Software Release Note

Z-Wave 500 Series SDK v6.81.06

Document No.:	SRN13926
Version:	11
Description:	-
Written By:	JFR;COLSEN;PSH;JSI;SSE;EFH;NTJ;BBR
Date:	2019-07-19
Reviewed By:	BBR;NTJ;JFR;JRM;JOPEDERSEN;HAKRONER;ABXAVIER
Restrictions:	Public

Approved by:

Date	CET	Initials	Name	Justification
2019-07-19	03:33:21	JFR	Jorgen Franck	on behalf of NTJ

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20170720	JFR	All	Initial draft based on SRN13389-6 – Z-Wave 500 Series SDK v6.71.01
1	20170922	PSH ABR JFR	Section 1, 2, 3 & 4	Changed document to describe new features in SDK v6.80.00
2	20170927	JFR	Section 2	Updated uVision Project Generator to v1.16
3	20170927	JFR	Section 1, 2, 3 & 4	Updated to SDK v6.81.00
4-5	20180322	JFR	Section 1 & 2	Updated to SDK v6.81.01
6	20180326	BBR	All	Added Silicon Labs template
7	20180531	JFR	Section 1 & 2	Updated to SDK v6.81.02
8	20181008	JFR	Section 1 & 2	Updated to SDK v6.81.03
9	20190123	JFR	Section 1 & 2	Updated to SDK v6.81.04
10	20190412	JFR	Section 1, 2, 4.1 & 4.2	Updated to SDK v6.81.05
11	20190717	JFR	Section 1 & 2	Updated to SDK v6.81.06

Table of Contents

List of Figures

1 INTRODUCTION

This chapter provides an introduction to the SDK as well the Z-Wave technology.

1.1 Introduction to the SDK

The Z-Wave 500 Series Software Development Kit (SDK) is intended to help developers creating Z-Wave Plus compliant products in a fast and cost-effective manner. The software consists of Z-Wave libraries supporting controller/slave devices and a Z-Wave Plus Application Framework, as well as code for a broad range of home automation applications. To realize a specific application, it is likely easier to modify an existing sample app instead of using MyProduct app as a starting point.

The Z-Wave 500 Series SDK version 6.81.06 is a mature release enabling support of Smart Start, etc. This release is intended for Z-Wave certified 500 Series based products entering volume production. All the Z-Wave Plus Applications are Z-Wave certified. For details regarding functionality, refer to Chapter 3 and 4. Finally, refer to [11] for a detailed description of contents.

Notice: Be aware that Z-Wave SDK 6.8x.xx requires Keil PK51 v9.54A.

1.1 Abbreviations

Abbreviation	Explanation
ACK	Acknowledge
API	Application Programming Interface
C	Command
CC	Command Class
CSA	Client-Side Authentication
DUT	Device Under Test
ID	Identifier
FLiRS	Frequently Listening Routing Slave. Communication to a FLiRS node can be established by a wakeup beam
NIF	Node Information Frame
NWI	Network Wide Inclusion (add node out of direct range)
NWE	Network Wide Exclusion (remove node out of direct range)
OTA	Over The Air (e.g. making a firmware update wireless)
OTW	Over The Wire (e.g. making a firmware update via the serial API interface)
S0	Security 0 Command Class
S2	Security 2 Command Class
SDK	Software Development Kit
SSA	Server-Side Authentication
TO	Test Observation (bug)

1.2 Introduction to the Z-Wave technology

Z-Wave is a wireless mesh protocol oriented to the residential control and automation market, but it may also be used in light commercial environments. The technology provides a simple yet reliable method to wirelessly control lights and A/V equipment in your house. Z-Wave works in the unlicensed industrial, scientific, and medical (ISM) bands around 900MHz. Regional frequencies vary slightly. Each Z-Wave network may comprise up to 232 nodes. Nodes may retransmit a message in order to guarantee delivery. The typical communication range between two nodes is 100 feet.

The Z-Wave eco system offers a routing protocol stack and a complete Z-Wave Plus Application Framework of device types and command classes for interoperable deployments. Interoperability is ensured between all device types thanks to the Z-Wave certification program. The Z-Wave logo is only granted to products passing certification.

1.2.1 Protocol Stack Overview

Z-Wave offers a routing protocol that reliably transfers messages up to 5 hops away, i.e., up to 500 feet. The protocol stack comprises a PHY/MAC layer to control access to RF media, a transport layer to handle frame integrity and retransmissions, and a network layer with all its routing magic and application interfaces.

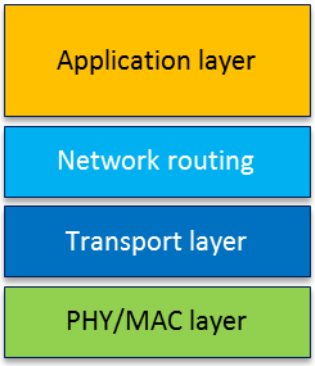

Figure 1. Z-Wave Protocol Stack

The maximum size of payload data is 46 bytes when routing is used. The Z-Wave protocol uses standard collision-avoidance methods—postponing a transmission a random number of milliseconds when media is busy. The Z-Wave transport layer controls the transfer of data between two nodes including acknowledgement and optional retransmission.

Multicast and broadcast may only be used in direct range. Broadcast and multicast may be used to reach more than one destination address. In case of multicast, the same payload will be delivered to selected nodes only.

The Z-Wave application layer is responsible for handling application commands. Commands are divided into two classes: Z Wave protocol and application-specific. Most protocol-related operations are just address assignment logic, but commands that are more complex are defined for advanced network management operations.

Each Z-Wave network has a unique 32-bit identifier called Home ID. Every new node joining the network inherits the same Home ID from the primary controller. Individual nodes in the network are addressed using an 8-bit Node ID that is unique within the network.

1.2.2 Classic Z-Wave

The following text makes references to classic nodes. In short, the term “Classic Z-Wave node” covers previous generations of Z-Wave nodes which do not implement recently introduced features such as Network-Wide Inclusion (NWI), Dynamic Route Resolution, and FLiRS communication.

1.2.3 Node Types

There are two main types of devices: controllers and slaves. Controllers can handle network management and communication to classic nodes. Slaves provide no network management capability.

1.2.3.1 Controllers

A controller node maintains a routing table for all operational links in the network. This table allows the controller to calculate routes between any two nodes in the network. The primary controller may refresh the routing table and distribute updated routing tables to other controllers.

Controllers come in three variants: portable, static, and bridge.

The portable controller is optimized for battery operation. It is typically used for remote control devices.

The static controller is intended for mains-powered control panels, gateways, or network managers. The static controller may also act as repeater for other nodes.

1.2.3.2 Slaves

A slave device has simpler functionality than a controller has and can be implemented without any external non-volatile storage. It may repeat a message for other nodes.

A special variant, the WakeUp slave, may be used for sensor-style devices such as alarms and sensors.

Referred to as duty cycling in the literature, the Frequently Listening Routing Slave (FLiRS) wakes up in fixed intervals to listen very shortly for a preamble pattern. This enables the design of products with battery lifetimes measured in years. Yet, it is possible to reach such devices on short notice.

WakeUp slaves and FLiRS nodes cannot operate as repeaters as they are sleeping most of the time to conserve battery.

1.2.4 Network Operation

Management of Z-Wave nodes constitute two main operations: inclusion/exclusion and association. Inclusion adds a new node to the network. Exclusion removes a node. Only primary controllers can include and exclude nodes.

Association is the creation of a logical connection between applications. In other words, it defines what controls what. Association is handled by the application layer.

1.2.5 Routing Principles

Z-Wave uses source routing to reach a destination. Source routing allows implementation of a lightweight protocol, thereby avoiding distributed routing tables in all repeaters. This puts a limit to the length of routes. Real-world deployments indicate that residential networks rarely have more than 2-hop routes. Z-Wave's support for 5-hop routes is enough and an efficient compromise.

The route is carried in the routing header, and every repeater forwards the frame according to the routing header. Only always-listening nodes can participate in routing, but routing may also be used to reach FLiRS nodes.

Network Wide Inclusion (NWI) allows a user to include a new node even though the new node is not within range of the primary controller. Dynamic route resolution allows a node to repair broken routes during normal operation. Classic nodes do not support NWI and dynamic route resolution.

Network Wide Exclusion (NWE) uses the same explorer strategy as Network Wide Inclusion (NWI) to accomplish an out-of-range exclusion of nodes from the network. It is also possible to remove a specific node from the network by specifying the Node ID.

1.2.6 Application Development

Depending on node type functionality (such as controller vs. slave), developers may choose from a selection of libraries. On top of the chosen library, an application designer may choose from a wide range of Command Classes, including light control, sensors, garage port control, and many others. Command Classes are a collection of functionally related commands. A device may implement several functions and therefore support more Command Classes.

Z-Wave applications are designed as a state machine periodically polled from the Z-Wave library. This allows for the design of products with fewer CPU resources than typically required for OS's with threads, tasks, priorities, etc. This again translates into inexpensive products suited for mass production. Depending on the actual product, an application may interface to the Z-Wave protocol stack in three ways:

Most constrained devices, like a light dimmer with one button, may have its application running in the on-chip 8051 MCU. In this configuration, the Z-Wave API is used directly via function calls provided by the binary image implementing the Z-Wave library.

Larger devices, like a remote control with display, may have its own host processor. The application designer may prefer to implement all application logic in the host processor, only running the Z-Wave protocol stack in the on-chip 8051 MCU. The Z Wave Serial API provides an abstracted version of the Z-Wave API that is accessed via an on-chip serial port. The application design principle for the Z-Wave part should still be a state machine that reacts to incoming events, callback functions, and timeouts.

Most advanced devices like IP gateways and PC-based light control servers may use an even more abstracted API provided via the Network Management Command Class. In this model, all communication is carried in IP packets. The Z/IP Gateway library provides this mapping.

1.2.7 Managing Interoperability

Interoperability is a key part of the Z-Wave eco system. Every product must pass certification to be granted the Z-Wave logo. The Z-Wave Alliance manages the Z-Wave certification program, but certification testing is performed by independent test houses. Certification makes sure that a product correctly implements all device and command classes that it claims to support.

2 RELEASED VERSIONS

2.1 SDK 6.81.06

```
Z-Wave Framework and Certified Applications.................. v4_03_00
Z-Wave Protocol and Serial API Applications.................. v6_07_00
Z-Wave Serial API Application Interface.......................... v8

Tools
=====
IMA Tool..................................................... v0_99
NVM Converter................................................ v0_07
uVision Project Generator.................................... v1_16
```

3 OVERVIEW

The Z-Wave 500 Series SDK version 6.8x contains the following major enhancements compared to SDK version 6.7x:

- Smart Start
- FLiRS multicast
- Improved power management
- Improved Application Framework

4 DETAILED DESCRIPTION

4.1 Re-certified Apps (SDK 6.80.05+)

The five apps (Door Lock Key pad, Sensor PIR, Switch On/Off, Power Strip and Wall Controller) have been re-certified to ensure full compliance.

Notice: Certification does not include Z-Wave Plus v2 requirements.

4.2 NVM Converter (SDK 6.80.05+)

The NVM Converter Tool converts the NVM based protocol data for a given Z-Wave network to a newer SDK version. Typically, used when upgrading a Gateway using a bridge controller based serial API to a newer SDK version to avoid re-installation of the Z-Wave network. From SDK 6.51.00+ the conversion process is performed automatically during an OTA/OTW firmware update.

4.3 Smart Start (SDK 6.80.00+)

Z-Wave SmartStart aims to shift the tasks related to inclusion of an end device into a Z-Wave network away from the end device itself, and towards the more user-friendly interface of the gateway.

Z-Wave SmartStart removes the need for initiating the end device to start inclusion. Inclusion is initiated automatically on power-ON and repeated at dynamic intervals for as long as the device is not included into a Z-Wave network. As the new device announces itself on power-ON, the protocol will provide notifications, and the gateway can initiate the inclusion process in the background, without the need for user interaction or any interruption of normal operation. This improvement also removes the possibility of other devices being included, as the SmartStart inclusion process only includes authenticated devices.

By moving the device authentication process into the manufacturing and distribution phase or service provider domain, the end user is no longer required to do anything but to power on the devices. This enables a simplified user experience where the device is genuinely ready to use, right out of the box. The device manufacturer or service provider can now prepare inclusion prior to the devices ending up at the end user's house.

Building on the elements introduced by S2 security, the Z-Wave SmartStart is not only easy for the end user, but also secure. Z-Wave SmartStart uses the same device specific keys (DSK) that form the foundation of the secure inclusion process of S2. Only authorized and intended devices are included in the Z-Wave network. Z-Wave SmartStart is based on the embedded SDK 6.8x and related gateway software components.

Smart start introduces several new API's for using learn mode and adding nodes to the network. For end nodes the ZW_NetworkLearnModeStart() is now used for controlling learn mode. And for controller nodes ZW_AddNodeToNetwork() is used. For details about the API see [3]

4.4 Smart Start and S2 QR Code Generation (SDK 6.80.00+)

Z-Wave devices supporting the Security 2 (S2) Command Class or Smart Start provisioning must provide a QR code physically on the device as well as on packaging. The actual marking and layout requirements are documented in [16] while the data string encoded in the QR code is specified in [24].

To facilitate on-the-fly generation of QR codes and S2 DSK to match the S2 Key pairs programmed into S2 and Smart Start devices, [25] defines a Production Control File that a device developer may hand over to the production facility doing the actual firmware programming, S2 key programming and QR code printing. The Production Control File defines the labels to be printed and the data to go into the QR code.

The current SDK release contains two software utilities to assist developers in creating the control file and to verify the contents of a QR code:

- QrCodeEncoder.xlsm
    - Encoding of QR code fields
    - Single-sample generation of QR codes for prototyping
    - Generation of dynamic string for Production Control File with fields to be replaced during production
- QrCodeDecoder.xlsm
    - Decodes the string contained in a QR code using an arbitrary smart phone QR code scanner application

All gray fields should be left untouched.

The encoder automatically generates a production control file named "ZwSmartStart.csv file" in the same folder as the QrCodeEncoder file.

The utilities are implemented using Excel sheets, incorporating several macro functions for SHA-1 checksum calculation, QR code rendering and control file generation. The utilities must therefore be stored in a folder that is not write protected.

4.5 FLiRS Multicast (SDK 6.80.00+)

Controllers can now send multicast to FLiRS nodes, so they react simultaneously to a command if they are in range of the controller.

The FLiRS multicast is based on a special destination address, so nodes that are not a part of a FLiRS multicast group will not wake up when a FLiRS multicast frame is send. The functionality is based on a learning process in the FLiRS node that works in the following way.

1’ transmission to a FLiRS multicast group:

1 Controller sends FLiRS multicast frame
2 All FLiRS nodes ignores the frame
3 Controller sends follow-up singlecast frame to all the FLiRS nodes in the group.
4 FLiRS nodes receives the follow-up singlecast and enables the FLiRS multicast address and passes the frame to the application

2’ transmission to a FLiRS multicast group

1 Controller sends FLiRS multicast frame
2 All FLiRS nodes in the multicast group will wakes up, receives the frame and passes it to the application
3 Controller sends follow-up singlecast frame to all the FLiRS nodes in the group.
4 FLiRS nodes receives the follow-up singlecast and passes the frame to the application

4.6 Improved Z-Wave Plus Framework (SDK 6.80.00+)

The Z-Wave Plus Application Framework is also improved in several places resulting in a simpler application development [19].

4.7 Improved Power Management (SDK 6.80.00+)

Power management is changed to get better synchronization between application and protocol. The time battery powered nodes are running at full power has been reduced considerably. For details about the new power management functionality see the Z-Wave framework guide [19] and the function ZW_Power_Management_Init() in [3]

4.8 Command Class Version version 3 (SDK 6.80.00+)

Version 3 of the Version Command class is now supported by the application.

4.9 MY Frequency Obsoleted (SDK 6.80.00+)

All the MY frequency targets are obsoleted. Use ANZ frequency instead of MY frequency in Malaysia.

4.10 New Variants of MyProductPlus Application (SDK 6.80.00+)

Two sensor type variants added to MyProductPlus application enabling support of battery-operated devices. The new battery-operated devices are as follows:

- Battery device supporting Command Class Battery
- Battery device supporting Command Class Battery and Command Class Wake Up.

4.11 Serial API Communication Interface Version (SDK 6.80.00+)

In SDK 6.80.0x changed the Serial API communication interface to version 8 enabling host software to check that this version of the serial API supports Smart Start.

4.12 Intermediate Applications Removed (SDK 6.80.00+)

The intermediate applications are now available in SDK 6.61.01+.

4.13 Support for NVM Ultra-Deep Sleep (SDK 6.80.00+)

Z-Wave modules using Adesto AT45DBxxxE chips as external NVM can now use the ultra-deep sleep mode in sleep mode. Regarding a full list of recommended EEPROM/FLASH components as external NVM, refer to [20].

5 Z-WAVE PROTOCOL

The Z-Wave Protocol (Z-Wave API library) is a low bandwidth half-duplex protocol designed for reliable and robust wireless communication in a low-cost control mesh network. This version supports the 500 Series single chips in various configurations. For an overview refer to [1] and for a detailed description of the API calls refer to [3]. The API consists of five different libraries; a Portable Controller library, a Static Controller library, a Bridge Controller library, a Routing Slave library, and an Enhanced 232 Slave library. The type of library used depends on the application features needed.

5.1 New Features

Refer to Chapter 3 and 4.

6 Z-WAVE FRAMEWORK AND EMBEDDED APPLICATIONS

The SDK contains code as well as compiled code for Z-Wave Plus embedded applications according to devices in [15]-[16], and command classes in [6]-[9]. The Z-Wave Plus embedded applications supports both non-secure and secure S0/S2 in one target.

Associations must be configured to examine all the features in the Z-Wave Plus embedded applications. Setting up associations is supported fully by the Z-Wave PC based Controller v5 and not older versions of the Z-Wave PC based Controller.

The code can be used as is to become familiar with Z-Wave or changed according to the needs of the application programmer.

A Z-Wave application based on earlier Z-Wave Single Chips requires porting of the source code to the 500 Series Single Chip. For details about porting, refer to [13], [14] and [3].

A Z-Wave Plus application based on earlier SDKs may also require porting to a newer version of the Z-Wave Plus Application Framework. For details, refer to [19].

6.1 Z-Wave Plus Application Certification Guide Lines

Introduction of the S2 security solution mandates additional requirements to the certification program. A large part of the S0/S2 security solution resides in the Z-Wave Protocol but parts are also located in the Z-Wave Plus Application Framework and Z-Wave Plus Applications. The application developer must therefore be aware of below list of 24 command class requirement numbers [8] applicable for slave-based devices during development:

Command Class Requirement Numbers (slaves)		
CC:009F.01.0E.11.008	CC:009F.01.05.11.018	CC:009F.01.00.11.09A
CC:009F.01.0E.11.003	CC:009F.01.05.11.017	CC:009F.01.00.11.04C
CC:009F.01.0D.11.007	CC:009F.01.00.11.070	CC:009F.01.00.11.034
CC:009F.01.0D.11.004	CC:009F.01.00.11.06E	CC:009F.01.00.21.00B
CC:009F.01.0D.11.003	CC:009F.01.00.11.061	CC:009F.01.00.21.009
CC:009F.01.0A.11.002	CC:009F.01.00.11.05E	CC:009F.01.00.21.008
CC:009F.01.08.11.007	CC:009F.01.00.11.092	CC:009F.01.00.21.007
CC:009F.01.06.11.003	CC:009F.01.00.11.051	CC:009F.01.00.11.050

Not all listed requirements may be relevant depending on the application in question.

6.2 Door Lock with Key Pad

The Door Lock with Key Pad application shows a lock implementation that has a built-in keypad. It will support user codes to open a door and thereby eliminate the need for traditional keys. Typically, it is possible to both lock and unlock the door remotely through the Z-Wave protocol. The Door Lock with Key Pad implementation is built upon the Z-Wave Plus Application Framework [19]. The Door Lock with Key Pad is based on Door Lock Keypad Device Type with Listening Sleeping Slave (LSS) as the Role Type and enhanced 232 slave. For detailed information, refer to [11].

6.3 My Product Plus

As an alternative to the Device Type code applications use My Product Plus, this application contains the minimum framework for developing a Z-Wave Plus application. The My Product Plus implementation is built upon the Z-Wave Plus Application Framework [19]. For detailed information, refer to [11].

6.4 On/Off Switch

The On/Off Power Switch application shows a switch implementation to turn on any device that is connected to power. Examples include lights, appliances etc. The On/Off Switch implementation built upon the Z-Wave Plus Framework [19]. The On/off Switch is based on On/Off Power Switch Device Type with Always On Slave (AOS) as the Role Type and enhanced 232 slave. For detailed information, refer to [11].

6.5 PIR Sensor

The PIR Sensor application shows a presence/movement detector implementation for controlling other devices and for sending notifications. The PIR Sensor implementation built upon the Z-Wave Plus Framework [19]. The PIR Sensor is based on Sensor – Notification Device Type with Reporting Sleeping Slave (RSS) as the Role Type and routing slave. For detailed information, refer to [11].

6.6 Power Strip

The Power Strip application shows an extension block implementation to turn on a number of devices that is connected to power. Examples include lights, appliances etc. The Power Strip implementation built upon the Z-Wave Plus Framework [19]. The Power Strip is based on Power Strip Device Type with Always On Slave (AOS) as the Role Type and enhanced 232 slave. The Power Strip show also how to implement a Multi-Channel device. For detailed information, refer to [11].

6.7 Production Test DUT

The Production Test DUT code for a device under test contains an example of how the basic tasks of testing devices in a Z-Wave network can be implemented using the Z-Wave API. Detailed information regarding the Production Test DUT code can be found in [11].

6.8 Production Test Generator

The Production Test Generator code contains an example of 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 in conjunction with the Production Test DUT to verify the TX / RX circuits on Z-Wave enabled products. Detailed information regarding the Production Test Generator code can be found in [11].

6.9 Serial API Plus

The Serial Applications Programming Interface (Serial API) allows a host to communicate with a Z-Wave chip. The host may be a PC or a less powerful embedded host CPU, e.g. in a remote control or in a gateway device. This solution is typically used when the whole application cannot reside on the Z-Wave chip itself. The Serial API code contains an example of how a serial UART interface to the Z-Wave protocol can be implemented. The Serial API supports both controller and slave applications. For detailed information, refer to [11]. In addition, detailed information about the Serial API code and how to interface to the Serial API can be found in [18] and [3].

6.10 Wall Controller

The Wall Controller application shows a push button switch panel implementation to control devices in the Z-Wave network from push buttons (physical or virtual) on a device that is meant to be mounted on a wall. Examples include scene and zone controller, wall mounted AV controllers. Device of this type can both be battery operated or mains powered. The Wall Controller implementation built upon the Z-Wave Plus Framework [19]. The Power Strip is based on Wall Controller Device Type with Always On Slave (AOS) as the Role Type and enhanced 232 slave. The Wall Controller shows how to implement an interrupt service routine (ISR) on application level. For detailed information, refer to [11].

6.11 Z-Wave Plus Application Framework

The Z-Wave Plus Application Framework simplifies implementation of robust Z-Wave Plus compliant and interoperable products. Many Z-Wave certification requirements are also handled by the Z-Wave Plus Application Framework making it a lot easier to pass certification. The Z-Wave logo is only granted to products passing certification.

6.11.1 Application Command Handlers

The Application Command Handlers code contains an implementation of various command classes used by Z-Wave Plus applications. For detailed information, refer to [19].

6.11.2 Application Utilities

The Application Utilities code contains an implementation of various general-purpose functions used by Z-Wave Plus applications. A large part of the S0/S2 security solution resides now in the Z-Wave Protocol. For detailed information, refer to [19].

7 TOOLS

The SDK contains various tools for helping SW developers writing and debugging code.

NOTICE: Some of the tools are not bundled together with the SDK anymore but are available on ZTS as individual programs:

7.1 IMA Tool Box

The IMA Tool Box supports an installation and maintenance procedure, which can ensure an easy installation and provide an operational qualification of the installation. Use this tool in combination with the Serial API based hex targets, that default incorporates IMA functionality. The source code of IMA Tool Box is also included. For detailed information, refer to [2].

7.2 uVision4 Project Generator

The Keil uVision4 Project Generator makes uVision projects for a sample application. The makefile system can generate uVision projects for the Keil uVision4 IDE by calling the project generator.

REFERENCES

[1] Silicon Labs, INS10243, Instruction, Z-Wave Protocol Overview.

[2] Silicon Labs, INS12712, Instruction, Z-Wave Network Installation and Maintenance Procedure User Guide.

[3] Silicon Labs, INS13954, Instruction, Z-Wave 500 Series Application Programming Guide v6.80.0x.

[4] Silicon Labs, SDS10242, Software Design Specification, Z-Wave Device Class Specification.

[5] Silicon Labs, SDS10865, Software Design Specification, Z-Wave Security Application Layer.

[6] Silicon Labs, SDS13781, Software Design Specification, Z-Wave Application Command Class Specification.

[7] Silicon Labs, SDS13782, Software Design Specification, Z-Wave Management Command Class Specification.

[8] Silicon Labs, SDS13783, Software Design Specification, Z-Wave Transport-Encapsulation Command Class Specification.

[9] Silicon Labs, SDS13784, Software Design Specification, Z-Wave Network-Protocol Command Class Specification.

[10] Silicon Labs, SDS13548, Software Design Specification, List of defined Z-Wave Command Classes.

[11] Silicon Labs, INS13933, Instruction, Z-Wave 500 Series SDK Contents v6.80.0x.

[12] Silicon Labs, INS12366, Instruction, Working in 500 Series Environment User Guide.

[13] Silicon Labs, APL12444, Application Note, Porting Z-Wave Appl. SW from ZW0301 to 500 Series.

[14] Silicon Labs, APL12445, Application Note, Porting Z-Wave Appl. SW from 400 to 500 Series.

[15] Silicon Labs, SDS11846, Software Design Specification, Z-Wave Plus Role Types Specification.

[16] Silicon Labs, SDS11847, Software Design Specification, Z-Wave Plus Device Types Specification.

[17] Silicon Labs, SDS12467, Software Design Specification, 500 Series Z-Wave Chip NVR Flash Page Contents.

[18] Silicon Labs, INS12350, Instruction, Serial API Host Appl. Prg. Guide.

[19] Silicon Labs, INS13953, Instruction, Z-Wave Plus Application Framework v6.80.0x.

[20] Silicon Labs, INS12213, Instruction, 500 Series Integration Guide.

[21] Silicon Labs, INS13474, Instruction, Z-Wave Security Whitepaper.

[22] Silicon Labs, SDS13349, Software Design Specification, Security considerations in Home Control installations.

[23] Silicon Labs, APL13434, Application Note, FAQ: On the use of S0, S2 and Supervision CC in product design and deployments.

[24] Silicon Labs, SDS13937, Software Design Specification, Node Provisioning QR Code Format.

[25] Silicon Labs, INS13975, Instruction, Smart Start Production control.

INDEX