLL supplied on the Developer's Kit as well as commercially available GUI applications. Regarding how to determine the current version of the Serial API protocol in the embedded sample code please refer to the API call **ZW_Version**. The following sections describe the Serial API implementation and how a host can communicate with the Serial API embedded sample code.

4.10.2.1 Frame Layout

The protocol between the PC (host) and the Z-Wave Module (ZW) consists of three frame types: ACK frame, NAK frame and Data frame. Each Data frame is prefixed with SOF byte and Length byte and suffixed with a Checksum byte. As of Serial API Version 4 a fourth frame type has been defined; the CAN frame.

ACK frame:

The ACK frame is used to acknowledge a successful transmission of a data frame. The format is as follows:

7	6	5	4	3	2	1	0
ACK (0x06)							

NAK frame:

The NAK frame is used to de-acknowledge an unsuccessful transmission of a data frame. The format is as follows:

7	6	5	4	3	2	1	0
NAK (0x15)							

Only a frame with a LRC checksum error is de-acknowledged with a NAK frame.

CAN frame:

The CAN frame is used by the ZW to instruct the host that a host transmitted data frame has been dropped. Happens when ZW expects an ACK as handshake for a transmitted frame, but instead get a new frame from host. The format is as follows:

7	6	5	4	3	2	1	0
CAN (0x18)							

Data frame:

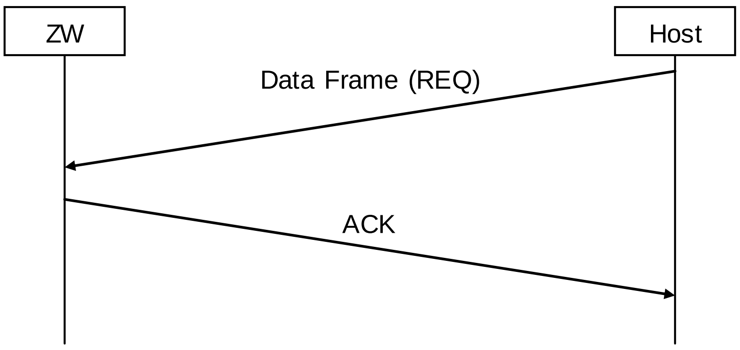
The Data frame contains the Serial API command including parameters for the command in question. The format is as follows:

7	6	5	4	3	2	1	0
SOF							
Length							
Type							
Serial API Command ID							
Command Specific Data							
…							
Checksum							

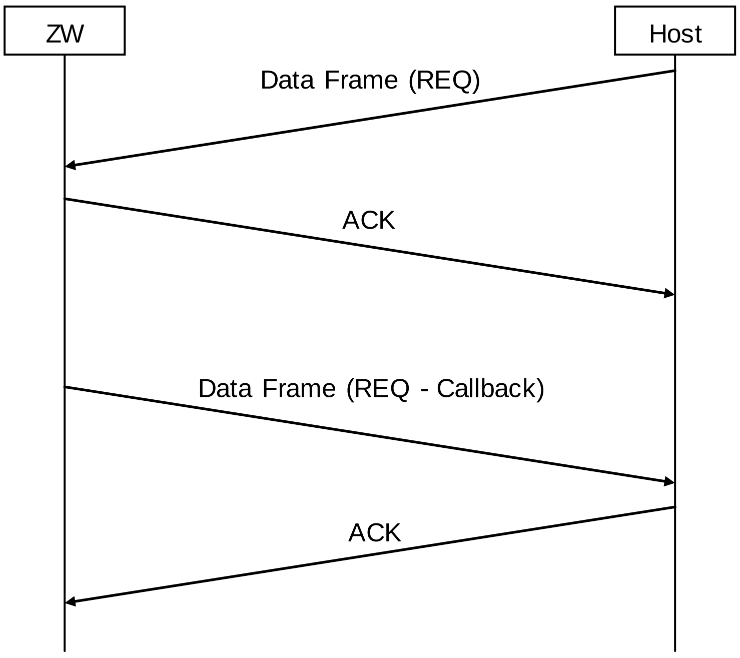
Field	Description
SOF	Start Of Frame. Used for synchronization and is equal to 0x01
Length	Number of bytes in the frame, exclusive SOF and Checksum. The host application is responsible for entering the correct length field. The current Serial API embedded sample code does no validation og the length field.
Type	Used to distinguish between unsolicited calls and immediate responses (not callback). The request (REQ) is equal to 0x00 and response (RES) is equal to 0x01.
Serial API Command ID	Unique command ID for the function to be carried out. Any data frames returned by this function will contain the same command ID
Command Specific Data	One or more bytes of command specific data. Possible callback handling is also defined here.
Checksum	LRC checksum used to check for frame integrity. Checksum calculation includes the **Length**, **Type**, **Serial API Command Data** and **Command Specific Data** fields. The Checksum is a XOR checksum with an initial checksum value of 0xFF. For a checksum implementation refer to the function ConTxFrame in the conhandle.c module

4.10.2.2 Frame Flow

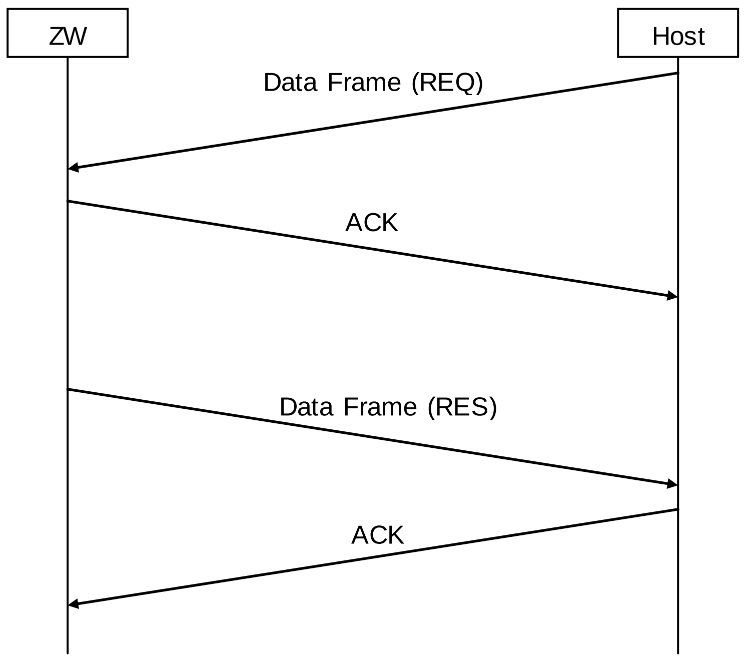
The frame flow between a host and a Z-Wave module (ZW) running the Serial API embedded sample code depends on the API call. There are four different ways to conduct communication between the host and ZW.



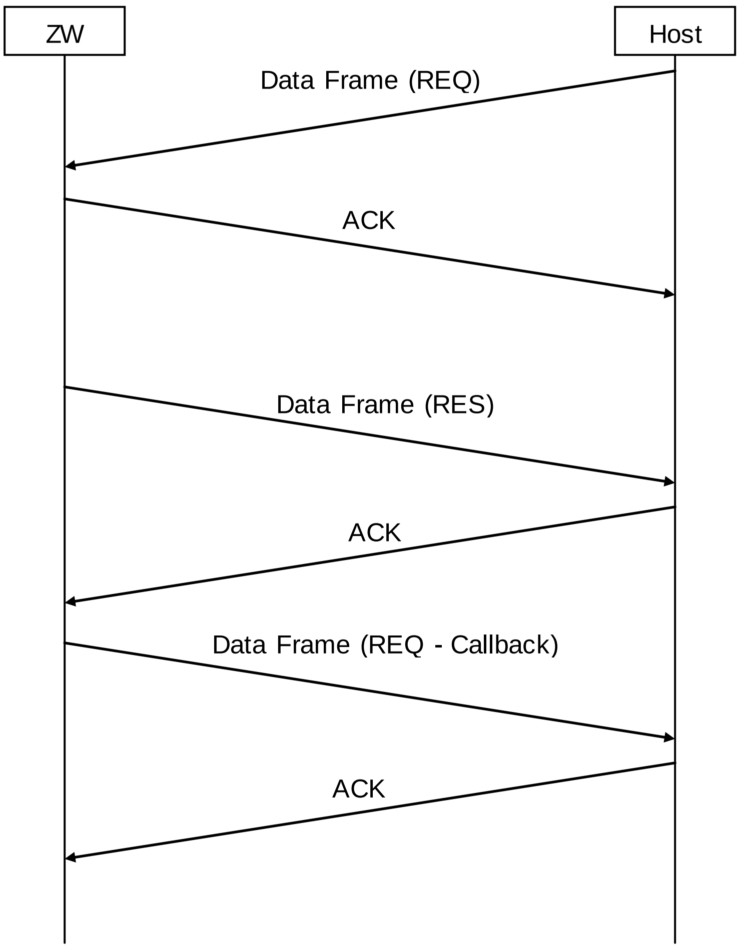
Data frame (REQ) from host, which is acknowledged by ZW when successfully received. An example could be the API call **ZW_SetExtIntLevel**.



Data frame (REQ) with callback function enabled from host by setting funcID different from zero, which is acknowledged by ZW when successfully received. The funcID is a sequence number used to correlate a given reply with the correct request. A data frame (REQ - callback) is returned by ZW with the result at command completion including a funcID to correlate callback with the original data frame. The host acknowledged the data frame when successfully received. Setting the funcID equal to 0 in the original data frame disable the callback handling. An example could be the API call **ZW_SetDefault**.



Data frame from host (REQ), which is acknowledged by ZW when successfully received. A data frame (RES) is returned by ZW with the result at command completion. The host acknowledges the data frame when successfully received. An example could be the API call **ZW_GetControllerCapabilities**.



Data frame (REQ) with callback function enabled from host by setting funcID different from zero, which is acknowledged by ZW when successfully received. The funcID is a sequence number used to correlate a given reply with the correct request. A data frame (RES) is returned by ZW with the status at command initiation. The host acknowledges the data frame when successfully received. A data frame (REQ - Callback) is returned by ZW with the result at command completion including a funcID to correlate callback with the original data frame. The host acknowledged the data frame when successfully received. Setting the funcID equal to 0 in the original data frame disable the callback handling. An example could be the API call **ZW_SendSUCID**.

4.10.2.3 Error handling

A number of scenarios exist, which can impede the normal frame flow between the host and the Z-Wave module running the Serial API embedded sample code (ZW).

A LRC checksum failure is the only case there is de-acknowledged by a NAK frame in the current Serial API embedded sample code. When a host receives a NAK frame can it either retry transmission of the frame or abandon the task. A task is defined as the whole frame flow associated with the execution of a specific Serial API function call. If a NAK frame is received by the Z-Wave module in response to a just transmitted frame, then the frame in question is retransmitted (max 2 retries).

Frames with an illegal length are ignored without any notification. Frames with an illegal type (only REQ and RES exists) are ignored without any notification

The Serial API embedded sample code can only perform one host-initiated task at a time. A data frame will be dropped without any notification (no ACK/NAK frame transmitted) by the ZW if it is not ready to execute a new host-initiated task. As of Serial API version 4 a CAN frame is transmitted by the ZW when a received data frame is dropped.

If no CAN frame is received the host detect the missing ACK/NAK by implementing a timeout mechanism in the receive function. The host timeout must correspond to the timeout defined in ZW. A reasonable timeout in the host is 2 seconds because the current Serial API embedded sample code has a default timeout of 1.5 seconds. The timeout in the Serial API (as of SerialAPI version 4) can also be set by using the FUNC_ID_SERIAL_API_SET_TIMEOUTS Serial API function:

Serial API:

HOST->ZW: REQ | 0x06 | RXACKtimeout | RXBYTEtimeout

ZW->HOST: RES | 0x06 | oldRXACKtimeout | oldRXBYTEtimeout

RXACKTimeout is the max no. of 10ms ticks the ZW waits for an ACK before timeout. RXBYTETimeout is the max no. of 10ms ticks the ZW waits for a new byte before timeout; this is only valid when a frame has been detected and is being collected.

In case the host expect an ACK but instead receive another data frame then it must read the whole data frame and ACK/NAK accordingly, it will probably also receive a CAN frame to indicate that the ZW has dropped the host transmitted data frame. Afterwards can the host restart transmission of the pending frame ZW never ACK'ed or possibly CAN'ed.

Communication between ZW and other Z-Wave nodes can also result in deviations from the normal frame flow. A get command on application level can for example result in multiple reports coming back and ZW will just pass on the reports to the host. This can happen in case the Z-Wave node did not hear ZW acknowledge the report and therefore it is retransmitted. To handle such scenarios requires a relaxed state machine on application level to handle multiple reports. The same apply for set and get commands.

4.10.2.4 Restrictions on functions using buffers

The Serial API is implemented with buffers for queuing requests and responses. This restricts how much data that can be transferred through MemoryGetBuffer() and MemoryPutBuffer() compared to using them directly from the Z-Wave API.

The PC application should not try to get or put buffers larger than approx. 80 bytes.

If an application requests too much data through MemoryGetBuffer() the buffer will be truncated and the application will not be notified.

If an application tries to store too much data with MemoryPutBuffer() the buffer will be truncated before the data is sent to the Z-Wave module, again without the application being notified.

4.10.2.5 Serial API Node List Command

As of Serial API protocol version 4 it is possible to call Serial API Node List Command to determine Serial API protocol version number, Serial API capabilities, nodes currently stored in the EEPROM (only controllers) and chip used in a specific Serial API Z-Wave Module with the FUNC_ID_SERIAL_API_GET_INIT_DATA Serial API function:

Serial API:

HOST->ZW: REQ | 0x02

(Controller) ZW->HOST: RES | 0x02 | ver | capabilities | 29 | nodes[29] | chip_type | chip_version

(Slave) ZW->HOST: RES | 0x02 | ver | capabilities | 0 | chip_type | chip_version

"ver" is the Serial API application Version number.

"capabilities" is a byte holding various flags describing the actual mode.

29 | 0 is the length of "nodes[]"

nodes[29] is a node bitmask.

Capabilities flag:
Bit 0: 0 = Controller API; 1 = Slave API
Bit 1: 0 = Timer functions not supported; 1 = Timer functions supported.
Bit 2: 0 = Primary Controller; 1 = Secondary Controller
Bit 3-7: reserved

The chip used can be determined as follows:

Z-Wave Chip	Chip_type	Chip_version
ZW0102	0x01	0x02
ZW0201	0x02	0x01
ZW0301	0x03	0x01

Timer functions are TimerStart, TimerRestart and TimerCancel.

4.10.2.6 Serial API Capabilities Command

As of Serial API protocol version 4 (to determine Serial API protocol version please refer to the Serial API Function described in paragraph 4.10.2.5) it is possible to call Serial API Capabilities Command to determine exactly which Serial API functions a specific Serial API Z-Wave Module supports with the FUNC_ID_SERIAL_API_GET_CAPABILITIES Serial API function:

Serial API:

HOST->ZW: REQ | 0x07

ZW->HOST: RES | 0x07 | SERIAL_APPL_VERSION | SERIAL_APPL_REVISION |
SERIALAPI_MANUFACTURER_ID1 | SERIALAPI_MANUFACTURER_ID2 |
SERIALAPI_MANUFACTURER_PRODUCT_TYPE1 |
SERIALAPI_MANUFACTURER_PRODUCT_TYPE2 |
SERIALAPI_MANUFACTURER_PRODUCT_ID1 | SERIALAPI_MANUFACTURER_PRODUCT_ID2 |
FUNCID_SUPPORTED_BITMASK[ ]

SERIAL_APPL_VERSION is the Serial API application Version number.

SERIAL_APPL_REVISION is the Serial API application Revision number.

SERIALAPI_MANUFACTURER_ID1 is the Serial API application manufacturer_id (MSB).

SERIALAPI_MANUFACTURER_ID2 is the Serial API application manufacturer_id (LSB).

SERIALAPI_MANUFACTURER_PRODUCT_TYPE1 is the Serial API application manufacturer product type (MSB).

SERIALAPI_MANUFACTURER_PRODUCT_TYPE2 is the Serial API application manufacturer product type (LSB).

SERIALAPI_MANUFACTURER_PRODUCT_ID1 is the Serial API application manufacturer product id (MSB).

SERIALAPI_MANUFACTURER_PRODUCT_ID2 is the Serial API application manufacturer product id (LSB).

FUNCID_SUPPORTED_BITMASK[ ] is a bitmask where every Serial API function ID which is supported has a corresponding bit in the bitmask set to '1'. All Serial API function IDs which are not supported have their corresponding bit set to '0'. First byte in bitmask corresponds to FuncIDs 1-8 where bit 0 corresponds to FuncID 1 and bit 7 corresponds to FuncID 8. Second byte in bitmask then corresponds to FuncIDs 9-16 and so on.

4.10.2.7 Serial API Power Management Commands

The Serial API Power Management Commands is designed for use in a system where a Z-Wave module is connected to a host CPU system via a serial port and a number of I/O pins are used for control of the power to the Host CPU system.

4.10.2.7.1 Pin Configuration Command

The Pin Configuration Command is used to map the power management input pin PoweredUp to a physical IO pin.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_PIN_UP_CONFIGURATION_CMD							
IO Pin							
Active Level							

IO pin (8bit):

The IO pin field specifies the physical I/O pin that should be used for this signal. The table of I/O pins is shown below

IO Pin defines	Value
PM_PHYSICAL_PIN_P00	0x00
PM_PHYSICAL_PIN_P01	0x01
PM_PHYSICAL_PIN_P02	0x02
PM_PHYSICAL_PIN_P03	0x03
PM_PHYSICAL_PIN_P04	0x04
PM_PHYSICAL_PIN_P05	0x05
PM_PHYSICAL_PIN_P06	0x06
PM_PHYSICAL_PIN_P07	0x07
PM_PHYSICAL_PIN_P10	0x10
PM_PHYSICAL_PIN_P11	0x11
PM_PHYSICAL_PIN_P12	0x12
PM_PHYSICAL_PIN_P13	0x13
PM_PHYSICAL_PIN_P14	0x14
PM_PHYSICAL_PIN_P15	0x15
PM_PHYSICAL_PIN_P16	0x16
PM_PHYSICAL_PIN_P17	0x17
PM_PHYSICAL_PIN_P22	0x22
PM_PHYSICAL_PIN_P23	0x23
PM_PHYSICAL_PIN_P24	0x24
PM_PHYSICAL_PIN_P30	0x30
PM_PHYSICAL_PIN_P31	0x31
PM_PHYSICAL_PIN_P32	0x32
PM_PHYSICAL_PIN_P33	0x33
PM_PHYSICAL_PIN_P34	0x34
PM_PHYSICAL_PIN_P35	0x35
PM_PHYSICAL_PIN_P36	0x36
PM_PHYSICAL_PIN_P37	0x37

Active Level (8bit):

The level the PoweredUp pin should have when it is active. Optional and not given then active state defaults to active Low.

0 – Low

1 – High

4.10.2.7.2 Power up Mode Configuration Command

The Power up Mode Configuration Command is used to configure the state of the PowerCtrl pins when the Serial API has to power up the host CPU system

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_MODE_CONFIGURATION_CMD							
Number of Pins (max 4)							
IO Pin 1							
Level 1							
IO Pin ..							
Level ..							
IO Pin x							
Level x							

Number of Pins (8 bit):

The number of pins that is contained in the command. The max number of pins is 4

IO Pin x (8 bit):

The physical pin that should be changed when the Serial API has to wake up the host CPU system. A full list of physical pins can be found in section 4.10.2.7.1.

Level x (8 bit):

The level the output pin should have when the specified power mode is set.

0 – Low

1 – High

4.10.2.7.3 Power Up on Z-Wave Configuration Command

The Power Up on Z-Wave Configuration Command is used to specify what Z-Wave command that should trigger a power up of the host CPU system. All Z-Wave commands received are checked if they match the wakeup values and masks configured.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_ZWAVE_CONFIGURATION_CMD							
Wakeup Match Mode							
Number of match bytes (max 8)							
Wakeup Value 1							
Wakeup Value ..							
Wakeup Value x							
Wakeup Mask 1							
Wakeup Mask ..							
Wakeup Mask x							

Wakeup Match Mode (8bit):

PM_WAKEUP_ALL

Wake up on all Z-Wave application commands received by the Z-Wave module.

PM_WAKEUP_ALL_NO_BROADCAST

Wake up on all Z-Wave application commands received by the Z-Wave module, except frames send as broadcast frames.

PM_WAKEUP_MASK

Wake up the host CPU when receiving a Z-Wave command where the first 5 bytes of the frame matches the specified value and mask.

Wakeup Mode define	Value
PM_WAKEUP_ ALL	0x01
PM_WAKEUP_ALL_NO_BROADCAST	0x02
PM_WAKEUP_MASK	0x03

Number of Match Bytes (8bit):

Number of bytes used to match an incoming Z-Wave command with, to see if it should trigger a wakeup. The max number of match bytes is 8.

Wakeup Value n (8bit*x):

The wakeup value is the value an incoming Z-Wave frame should be checked against to see if it should trigger a wakeup.

Wakeup Mask n (8 bit*x):

The wakeup mask is a mask that can be used to mask out bits or bytes in the received Z-Wave frame before it is compared with the Wakeup value.

The Wakeup value and Wakeup mask are checked like this in the Serial API

```
If ((Z-Wave Frame & Wakeup Mask) == Wakeup Value)

        DoWakeup();
```

Example:

If the host CPU wants to trigger a wakeup on an Simple AV Set command with the Command Power the following command should be send to the Z-Wave module.

The simple AV Set command has the following structure:

7	6	5	4	3	2	1	0
COMMAND_CLASS_SIMPLE_AV_CONTROL							
SIMPLE_AV_CONTROL_SET							
Sequence Number							
Reserved					Key Attributes		
Item ID MSB							
Item ID LSB							
AV Command MSB,1							
AV Command LSB,1							

In this Z-Wave command we want to match the command class, the command, the key attributes and the AV command. We do not care about the sequence number, the reserved field and the item ID. So the Power Up on Z-Wave command would look like this:

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_ZWAVE_CONFIGURATION_CMD							
PM_WAKEUP_MASK							
8 (Match the 8 first bytes)							
COMMAND_CLASS_SIMPLE_AV_CONTROL							
SIMPLE_AV_CONTROL_SET							
0 (don't care)							
0 (key down)							
0 (don't care)							
0 (don't care)							
0 (AV Command MSB)							
0x27 (AV command Power)							
0xFF (match all bits)							
0xFF (match all bits)							
0x00 (don't match)							
0x07 (match bits 0,1,2)							
0x00 (don't match)							
0x00 (don't match)							
0xFF (match all bits)							
0xFF (match all bits)							

4.10.2.7.4 Power Up on Timer Configuration Command

The Power Up on Timer Configuration Command is used to specify that the Z-Wave module should power up the host CPU system after a specified time has passed.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_TIMER_CONFIGURATION_CMD							
Timer Resolution							
Timer (MSB)							
Timer (LSB)							

Timer Resolution (8bit):

PM_TIMER_SECONDS	The timer resolution is in Seconds.
PM_TIMER_MINUTES	The timer resolution is in minutes.

Timer Resolution define	Value
PM_TIMER_SECONDS	0x01
PM_TIMER_MINUTES	0x02

Timer (16bit):

The time that should elapse before the host CPU is set to the POWER_MODE_RUNNING again

4.10.2.7.5 External Power Up Configuration Command

The External Power Up Configuration Command is used to specify that a level change on an input pin should trigger a power up of the host CPU system.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERUP_EXTERNAL_CONFIGURATION_CMD							
IO Pin							
Power Up Level							

IO pin (8bit):

The IO pin field specifies the physical I/O pin that should be used for this signal. The full table of I/O pins can be found in section 4.10.2.7.1

Power Up Level (8bit):

The level the input pin should trigger a power up of the host CPU system.

0 – Low

1 – High

4.10.2.7.6 Power down Mode Configuration Command

The Power down Mode Configuration Command is used to request that the Z-Wave module sets a specific power down mode. If the PoweredUp pin is configured the PowerCtrl pins will not be changed before the PoweredUp pin goes NOT active.

7	6	5	4	3	2	1	0
FUNC_ID_POWER_MANAGEMENT							
PM_POWERDOWN_MODE_CONFIGURATION_CMD							
Number of Pins (max 4)							
IO Pin 1							
Level ..							
IO Pin x							
Level 1							
IO Pin ..							
Level x							

Number of Pins (8 bit):

The number of pins that is contained in the command. The max number of pins is 4

IO Pin x (8 bit):

The physical pin that should be changed when the Serial API powers down the host CPU system. A full list of physical pins can be found in section 4.10.2.7.1.

Level x (8 bit):

The level the output pin should have when the specified power mode is set.

0 – Low

1 – High

4.10.2.8 Serial API Ready Command

The Ready Command is used by the host to inform the Z-Wave module that it is ready to receive command on the UART.

7	6	5	4	3	2	1	0
FUNC_ID_READY							
[SerialLinkState]							

SerialLinkState (8 bit):

Set the Serial link state between HOST and the SerialAPI Z-Wave module.

SERIAL_LINK_DETACHED – The Serial link state should be DETACHED or SerialAPI stops sending data to HOST until either READY is transmitted again in connected state or any valid SerialAPI command is received from HOST.

SERIAL_LINK_CONNECTED – The Serial link state should be CONNECTED or SerialAPI sends data to HOST when needed.

The SerialAPI Z-Wave module starts up after reset in the Serial link state DETACHED.

SerialLinkState define	Value
SERIAL_LINK_DETACHED	0x00
SERIAL_LINK_CONNECTED	0x01

4.10.2.9 Serial API Softreset Command

The host CPU system can make a software reset of the Z-Wave module by using the Softreset Command.

7	6	5	4	3	2	1	0
FUNC_ID_SERIAL_API_SOFT_RESET							

4.10.2.10 Serial API Watchdog Commands

Some PC based applications cannot guarantee kicking the watchdog before timeout causing the watchdog to reset the Z-Wave ASIC unintentionally. The following Watchdog Commands are therefore available to avoid this:

- Start watchdog: Enable watchdog and ApplicationPoll kick watchdog
- Stop watchdog: Disable watchdog and stop kick watchdog in ApplicationPoll

Watchdog handling disabled when powered up and Sleep/FLiRS mode will temporary stop watchdog.

The host CPU system can start watchdog functionality by using the Serial API function FUNC_ID_ZW_WATCHDOG_START:

7	6	5	4	3	2	1	0
FUNC_ID_ZW_WATCHDOG_START							

The host CPU system can stop watchdog functionality by using the Serial API function FUNC_ID_ZW_WATCHDOG_STOP:

7	6	5	4	3	2	1	0
FUNC_ID_ZW_WATCHDOG_STOP							

4.10.2.11 Serial API Files

The Product\SerialAPI directory contains sample source code for controller/slave applications on a Z-Wave module. The application uses also a number of utility functions described in section 3.3.11.

MK.BAT

Make bat file for building the sample application in question. To only build applications using EU frequency enter: **MK "FREQUENCY=EU"** in command prompt.

Makefile

This is the Makefile for the sample application in question defining the targets built. Refer to section 3.2.1.1 for additional details.

Makefile.common_ZW0x0x_supported_functions

This makefile makes a text file showing the supported serial API functions for the given target.

MakePatch.bat

Make hex files for patch system including the <appl>_ZW040x_<freq>_devmode_patch_RAM.hex targeted for SRAM when using development mode.

Config_app.h

This header file contains defines for application version.

UART_buf_io.h / UART_buf_io.c

Low level routines for handling buffered transmit/receive of data through the UART.

conhandle.h / conhandle.c

Routines for handling Serial API protocol between PC and Z-Wave module.

serialappl.h / serialappl.c

This module implements the handling of Serial API protocol. That is, parses the frames, calls the appropriate Z-Wave API library functions and returns results etc. to the PC. Enable/disable support of a given Serial API function in serialappl.h header file.

serialappl_patch.c

This file contains the patched source code of serialappl.c

SerialAPI_Ctl_Bridge_ZW040x.mpw / SerialAPI_Ctl_Bridge_ZW040x_....Uv2
SerialAPI_Ctl_Installer_ZW040x.mpw / SerialAPI_Ctl_Installer_ZW040x_....Uv2
.
.
.
SerialAPI_ Slave_Routing_ZW040x_....Uv2

uVision4 *.Uv2 project files created by makefile system using uVisionProjectGenerator software.

Supported.bat

Batch file called by Makefile.common_ZW0x0x_supported_function to obtain delayed environment variable expansion when using SET in DOS prompt.

make-supported-functions-include.bat

Windows batch script for generating SerialAPI defines for supported functions based on what exists in library.

serialapi-supported-func-list.txtt

Template file for generating SerialAPI defines for supported functions based on what exists in library. Enable/disable support of a given Serial API function in serialappl.h header file.

4.10.3 Power management

4.10.3.1 System overview

The power management API is designed for use in a system where a Z-Wave module is connected to a host CPU system via a serial port and a number of I/O pins are used for control of the power to the Host CPU system.

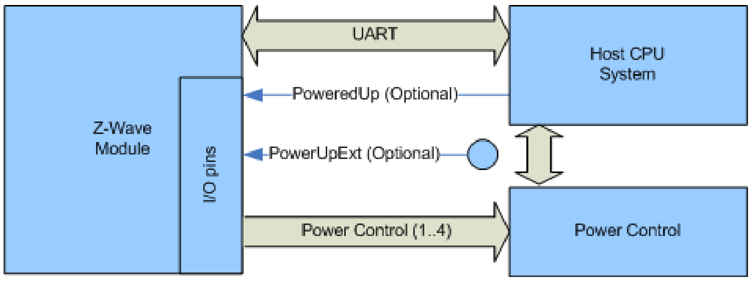


Figure 2, Power Management system

In a system like this it is necessary to have a communication protocol between the two CPU systems that ensures that the correct power state is selected and the Z-Wave module and the host CPU system always is in agreement about what power state they are using at all times.

All power management configuration and setup is done runtime using the serial API interface from the host processor system. The Z-Wave module must therefore be powered at all times in the system and decisions to power down the system always comes from the host CPU system. Power management is also possible on a Z-Wave module without external non-volatile memory.

4.10.3.2 I/O pins

A number of I/O pins on the Z-Wave module and the host processor system can be used for the power management API. No GPIO pins will be configured or changed before the host CPU configures the pin. All GPIO pins will be in their reset state (input, pull up enabled) until the host CPU issues an serial API command that configures or change status of a pin.

All GPIO's used as input on the Z-Wave module must be asserted for at least 20ms when changing level to allow the firmware to detect the change of the input pin status.

PoweredUp pin (Optional)

An input pin on the Z-Wave module is needed to communicate from the host processor to the Z-Wave module that the host processor system is now ready to be powered down. This pin is necessary if the host CPU system is not able to send commands on the UART during the power down sequence because the UART driver or the OS has been stopped. If configured the PoweredUp pin is set active on system power on.

Z-Wave module	Input
Host CPU	Output

PowerCtrl(1..4)

The PowerCtrl pins are used to control the power management hardware from the Z-Wave module.

Z-Wave	Output
Host CPU	N/A

4.10.3.3 Power management configuration sequence

When the serial API starts up for the first time it assumes that there is no power management present. The power management is activated in the Z-Wave module by configuring the power up mode.

See section 4.10.2.7 for a detailed description of the serial API commands.

When configuring the power management the following sequence of events should happen:

- The host configures the PoweredUp pin by using the Serial API Power Management Pin Configuration command. (Optional)
- The host configures the Power Up PowerCtrl pin(s) by using the Serial API Power Management Pin Configuration command
- The host configures the Wake up criteria's by using the Power Up on Z-Wave Configuration Command (see section 4.10.2.7.3) and/or the Power Up on Timer Configuration Command (see section 4.10.2.7.4).

4.10.3.4 Power up sequence

When powering up the following sequence of events should happen:

1. The Z-Wave module receives a command via RF that triggers a power up of the system.
2. The Z-Wave module changes the state of the power control I/O pins to the POWER_MODE_RUNNING state
3. The Z-Wave module waits for the Serial API Ready command on the UART
4. The host CPU system powers up and sets the PoweredUp pin active. (Optional)
5. When ready the host CPU system sends the serial API Ready command.
6. When the Ready command is received the Z-Wave module sends the command that triggered the power up to the host CPU system.

4.10.3.5 Power down sequence

When powering down the following sequence of events should happen:

1. The host must have performed the configuration sequence specified in section 4.10.3.3
2. The host processor determines that the system should power down now (based on, activity, timer, received commands, etc.)
3. The host processor sends an Serial API Set Power Mode command to the Z-Wave module
4. The Z-Wave module starts to monitor the PoweredUp pin (if configured) and continues to next state in power down sequence when the PoweredUp pin goes NOT active.

5. The Z-Wave module changes the state of the power control I/O pins according to the power mode requested by the host.

4.10.3.6 Power modes

The power management API supports any number of power modes that the host CPU system wants to use. The power modes can be divided into 2 different groups:

POWER_MODE_RUNNING

In power mode running the host CPU system is running. The host CPU system can receive commands send from the Z-Wave module on the UART.

POWER_MODE_POWERDOWN

In power mode power down the host CPU system is unable to receive commands send on the UART. All Z-Wave RF commands received by the Z-Wave module will be discarded if they do not trigger a wakeup. The only transition of power mode from this mode it to go to the POWER_MODE_RUNNING.

4.11 PC based Controller Sample Application

The PC\Source\SampleApplications\ZWavePCController directory contains sample application source code in C# that implements a PC based Controller using the development tool Visual Studio 2008.

For further information about the features of the PC based Controller, see [1].

4.12 PC based Installer Tool Sample Application

The PC\Source\SampleApplications\ZWaveInstaller directory contains sample application source code in C# that implements a PC based Installer Tool using the development tool Visual Studio 2008.

For further information about the features of the PC based Installer Tool, see [2].

4.13 PC based Z-Wave Bridge Sample Application

The PC\Source\SampleApplications\ZWaveUPnPBridge directory contains sample application source code in C# that implements a PC based Z-Wave to UPnP Bridge using the development tool Visual Studio 2008.

The Z-Wave to UPnP bridge sample application contains UPnP.dll and UPnP_AV.dll from http://opentools.homeip.net/dev-tools-for-upnp

For further information about the features of the PC based Z-Wave to UPnP Bridge, see [3].

5 TOOL SAMPLE CODE

The Z-Wave Developer's Kit includes tool sample code to enable customization of production environment.

5.1 Z-Wave Programmer Firmware

The SDK contains sample code that demonstrates how to program the 100/200/300/400 Series ASIC. The ZDP03A Z-Wave Development Platform [14] supports this purpose. The Z-Wave Programmer firmware resides on the AVR ATmega128 chip on ZDP03A and controlled by the PC based Z-Wave Programmer application [7]. For a detailed description of the communication protocol between the AVR and PC based Z-Wave Programmer application, refer to [12].

Source code developed in the following environment:

- WinAVR v20071221:
  - GNU Binutils 2.18 (including assembler, linker, etc.)
  - Compiler Collection (GCC) 4.2.2
  - avr-libc 1.6.0
- Z-Wave Library v2.91
- Keil uVision PK51 v8

Project environment:

- Eclipse Platform v3.5 with plugins:
  - AVR Eclipse Plugin
  - (optional) Polarion Subversive SVN Connectors
  - (optional) Eclipse Subversive - SVN Team Provider Project

The AVR ISP In-System Programmer programs the AVR Atmega128.

5.1.1 ATmega_ZWaveProgFW Files

The Tools\Programmer\ATmega_ZWaveProgFW directory contains the source code for the 400 Series low level programming application.

MK.BAT

Batch file used to build AVR based sample applications in versions for the firmwave update (via Z-Wave Programmer) and complete ATMega128 firmware (via AVR ISP In-System Programmer).

MAKE_FIRMWARE.BAT

Batch file used to make complete ATMega128 firmware from bootloader firmware and firmware update. Called by MK.BAT.

MAKE_MTP.BAT

Batch file used to build the ZW040x Execute Out of SRAM application, that give the ability to the ATMega128 firmware to access the MTP memory of the ZW040x chip. Called by MK.BAT.

.cproject; .project; .settings

Project files of the Eclipse IDE used to edit AVR based sample application source code.

src\ATMega_spi.c; .h

Source code of the implementation of the software SPI, which is connected to the Z-Wave Module.

src\commands.h

This header file contains definitions of the commands of the Z-Wave Programmer Communication Protocol [12].

src\conhandle.c; .h

Source files, contains the functions for handling the Programmer frames via the UART.

src\eeprom_if.c; .h

Source code of the Z-Wave Module External non-volatile memory interface. Reading / writing of the Z-Wave Module External non-volatile memory via the software SPI was implemented.

src\mtp.c; .h

Source code of the ZW040x Execute out of SRAM application, which implements the ZW040x MTP memory interface.

src\ports.h

Header file with definitions of port names of the ATMega128 in ZDP03 (ZDP02) board.

src\UART_buf_io.c; .h

Source code of buffered transmit/receive of data through the UART.

src\ZWaveFlash.c; .h

Main source code of the Z-Wave Programmer Firmware. Contains the implementation of all programmer commands handlers and Z-Wave chips programming algorithms.

6 REQUIRED DEVELOPMENT COMPONENTS

6.1 Software development components

There is an additional 3rd party software tool that is required to develop Z-Wave applications that is not supplied with the Z-Wave Developer's Kit. That is the Keil PK51 v9.0 Professional Developer's Kit for the 8051 microcontroller:

Z-Wave libraries and sample applications are built and tested on version 9.02a but newer versions should also apply according to Keil's recommendations. It is not possible to use earlier Keil PK51 versions than v9.0 in connection with this SDK because Keil changed object format.

The Keil Developer's Kits can be purchased directly from Keil or from one of their local distributors. Please visit **www.keil.com** for details. Alternatively can it be purchased from Sigma Designs.

Keil Software, Inc. 1501 10th Street, Suite 110 Plano, TX 75074 USA	
Toll Free:	**800-348-8051**
Phone:	972-312-1107
Fax:	972-312-1159
Sales:	**sales.us@keil.com**
Support:	**support.us@keil.com**

Keil Elektronik GmbH Bretonischer Ring 15 D-85630 Grasbrunn Germany	
Toll Free:	**-**
Phone:	(49) (089) 45 60 40 0
Fax:	(49) (089) 46 81 62
Sales:	**sales.intl@keil.com**
Support:	**support.intl@keil.com**

6.2 100/200/300/400 Series ASIC programmer

This Z-Wave Developer's Kit comes with the Z-Wave Programmer included. The Z-Wave Programmer is used for downloading new firmware to the Z-Wave ASIC. The Z-Wave Programmer is also used when setting lock bits, programming the external non-volatile memory on the Z-Wave module etc.

For a detailed description of the ZDP03A Z-Wave Development Platform refer to [14].

6.3 Hardware development components for 400 Series

The 400 Series based embedded sample application are designed for the ZDP03A Z-Wave Development Platform in combination with a Z-Wave module hosting the sample application. Some applications use also the AVR processor on ZDP03A as host together with a serial API application running on the Z-Wave module. The Z-Wave modules exist in two variants:

- ZM4101 Z-Wave Module [15] mounted on a ZM4125 Z-Wave Module [17].
- SD3402 Z-Wave ASIC [16] mounted on a ZM4225 Z-Wave Module [18].

REFERENCES

[1] SD, INS10240, Instruction, PC Based Controller User Guide.
[2] SD, INS10241, Instruction, PC Installer Tool Application User Guide.
[3] SD, INS10245, Instruction, Z-Wave Bridge User Guide.
[4] SD, INS10336, Instruction, Z-Wave Reliability Test Guideline.
[5] SD, INS10249, Instruction, Z-Wave Zniffer User Guide.
[6] SD, INS10250, Instruction, Z-Wave DLL User's Manual.
[7] SD, INS10679, Instruction, Z-Wave Programmer User Guide.
[8] SD, INS10236, Instruction, Development Controller User Guide.
[9] SD, INS10326, Instruction, ZW0201 Single Chip Implementation Guidelines.
[10] SD, INS10680, Instruction, Z-Wave XML Editor.
[11] SD, INS10681, Instruction, Secure Development Controller (AVR) User Guide.
[12] SD, INS11072, Instruction, Z-Wave Programmer Communication Protocol.
[13] SD, INS10326, Instruction, ZW0201 Single Chip Implementation Guidelines.
[14] SD, DSH11243, Datasheet, ZDP03A Z-Wave Development Platform.
[15] SD, DSH11055, Datasheet, ZM4101 Module / ZW0401 Single Chip.
[16] SD, DSH11036, Datasheet, SD3402 Datasheet.
[17] SD, DSH11307, Datasheet, Z-Wave ZM4125 Module
[18] SD, DSH11306, Datasheet, Z-Wave ZM4225 Module.
[19] SD, INS12034, Instruction, Z-Wave 400 Series Application Programming Guide v6.01.03.
[20] SD, SDS10242, Software Design Specification, Z-Wave Device Class Specification.
[21] SD, SDS11060, Software Design Specification, Z-Wave Command Class Specification.
[22] SD, INS11596, Instruction, Micro RF Link Tool.
[23] SD, INS11709, Instruction, Working in 400 Series Environment User Guide.
[24] SD, INS11552, Instruction, 400 Series Crystal Calibration User Guide.
[25] SD, INS12131, Instruction, Micro PVT Tool.

INDEX