Application Note

Porting Z-Wave Appl. SW from 500 to 700

Document No.:	APL14440
Version:	2
Description:	The purpose of this document is to give guidelines for the Z Wave application developer, when porting software applications based on Z-Wave Framework from 500 to 700.
Written By:	JFR;COLSEN;NOBRIOT;AYURTTAS
Date:	2019-02-20
Reviewed By:	EFH;JFR;PSH;SSE;BBR;COLSEN;AMUNKHAUS;JRM;CRASMUSSEN;SCBROWNI
Restrictions:	Public

Approved by:

Date	CET	Initials	Name	Justification
2019-02-20	07:19:02	NTJ	Niels Johansen	

REVISION RECORD

Doc. Rev	Date	By	Pages affected	Brief description of changes
1	20181129	JFR	ALL	Initial draft; based on APL12444
2	20181210	CHOLSEN	3	Added section about porting from 500 based SDK to 700 based SDK.
2	20190116	MLEDESMA	ALL	Grammar and structure (consistent format) modification

Table of Contents

1 ABBREVIATIONS

Abbreviation	Explanation
API	Application Programming Interface
OTA	Over The Air (firmware update)
SDK	System Development Kit

2 INTRODUCTION

2.1 Purpose

The purpose of this document is to give guidelines to Z-Wave application developers for porting applications based on the Z-Wave Framework from 500-based SDKs to 700-based SDKs.

2.2 Audience and prerequisites

The audience of this document is Z-Wave partners and Silicon Labs.

3 PORTING AREAS

The following sections describe where to focus the effort when porting from a 500 application based on the Z-Wave Framework to a 700 application based on the Z-Wave Framework. In general, all differences described are applied to the certified apps in the 700 based SDK.

Because of the change in chip architecture, it is strongly recommended to take one of the applications from the 700 based SDK and add the business logic of a 500 application.

3.1 Non-volatile memory

The 500 applications had two files named eeprom.c and eeprom.h. Non-volatile memory was configured in these files by listing all the variables that should be contained in the EEPROM. This has changed in the 700 applications where a part of the flash inside the Z-Wave chip is dedicated to a file system. The file system is developed by Silabs and is called NVM3. Instead of a long list of variables in the 500 applications, the application must define files. Taking a look at NVM3SetCaretakerVerify(), invoked by the applications, and the variables used by this function is a good place to start.

3.2 RTOS

The Z-Wave 700-based SDK utilizes FreeRTOS and is split into two tasks: one for the Z-Wave protocol and one for the Z-Wave Application Framework and the app. Given this architecture, direct function calls between the application and the protocol are no longer allowed when the application task has been started (ApplicationTask()). Instead, communication between the two tasks are based on four queues: one for transmitting frames, one for receiving frames, one for configuration commands, and one for receiving statuses for either frames or configuration commands.

Values that could previously be fetched by calling a protocol function are now stored in a variable that is passed to the application as an argument to the FreeRTOS task.

3.3 Peripherals

In the 500-based SDK, drivers were part of the Z-Wave protocol, but in the 700-based SDK, these drivers are now generic gecko drivers.

3.4 Board files

Several board files have been introduced to abstract from hardware and thereby ease the transition from Silabs development boards to custom made hardware. The board files either begin with “board_” or “extension_board_”. It is not required to use all these files.

3.5 Using existing command classes

Some of the command classes delivered with the 700-based SDK no longer depend on

Definitions? in config_app.h but instead depend on external functions that must be implemented by the application.

Several of the command class APIs have been renamed to a more consistent naming scheme and the rest will follow.

3.6 Implementing new command classes

The Z-Wave 500 chip did not have a stack and the mutex used in command classes was introduced to minimize the use of RAM for sending frames. The Z-Wave 700 chip does have a stack, which means that variables can be allocated on the stack and deleted when they are no longer needed.

Upon receiving a get command, a response must be sent, and the RAM needed for the report can now be allocated on the stack. This simplifies the command class handler because the old 500 function GetResponseBuffer() is no longer required. Instead, a variable of the type ZW_APPLICATION_TX_BUFFER is defined, and the frame content is written to this variable. It is then passed to the protocol by invoking Transport_SendResponseEP().

3.7 Z-Wave Plus Version 2

Z-Wave Plus has been upgraded to version 2 and this introduces several requirements to the application. One of the features is True Status, which is described in more detail in [1].

Another new requirement is the Identify feature. Every application must be able to identify itself using the Indicator command class. The use of Indicator CC is described in [1].

REFERENCES

[1] Silicon Labs, INS14259, Instruction, Z-Wave Plus V2 Application Framework SDK7

INDEX

No index entries found.