

User Manual

DA14585 IoT Multi Sensor Development Kit Software Reference Applications

UM-B-096

Abstract

The IoT Multi Sensor Development Kit (MSK) based on DA14585 supports 15 Degrees of Freedom and includes five references designs: IoT Sensors, Smart Tag and three different types of beacons. The corresponding apps run on iOS/Android devices and the cloud services offers great flexibility to customers in product design. This document describes the architecture and the implementation details of DA14585 IoT MSK reference design and the supporting applications.

Contents

Abstract	1
Contents	2
Figures	7
Tables	9
1 Terms and Definitions	12
2 References	12
3 Introduction	14
3.1 Internet of Things	14
3.2 DA14585 IoT MSK Hardware Features	14
3.3 DA14585 IoT MSK Hardware Architecture	16
4 Quick Start Guide	16
4.1 Setting up the Hardware	16
4.2 Downloading and Programming	17
4.2.1 Prerequisites	17
4.2.2 Opening, Building and Downloading the Projects	17
4.2.3 Burning Images in Flash	19
4.3 Software over the Air Updates	19
4.4 Magnetometer Auto-Calibration Procedure	20
4.4.1.1 Prerequisites for Magnetometer Calibration	21
4.4.1.2 Procedure of Magnetometer Calibration	21
4.4.1.3 Magnetometer Sensor and Calibration State Reporting	21
5 IoT Sensors Reference Application	22
5.1 Software Features	22
5.2 Software Architecture	22
5.2.1 Project Files	22
5.2.2 Source Files	23
5.2.3 Application Configuration	25
5.3 Operation Overview	27
5.3.1 General Description	27
5.3.2 Application initialization	27
5.3.3 Advertise	28
5.3.4 Connected/Sensors Idle	28
5.3.5 Connected/Sensors Active	29
5.3.6 Connected/Sensors Stopped	30
5.3.7 Disconnect	30
5.4 Wkup_adapter	31
5.5 Sensor Interface	31
5.5.1 General Description	31
5.5.2 Driver Adaptation Layer	32
5.5.3 Sensor Interface API	32

5.6	Device Drivers	34
5.6.1	Environmental Sensor	34
5.6.2	Motion Sensor.....	34
5.6.3	Magneto Sensor	35
5.6.4	Optical Sensor	35
5.6.5	GPIO Extender	35
5.6.6	Power Amplifier.....	36
5.7	Sequence Diagrams.....	36
5.7.1	Sensor fusion data reporting	36
5.7.2	Environmental Data Reporting	37
5.8	Dialog Wearable Service V2	38
5.8.1	Features Report Structure	39
5.8.2	Multi Sensor Report and Sensor Report.....	40
5.8.2.1	Sensor Report for Accelerometer, Gyroscope, and Magnetometer.....	41
5.8.2.2	Sensor Report for Temperature, Humidity, Gas, and Barometric Pressure	41
5.8.2.3	Sensor Report for Indoor Air Quality (IAQ).....	42
5.8.2.4	Sensor Report for Ambient Light and Proximity	42
5.8.2.5	Sensor Report for Button.....	42
5.8.2.6	Sensor Report for Sensor Fusion	42
5.8.3	Report Structures for Configuration and Control	43
5.8.3.1	Start Command.....	44
5.8.3.2	Stop Command.....	44
5.8.3.3	Read Parameters from Flash Memory	44
5.8.3.4	Reset to Factory Defaults	44
5.8.3.5	Store Basic Configuration in Flash Memory	45
5.8.3.6	Store Calibration Coefficients and Control Configuration in Flash Memory.....	45
5.8.3.7	Return Running Status	45
5.8.3.8	Reset Sensor Fusion and Calibration Configuration	45
5.8.3.9	Basic configuration	46
5.8.3.10	Read Basic Configuration	48
5.8.3.11	Set Sensor Fusion Coefficients Command	49
5.8.3.12	Read Sensor Fusion Coefficients	50
5.8.3.13	Set Calibration Coefficients	50
5.8.3.14	Read Calibration Coefficients	50
5.8.3.15	Set Calibration Control Flags.....	51
5.8.3.16	Read Calibration Control	52
5.8.3.17	Fast Accelerometer Calibration	52
5.8.3.18	Set Calibration Modes	52
5.8.3.19	Read Calibration Modes	53
5.8.3.20	Read Device Sensors	53
5.8.3.21	Read Software Version.....	54

5.8.3.22	Start LED Blink	54
5.8.3.23	Stop LED Blink.....	54
5.8.3.24	Set Proximity Hysteresis Limits	55
5.8.3.25	Read Proximity Hysteresis Limits	56
5.8.3.26	Calibration Complete	56
5.9	Sensor Calibration Library.....	56
5.9.1	Overview	56
5.9.1.1	Modes of Operation	56
5.9.1.2	Calibration Routines	57
5.9.1.3	Calibration Procedure	58
5.9.2	API Usage.....	58
5.9.2.1	Allocation	59
5.9.2.2	Initialization	59
5.9.2.3	Processing	61
5.10	Sensor Fusion Library.....	62
5.10.1	Overview	62
5.10.1.1	Modes of operation	62
5.10.2	API	63
5.10.2.1	Memory allocation.....	63
5.10.2.2	Initialization	63
5.10.2.3	Processing	63
5.11	IoT MSK Android and iOS Application	64
5.11.1	Installing from Apple AppStore	64
5.11.2	Installing from the PlayStore.....	65
5.11.3	Scan Screen	65
5.11.4	Side Menu.....	66
5.11.4.1	Sensor Screen: Environmental Sensors.....	67
5.11.4.2	Sensor screen: IMU sensors	68
5.11.4.3	Sensor Fusion 3D screen	68
5.11.5	Basic Settings Screen.....	70
5.11.6	Magnetometer Calibration Settings Screen	71
5.11.6.1	Magnetometer Calibration File	72
5.11.7	SmartFusion Coefficients Screen	72
5.11.8	Fast Accelerometer Calibration Screen.....	73
6	Smart Tag Reference Application.....	74
6.1	Introduction	74
6.2	Software Features	74
6.2.1	Profiles and Services	74
6.2.2	Alerts.....	75
6.2.3	Advertising and Sleep Phases.....	75
6.2.4	Push-Button Interface	75
6.2.5	Security	76
6.2.6	Battery Level.....	76

6.3	Software Architecture	76
6.4	Operation Overview and State Machines	76
6.4.1	Application Configuration Parameters	77
6.4.2	Application Task State Machine	77
6.4.3	Callback Functions	79
6.4.4	Advertising	79
6.4.5	Connection	80
6.4.6	Security	80
6.4.7	Push button	81
6.4.8	Proximity Reporter and Alerts	81
6.4.9	PWM Engine	82
6.4.10	SmartTag Sequence Diagram	84
6.5	Application of SmartTags in Android and iOS	84
6.5.1	Overview	84
6.5.2	Installation	85
6.5.3	Device List	86
6.5.4	Device Details	86
7	Beacon Reference Applications	91
7.1	Introduction	91
7.2	What is a Beacon?	92
7.3	Beacon Example	92
7.4	Beacon Formats	93
7.4.1	iBeacon	93
7.4.2	AltBeacon	93
7.4.3	Eddystone	94
7.4.3.1	Eddystone-UID	96
7.4.3.2	Eddystone-URL	97
7.4.3.3	Unencrypted Eddystone-TLM	98
7.5	Beacon Parameters	99
7.5.1	Advertising Data	99
7.5.1.1	Using the user_default_beacon_config Struct	100
7.5.1.2	Reading Advertising Data from Flash	100
7.5.2	Advertising Interval	101
7.6	Software Features	102
7.7	Software Architecture	102
7.8	Operation Overview	104
7.8.1	Configuration Switches	104
7.9	User Advertise SW Module	105
7.9.1	Style	105
7.9.2	Pattern	106
7.9.3	User Advertise SW Module Callbacks	108
7.10	Device Configuration Service	108
7.10.1	Device Configuration Service Specification	108

7.11	Environmental Data Notifications Service.....	109
7.11.1	Environmental Data Notifications Service Specification.....	109
7.12	Beacon Configuration.....	110
7.12.1	Beacon Configuration Memory Map.....	110
7.13	Battery Level Sampling.....	112
7.14	Beacon Examples for DA14585 IoT MSK.....	112
7.14.1	AltBeacon.....	112
7.14.1.1	AltBeacon Example Sequence Diagram.....	114
7.14.2	Eddystone.....	114
7.14.2.1	Eddystone Example Sequence Diagram.....	115
7.14.3	iBeacon.....	116
7.14.3.1	iBeacon Example Sequence Diagram.....	117
7.15	Finding Advertising Devices.....	117
8	Memory Footprint and Power Measurements.....	118
8.1	Memory Footprint.....	118
8.2	Power consumption.....	119
9	IoT Cloud Applications.....	120
9.1	Introduction.....	120
9.2	User and Device Management.....	120
9.2.1	Users.....	120
9.2.2	Devices.....	122
9.3	Historical Data.....	122
9.3.1	Cloud Application.....	122
9.3.2	Mobile Application.....	123
9.4	Alerting.....	124
9.4.1	Cloud Application.....	124
9.4.2	Mobile Application.....	124
9.5	Control.....	125
9.5.1	Cloud Application.....	125
9.5.2	Mobile application.....	126
9.6	Amazon Alexa.....	127
9.6.1	Skill Activation.....	127
9.6.2	How to Use.....	128
9.7	IFTTT.....	131
9.7.1	Create an IFTTT Applet.....	131
9.7.2	IFTTT Web Hook Key.....	136
9.7.3	Web App - IFTTT Settings.....	137
9.7.4	Mobile App: IFTTT Settings.....	138
	Appendix A Multi Sensor Kit Boot Sequence.....	140
	Appendix B Memory Map.....	141
	Appendix C Using the mkimage Application.....	143
C.1	mkimage Scripts.....	143

C.2	mkimage modes	144
C.2.1	mkimage Single	144
C.2.2	mkimage Multi	144
C.2.3	mkimage Whole_img	145
C.2.4	mkimage Multi_no_suota	145
C.2.5	mkimage cfg	145
Appendix D Flash Programming in MSK Applications.....		146
D.1	Basic Information About the MSK Applications	146
D.1.1	Product Header	146
D.1.2	Image Header	146
D.1.3	Beacon Configuration Struct and Configuration Struct Header	146
D.1.4	Smart Tag Bonding Data, IoT Flash Base, IoT Flash Base Cal	147
D.2	Flash Programming	147
D.2.1	Burning the Whole Image in Flash Memory	147
D.2.2	Preparing the Various .img and .bin Files Manually	150
Appendix E Using the SUOTA Application for Android		154
Appendix F Miscellaneous		160
F.1	Calculating the RSSI Value	160
Appendix G Troubleshooting		161
G.1	Magnetometer or Sensor Fusion Have Inconsistent Behaviors	161
G.2	Device Does Not Connect after Switching to a Different Firmware	161
G.3	Proximity Sensor is always On	161
Revision History		162

Figures

Figure 1: Internet of Things (IoT).....	14
Figure 2: Hardware Architecture of DA14585 IoT MSK	16
Figure 3: Setting up the Hardware	17
Figure 4: MSK Target Apps Folder	18
Figure 5: Smart Tag Folder	18
Figure 6: Smart Tag Output Folder	18
Figure 7: SUOTA Feature.....	19
Figure 8: DA14585 IoT MSK Software Architecture	23
Figure 9: General Application Flow of DA14585 IoT MSK	27
Figure 10: DA14585 IoT MSK in Advertise state	28
Figure 11: Activating sensors	29
Figure 12: Overview of Sensors Data Path	30
Figure 13: Sensors Interface Overview	32
Figure 14: SI Registration Path	33
Figure 15: TIMED Sensors Timeline	33
Figure 16: Sequence Diagram of Sensor Fusion Reporting	37
Figure 17: Sequence Diagram of Environmental Sensor Reporting	38
Figure 18: Installing Dialog IoT Sensors App from Apple AppStore	64
Figure 19: Installing Dialog IoT Sensors App from Google PlayStore	65
Figure 20: Scan screen of Dialog IoT Sensors App	65

Figure 21: Side Menu	66
Figure 22: Environmental Sensor Screen of Dialog IoT Sensors App	67
Figure 23: IMU Sensor Screen of Dialog IoT Sensors App	68
Figure 24: 3D Screen of Dialog IoT Sensors App	69
Figure 25: Basic settings of Dialog IoT Sensors App	70
Figure 26: Calibration Settings of Dialog IoT Sensors App	71
Figure 27: Sensor Fusion Settings of Dialog IoT Sensors App	73
Figure 28: Accelerometer Calibration Settings of Dialog IoT Sensors App	73
Figure 29: Smart Tag Application Task FSM	78
Figure 30: Smart Tag Reference Application Sequence Diagram	84
Figure 31: SmartTags APP Icon	85
Figure 32: Installation of Dialog SmartTags App	86
Figure 33: Device List	86
Figure 34: Device Details	87
Figure 35: Distance Alert Window	88
Figure 36: How to Access the Side Menu	89
Figure 37: Device Information	89
Figure 38: Disclaimer	90
Figure 39: Seek & Find screens	91
Figure 40: Beacon Protocol Logos	91
Figure 41: Bluetooth Low Energy Beacon	92
Figure 42: Description of the Exhibit on a Smartphone	93
Figure 43: iBeacon Frame	93
Figure 44: AltBeacon Frame	94
Figure 45: Eddystone Modes Supported by the Dialog Beacon Reference Design	95
Figure 46: Eddystone Different Mode Frames Analyzed	95
Figure 47: Example of a Device Configuration Struct in Flash Memory	101
Figure 48: Beacon SW System Overview	103
Figure 49: Operation Overview	104
Figure 50: User Advertise usage example	107
Figure 51: Data Format in Write Configuration	108
Figure 52: Indication Data Format in Read Response	109
Figure 53: AltBeacon Example Transition Diagram	113
Figure 54: Altbeacon Sequence Diagram	114
Figure 55: Eddystone UID/URL - TLM Example Transition Diagram	115
Figure 56: Eddystone Sequence Diagram	115
Figure 57: iBeacon Example Transition Diagram	116
Figure 58: iBeacon Sequence Diagram	117
Figure 59: Locate App Screenshot	118
Figure 60: User Management	121
Figure 61: Create Account or Pairing with an Existing Account	121
Figure 62: Device Management	122
Figure 63: Historical Data: Cloud application	123
Figure 64: Historical Data: Mobile Application	123
Figure 65: Alerting: Cloud Application	124
Figure 66: Alerting: Mobile application	125
Figure 67: Control: Cloud Application	126
Figure 68: Control: Mobile Application	127
Figure 69: Dialog Evaluation Kit Skill for Alexa	128
Figure 70: Alexa: Skill Activation Succeeded Screen	128
Figure 71: Alexa: Sign in Procedure	129
Figure 72: Alexa Mobile Application Main Screen	129
Figure 73: IFTTT Signup Page	131
Figure 74: New Applet	131

Figure 75: Make an IFTTT Applet	132
Figure 76: Make an IFTTT Applet Step 1	132
Figure 77: Make an IFTTT Applet Step 2	133
Figure 78: Make an IFTTT Applet, Complete Step 2	133
Figure 79: Make an IFTTT Applet, Create That	134
Figure 80: Make an IFTTT Applet, Step 3	134
Figure 81: Make an IFTTT Applet, Step 4	135
Figure 82: Make an IFTTT Applet Step 5	135
Figure 83: Make an IFTTT applet, Finish	136
Figure 84: IFTTT Web Hook Key, step 1	136
Figure 85: IFTTT Web Hook Key Step 2	137
Figure 86: IFTTT Web Hook Key Step 3	137
Figure 87: Web App - IFTTT Settings	138
Figure 88: IFTTT Settings - Mobile app	139
Figure 89: Email When "Temperature" Occurred	139
Figure 90: Application Programmed in OTP Flags	140
Figure 91: Analyzing a Flash Memory Image	141
Figure 92: Initial Window to Choose Device and Connection Type	148
Figure 93: Opening SmartSnippets Board Setup	148
Figure 94: Smart Snippets Board Setup Window	149
Figure 95: SPI Flash Programmer	149
Figure 96: Initial Window to Choose Device and Connection Type	151
Figure 97: Device Config Struct Format in .txt File	151
Figure 98: Programming the Various Fields of the Device Configuration Struct	152
Figure 99: Creating a Custom Dev_Conf_Struct .bin File	152
Figure 100: SUOTA App Icon	154
Figure 101: Device Selection Menu	155
Figure 102: DIS Screen	156
Figure 103: File Selection Screen	157
Figure 104: SUOTA Parameter Settings	158
Figure 105: SUOTA Uploading Screen	159
Figure 106: Successful Update Screen	160

Tables

Table 1: SUOTA Profile Enabling	20
Table 2: Source Files for DA14585 IoT MSK: Overview	23
Table 3: Source Files Specific to DA14585 IoT MSK	24
Table 4: Header Files for the Configuration of DA14585 IoT MSK	25
Table 5: Configuration Parameters	26
Table 6: DWSv2 Characteristics	38
Table 7: Features Report Structure	39
Table 8: Multi Sensor Report	40
Table 9: Sensor Report	40
Table 10: Report Types/Report ID's	40
Table 11: Report Structure for Accelerometer, Gyroscope, and Magnetometer	41
Table 12: Bitfield Structure for snsr_state	41
Table 13: Environmental Sensor Report	41
Table 14: Indoor Air Quality (IAQ) Report	42
Table 15: Sensor Report for Ambient Light and Proximity	42
Table 16: Sensor Report for Button	42
Table 17: Sensor Report for Sensor Fusion	42
Table 18: Report Structure for Commands	43

Table 19: Commands	43
Table 20: Start Command	44
Table 21: Start Command Reply	44
Table 22: Stop Command.....	44
Table 23: Stop Command Reply	44
Table 24: Read Flash Command	44
Table 25: Reset to Defaults (RtD) Command.....	44
Table 26: Store Basic Configuration Command	45
Table 27: Store Calibration and Control Command	45
Table 28: Return Running Status Command	45
Table 29: Return Running Status Reply.....	45
Table 30: Reset Sensor Fusion and Calibration Configuration command	45
Table 31: Basic Configuration Command.....	46
Table 32: Read Basic Configuration Command	48
Table 33: Read Basic Configuration Command reply	48
Table 34: Set Sensor Fusion Coefficients Command	49
Table 35: Read Sensor Fusion Coefficients Command	50
Table 36: Read Sensor Fusion Coefficients Command Reply.....	50
Table 37: Set Calibration Coefficients Command	50
Table 38: Read Calibration Coefficients Command	50
Table 39: Read Calibration Coefficients Command Reply.....	50
Table 40: Set Calibration Control Flags Command.....	51
Table 41: Calibration Control Flags #1	51
Table 42: Calibration Control Flags #2.....	51
Table 43: Calibration Parameters.....	51
Table 44: Read Calibration Control Flags Command	52
Table 45: Read Calibration Control Flags Command Reply	52
Table 46: Fast Accelerometer Calibration Command	52
Table 47: Fast Accelerometer Calibration Reply.....	52
Table 48: Set Calibration Modes Command	52
Table 49: Read Calibration Modes Command	53
Table 50: Read Calibration Modes Command Reply	53
Table 51: Read Device Sensors Command	53
Table 52: Read Device Sensors Command reply	53
Table 53: Read Application Software Version Command	54
Table 54: Read Application Software Version Command Reply	54
Table 55: Start LED Blink Command	54
Table 56: Stop LED Blink Command.....	54
Table 57: Set Proximity Hysteresis Limits Command	55
Table 58: Read Proximity Hysteresis Limits Command	56
Table 59: Read Proximity Hysteresis Limits Command Reply.....	56
Table 60: Calibration Complete Notification	56
Table 61: Alert Types	75
Table 62: Push-Button Interface.....	75
Table 63: Smart Tag Application Configurable Parameters.....	77
Table 64: Application Task: FSM States	77
Table 65: State Transitions of the Application Task FSM	78
Table 66: Example of Advertising Data from a Museum Beacon.....	92
Table 67: AltBeacon Protocol Fields	94
Table 68: Eddystone Frame Types	96
Table 69: Eddystone UID Frame.....	96
Table 70: Frame Specification.....	97
Table 71 URL Scheme Prefix	97
Table 72: Eddystone-URL HTTP URL Encoding	97

Table 73: Eddystone TLM Frame Specification	98
Table 74: Format of Struct user_beacon_config_tag	99
Table 75: Advertising Interval Location	101
Table 76: Source Files of Beacon Reference Applications	103
Table 77: List of Software Configuration Switches	105
Table 78: List of Profile Configuration Switches	105
Table 79: Characteristics of the Device Configuration Service	108
Table 80: Characteristics of the Environmental Data Notifications Service	109
Table 81: Configuration Data Format	110
Table 82: Memory Footprint	118
Table 83: Power Consumption	119
Table 84: Parts of the Image Depending on the Application	141
Table 85: Available mkimage scripts	143
Table 86: Product Header Format	146
Table 87: Image Header Format	146
Table 88: Beacon Configuration Header	147
Table 89: Beacon Configuration Struct Format	147
Table 90: Files Needed or Created During Flash Programming	150

1 Terms and Definitions

ADC	Analog to Digital Converter
BASS	Battery Service Server
BLE	Bluetooth Low Energy
DWS	Dialog Wearable Service
FSM	Finite State Machine
GAP	Generic Access Profile
GAPC	Generic Access Profile Controller
GAPM	Generic Access Profile Manager
GATT	Generic Attribute Profile
MTU	Maximum Transmission Unit
MSK	Multi Sensor Development Kit
UART	Universal Asynchronous Receiver/Transmitter
IoT	Internet of Things
NVDS	Non-Volatile Data Storage
PC	Personal Computer
RAM	Random Access Memory
Report	Notification of Sensor Data and Control
RSSI	Received Signal Strength Indicator
SF	Sensor Fusion
SFL	Sensor Fusion Library
SUOTA	Software Update over the Air
IAQ	Indoor Air Quality
MSK	Multi Sensor Kit
SI	Sensors Interface
MEMS	Microelectromechanical Systems
IMU	Inertial Measurement Unit
FIFO	First In First Out buffer

2 References

- [1] UM-B-080, DA14585/586 SDK 6 Software Developer's Guide, User Manual, Dialog Semiconductor.
- [2] UM-B-079, DA14585/586 SDK 6 Software Platform Reference, User Manual, Dialog Semiconductor.
- [3] RW-BLE-GAP-IS, GAP Interface Specification, Riviera Waves.
- [4] UM-B-048, DA14585/DA14586 Getting Started Guide with the Basic Development Kit V2.0, User Manual, Dialog Semiconductor.
- [5] DA14585, Datasheet v3.2, Dialog Semiconductor.
- [6] UM-B-089, DA14585 Range Extender Reference Application, User Manual, Dialog Semiconductor.
- [7] <https://www.bluetooth.com/specifications/assigned-numbers/company-identifiers>
- [8] UM-B-065, Bluetooth® Smart Communication Interface Board, User Manual, Dialog Semiconductor.

- [9] <https://github.com/google/eddystone>
- [10] <https://github.com/google/eddystone/tree/master/eddystone-uid>
- [11] <https://github.com/google/eddystone/tree/master/eddystone-url>
- [12] <https://github.com/google/eddystone/blob/master/eddystone-tlm/tlm-plain.md>
- [13] <https://github.com/AltBeacon/spec>
- [14] UM-B-095, DA14585 IoT Multi Sensor Development Kit Hardware Design, User Manual, Dialog Semiconductor.
- [15] <https://www.bluetooth.com/specifications/gatt>.
- [16] AN-B-001, DA14580/581/583 Booting from serial interfaces, Application Note, Dialog Semiconductor

3 Introduction

3.1 Internet of Things

With yearly shipments of more than 10 billion microcontrollers that all can exchange information locally or through the Internet, a huge variety of so called "intelligent devices" are enabled. These devices include motion sensors, pool pumps, gas/electric meters, street lamps, and many more.

All these devices can be accessed over the Internet; thanks to the rapid increase in infrastructure coverage and Internet access. This evolution is often called the Internet of Things (IoT). Other names include Internet of Everything (IoE), Web of Things, Embedded Web, and Industry 4.0. The goal is to establish an Internet connection for the small "things" you carry with you or use in a factory, hospital, city, or at home (Figure 1). IoT industrial experts predict that there are going to be more than 50 billion connected devices within the next 10 years.

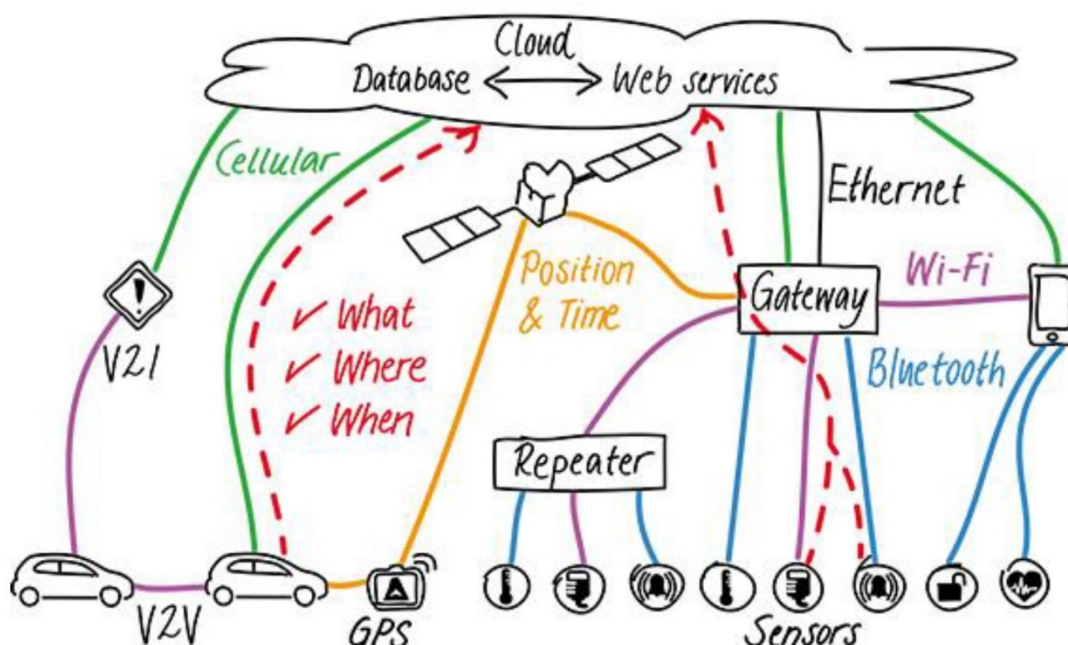


Figure 1: Internet of Things (IoT)

Dialog's IoT Multi Sensor development kit (MSK) is a compact development kit with 15 degrees of freedom accompanied by five software reference applications and incorporating versatile sensor management with cloud applications. The IoT MSK is based on DA14585 Bluetooth Low Energy SoC and a number of motion and environmental sensors. The five software reference applications can be used by Dialog's apps running in iOS or Android devices. Users can build a rich cloud application with the data from this IoT MSK in just a few configuration steps. In addition, the cloud applications provided by Dialog can be used to monitor the data from the IoT MSK sensors and program IFTT events. This IoT MSK offers users high flexibility in product design to cope with the fast changing IoT market.

3.2 DA14585 IoT MSK Hardware Features

- Highly integrated Dialog Semiconductor DA14585 Bluetooth® Smart SoC
- Stand-alone module

- Low cost due to printed antenna
- Low cost PCB
- Combined accelerometer/gyroscope sensor unit
- Combined sensors:
 - Accelerometer and gyroscope sensor unit
 - Indoor Air Quality, Temperature, Humidity and Pressure
 - Ambient Light Sensor and Proximity
- Access to processor via JTAG and UART from the enclosure
- Programmable RF power up to +9.3dBm
- Three Led indicators
- General purpose push button
- Expansion slots
- Powered by two low cost AAA alkaline batteries

3.3 DA14585 IoT MSK Hardware Architecture

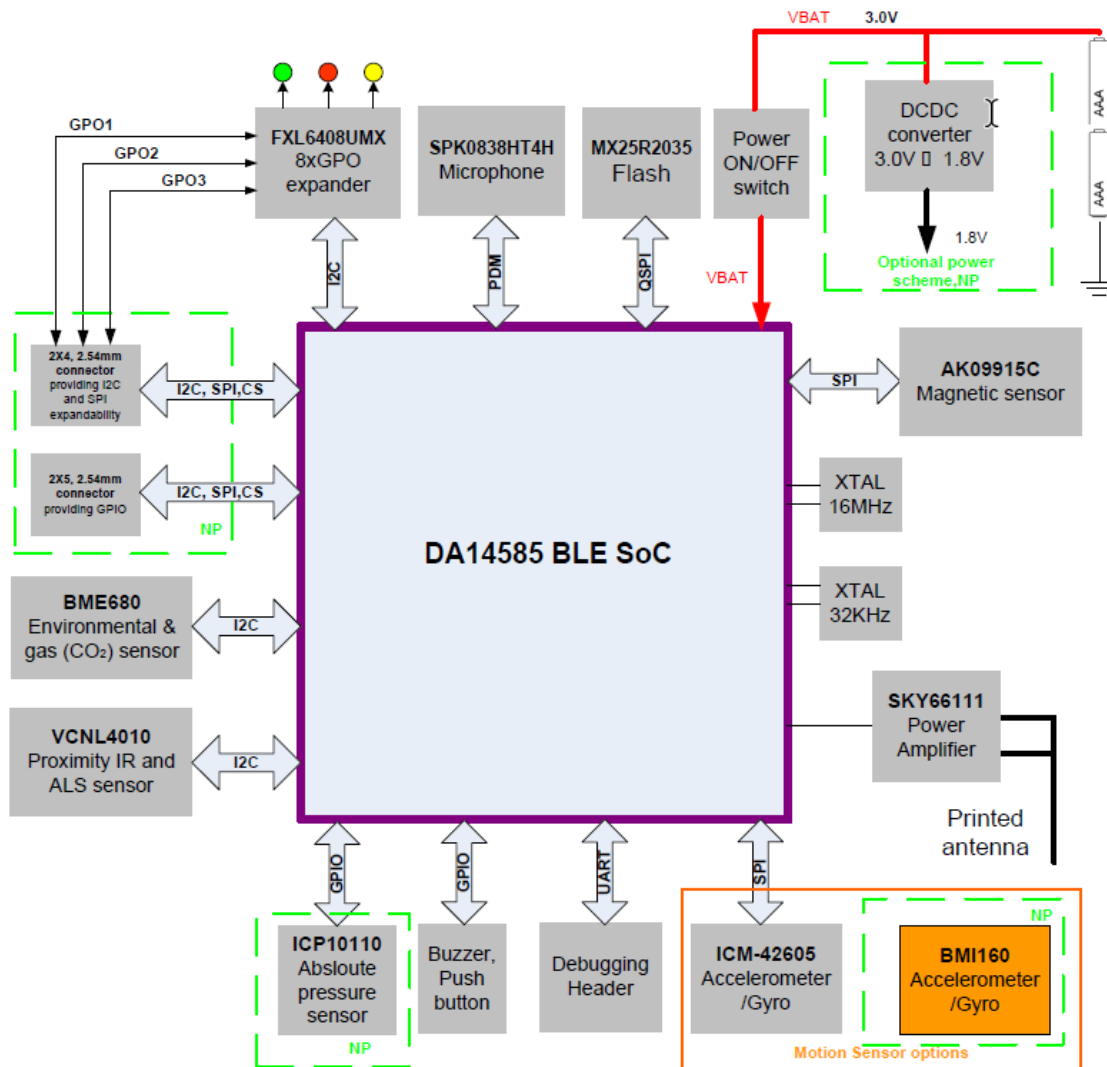


Figure 2: Hardware Architecture of DA14585 IoT MSK

4 Quick Start Guide

The quick start guide describes an easy way to customize the Dialog MSK reference design SW and explains how to program the MSK reference design HW. The latest software can be downloaded from the [Dialog support portal](#).

For additional information about the software architecture, please refer to [1] and [2].

4.1 Setting up the Hardware

To program the Flash or OTP memory of the DA14585 IoT MSK reference design, connect the reference hardware (please refer to [14] for more information) to the Communication Interface Board (CIB, please refer to [8] for more information) using the IDC10-1.27 mm cable (Figure 3).

Before connecting DA14585 IoT MSK to the CIB with the IDC cable, turn the side switch of the MSK to OFF to cut off the battery and turn the CIB switch SW2 to On to provide power to the MSK.

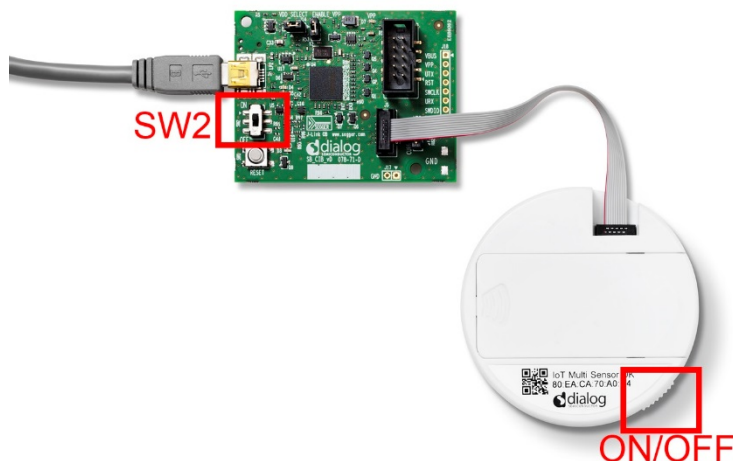


Figure 3: Setting up the Hardware

4.2 Downloading and Programming

4.2.1 Prerequisites

Users should install the following tools in order to be able to create and burn new images:

- A Development PC running Windows operating system. Please refer to *section 7* of [4] for details.
- Dialog's SmartSnippets™ Studio from <https://support.dialog-semiconductor.com>. Please refer to *section 8* of [4] for details.
- KEIL μVision IDE from <https://www.keil.com/demo/eval/arm.htm>. Please refer to *section 8* of [4] for details. The μVision version recommended to use is v5.23.0.0.

4.2.2 Opening, Building and Downloading the Projects

For details on how to open, build, and debug an application using the Keil μVision environment, please refer to *section 9* of [4]. To open one of the five available reference applications (IoT sensors, Smart Tag, altbeacon_dynamic, ibeacon_suota_button, eddy_uid_url_tlm) that work in DA14585 IoT MSK HW, go to <root_folder>/projects/target_apps (Figure 4).

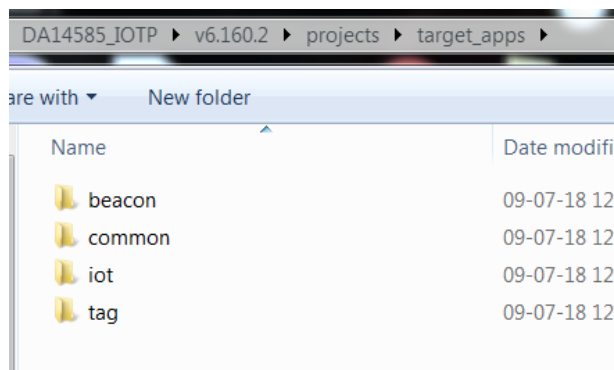


Figure 4: MSK Target Apps Folder

From here, one can choose the desired application. For example, [Figure 5](#) shows the interface after users select "**tag**" and navigate to "`..\projects\target_apps\tag\smart_tag\Keil_5`".

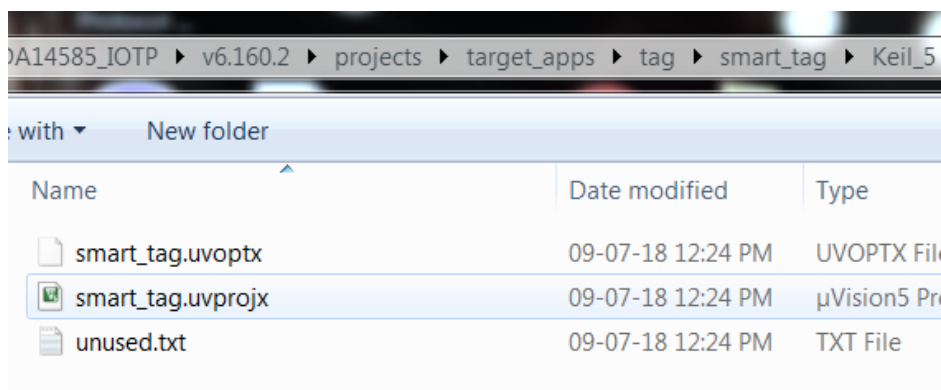


Figure 5: Smart Tag Folder

Opening `smart_tag.uvprojx` opens the project in the Keil uVision environment. Similarly, to open any of the five available reference applications, look for the "**Keil**" folder in every folder, and open the corresponding `.uvprojx` file.

Once the desired reference application has been built in the folder "`..\projects\target_apps\tag\smart_tag\Keil_5\out_585`", one can find the generated `.hex` file ([Figure 6](#)).

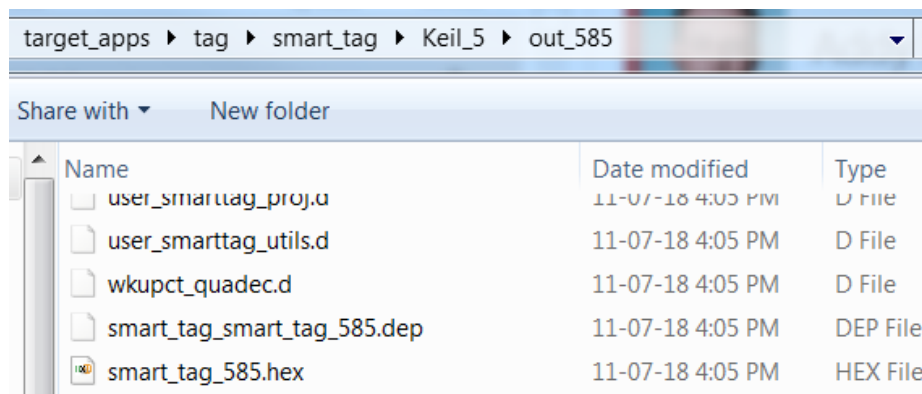


Figure 6: Smart Tag Output Folder

The .hex file can be used to generate images using the mkimage console application. [Appendix C](#) describes how to use the mkimage application and various script .bat files provided in ".\Utilities\mkimage_utils_scripts" to create the desired images. These generated images are in .bin format.

4.2.3 Burning Images in Flash

There are two different types of images to be burnt in flash:

- The multi-images for IoT Sensors and Smart Tag reference applications
 - This is a full flash image that contains two images of the application and a product header at the end. The SUOTA updates one of the two images, and the secondary bootloader (see [Appendix A](#)) on boot selects the most recent valid application image to load in SRAM.
- The whole images for Beacon reference applications
 - Essentially a whole image is a multi-image (two images with a product header) with a configuration struct for the beacon.

In the examples of Smart Tag and IoT sensors, users can run the provided scripts described in [Appendix C.1](#) with the provided necessary files in the same folder (see [Table 85](#)).

In the Beacon examples, users can optionally modify the provided default configuration struct (dev_conf_struct_default.cfg) to create a customized configuration or use it as is. Running the corresponding script ([Table 85](#)) will generate the desired whole image.

The .bin files created by the mkimage scripts or the mkimage application different modes must then be written in flash. Having either the Beacon whole images or the Smart Tag/IoT sensors multi-images ready makes it easy for users to burn the .bin files in flash (see [Appendix D.2](#)). Detailed information on how to customize images, creating a custom configuration struct for Beacon, and others can be found in [Appendix D](#). [Appendix B](#) describes how the memory map is structured depending on the application.

4.3 Software over the Air Updates

The Software Update over the Air (SUOTA) profile is a custom profile designed by Dialog Semiconductor to perform software upgrades on DA1458x devices. All DA14585 IoT MSK reference applications except AltBeacon and Eddystone can be upgraded by the SUOTA profile from a central device.



Figure 7: SUOTA Feature

Please define the flag in [Table 1](#) to use the SUOTA profile.

Table 1: SUOTA Profile Enabling

Parameter	Variable/Macros	Source File Name
Enable SUOTA	CFG_PRF_SUOTAR	user_profiles_config.h

An external SPI Flash memory shall be connected to the DA14585. During initial use of the Beacon reference design, the SPI Flash memory shall be prepared for a dual image.

According to the specification of the SUOTA profile, two roles have been defined:

- The SUOTA Initiator:
 - It operates as a GATT client transmitting the software upgrade file
- The SUOTA Receiver:
 - It operates as a GATT server, which receives and applies the software upgrade.

The MSK reference application software implements the SUOTA Receiver role. For more details about the SUOTA profile, please refer to *section 8.7* in [\[1\]](#).

The software images downloaded over the SUOTA profile are stored in the external SPI Flash memory. The [SmartSnippets™](#) tool can be used for programming the boot loader in OTP and for programming the product header and the dual images in SPI Flash.

To support SUOTA, a dual image boot loader must be programmed in the OTP memory of the DA14585 and the corresponding non-volatile memory map must be applied in SPI Flash memory during the production phase.

The MSK software sets the SPI Flash memory in power down mode to reduce the power consumption. The secondary boot loader in OTP will ensure a correct enabling of the OTP after wake-up from Deep Sleep mode. Instructions on how to create a secondary bootloader are provided in *Appendix A* in [\[2\]](#).

The details of the specific memory map and the procedure to create and write an image compliant to the required memory map are described in detail in [Appendix C](#) and [Appendix D](#).

Instructions on how to use the SUOTA Application for Android can be found in [Appendix E](#).

4.4 Magnetometer Auto-Calibration Procedure

When operating in auto-calibration mode, it is necessary to perform a calibration procedure in order provide the calibration algorithm with enough raw sensor data for it to accurately determine the hard and soft iron magnetometer distortions that exist and calculate the calibration coefficients required to compensate for them. Until this calibration procedure has been completed, the magnetometer will not be properly calibrated and drift artefacts will be observable in the 3D orientation visualization.

This calibration procedure involves rotating the board through a representative set of orientations so that a complete sphere of magnetometer vector positions has been captured. In order to produce sufficient data for successful calibration, it is recommended to rotate the board randomly through all orientations as quickly as possible for about 10 to 20 seconds.

The calibration procedure should be performed in a uniform and static magnetic environment. When the magnetic environment changes the calibration procedure must be repeated in order to allow the algorithm to adapt.

When operating in an environment with much variability in the strength and direction of the magnetic field, it is recommended that the auto-calibration procedure is first performed in a controlled location. Auto-calibration should then be disabled by setting the 'Update' control setting to OFF to prevent further modification of the calibration coefficients. This will prevent the constantly changing magnetic environment from upsetting the calibration algorithm and introducing drift artefacts in the 3D visualization.

4.4.1.1 Prerequisites for Magnetometer Calibration

- The device should be away from any sources of magnetic interference. Electric or electronic devices located near the IoT sensor might influence the outcome. Also ferromagnetic materials used in office desks or chairs should be kept away from the device while calibrating.
- The device should be kept within a single location while calibrating it. The calibration algorithms are designed to compensate within a stable and uniform magnetic environment. When sudden changes in the magnetic environment occur, it will take some time for the calibration algorithm to adapt.
- The magnetization level of the battery is of importance, but the calibration algorithm will fail, even without attached battery, when the other conditions are not met. The temperature of the battery is also important and should be kept stable during the calibration process.
- Make sure the IoT Multi Sensor DK batteries are not depleted.

4.4.1.2 Procedure of Magnetometer Calibration

1. Connect to the device using the provided iOS or Android IoT application.
2. Stop the device, go to the basic configuration page.
3. Select "Reset to Defaults".
4. Select Calibration Mode: Continuous and Auto Calibration Mode: Basic or SmartFusion.
5. Press "Write configuration to NV" in order to preserve the settings.
6. Press START and rotate the device with random orientations with 1 to 2 rotations per second for 15 to 20 seconds. When rotation is stopped, the 3D image will stabilize and the magnetometer value will be in effect in the sensor fusion algorithm.
7. If the calibration fails, restart the procedure after checking the battery and changing the calibration environment (sources of magnetic interference).
8. Every time STOP is pressed, the calibration coefficients are saved.

4.4.1.3 Magnetometer Sensor and Calibration State Reporting

In order to provide users some visibility of the state of the magnetometer sensor and the applied calibration, various indicators have been provided in the app's user interface.

In the compass pane on the sensor screen, two icons can be seen:

The  icon indicates whether or not calibration has been applied to the magnetometer data. If struck out, it means that the magnetometer data is uncalibrated, either because calibration is disabled or the 'apply' flag is not set.

The  icon indicates whether or not the sensor data is valid. If visible, it means that the magnetometer is either starting up, saturated, or the calibration routine has detected some other issue with the magnetometer data.

Overlaid on all screens is an  icon indicating the state of the calibration routine. The various states include:

Cross: Calibration is disabled.

Question Mark: The calibration routine is initialized and is not yet applied to the magnetometer data.

Red Circle: The quality of the applied calibration is poor.

Yellow Circle: The quality of the applied calibration is OK but not optimal.

Green Circle: The quality of the applied calibration is good.

5 IoT Sensors Reference Application

5.1 Software Features

This section explains the advanced software features of Dialog's DA14585 IoT MSK reference applications:

- GAP Peripheral role.
- GATT Based bidirectional data and control transfers.
- Sensor fusion library with update rate from 10 Hz to 100 Hz and selection between three different sensors: accelerometer, gyroscope and magnetometer.
- Three environmental sensors: pressure, humidity and temperature.
- Ambient Light and proximity sensor.
- Indoor Air Quality feature.
- Switch between Sensor Fusion data and raw data modes.
- Notifications for sensor data streaming.
- Sensor range and update rate control.
- Configuration set, save and restore.
- Three calibration modes for magnetometer.
- Gyroscope drifting elimination algorithm.
- iOS and Android Central Application.
- Auto sleep and motion wakeup.

5.2 Software Architecture

5.2.1 Project Files

The project resides in "projects\target_apps\iot\iot_585\Keil_5\iot585.uvprojx". The same project supports both Raw and Sensor fusion operation. This mode of operation is switchable from the central device application.

Project specific folders are:

- Sensor Fusion library files: iot\common_iot_files\lib
- Sensor Calibration files: target_apps\iot\common_iot_files\src\calibration
- IoT Profile files: target_apps\iot\common_iot_files\src\profiles

The project shares the following common folders:

- Driver Adaptation Layer files: projects\target_apps\common\src\Driver_Adaptation_Layer
- Drivers folders: projects\target_apps\common\src\drivers
- Sensor Interface folder: projects\target_apps\common\src\Sensors_Interface
- Common SDK folder: SDK_585\sdk\

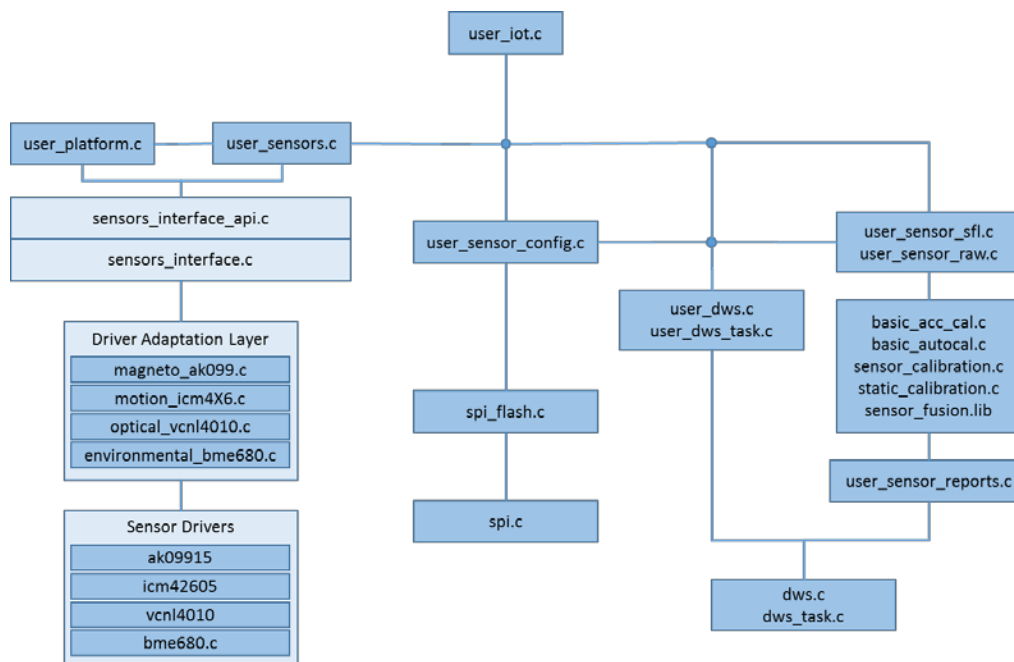


Figure 8: DA14585 IoT MSK Software Architecture

5.2.2 Source Files

The reference applications of DA14585 IoT MSK are developed based on the DA14585 SDK software. For more information on the SDK, please refer to [1] and [2].

The source code files of the reference applications are briefly described in [Table 2](#).

Table 2: Source Files for DA14585 IoT MSK: Overview

Group	Files	Description
User Platform Files (IoT DK)	user_periph_setup.c i2c_gpio_extender.c user_iot_dk_utils.c sensors_periph_interface.c	Peripheral setup, Range extender, Led control functions, Sensors i2c-spi communication helper functions.
User Application (IoT DK)	user_dws.c user_dws_task.c user_iot.c	Main application files. File user_iot.c contains the connect/disconnect/advertise handlers. Files user_dws.c and user_iot_task.c contain the user interface to the DWS.
User Profiles (IoT DK)	dws.c dws_task.c	Service functions and FSM handlers.
Sensor Calibration (IoT DK)	basic_acc_cal.c basic_autocal.c static_calibration.c sensor_calibration.c smartfusion_autocal.lib	Accelerometer calibration functions. Magnetometer calibration functions, specific to each calibration mode.

DA14585 IoT Multi Sensor Development Kit Software Reference Applications

Company Confidential

Group	Files	Description
User Sensors (IoT DK)	user_sensor_config.c user_sensors.c user_sensor_reports.c user_sensor_raw.c user_sensor_sfl.c	Sensor initialization. Sensor state machine. Main application cb functions.
User sfl	user_sfl_util.c sensor_fusion.lib	Files to process sensor data
Sensor interface (IoT DK)	sensors_interface_api.c sensors_interface.c	New interface to simplify sensor integration to project.
Driver Adaptation Layer (IoT DK)	magneto_ak099.c motion_icm4X6.c optical_vcnl4010.c environmental_bme680.c	This layer defines a common instruction set for the "Sensor interface" to access the sensors.
bsec_lib (IoT DK)	user_iaq.c libalgobsec_full.lib	Functions that estimate the air quality output.
wkup_adapter (IoT DK)	wkup_adapter.c	A wrapper to the "wkupct_quadec" to further extend the possibilities of the module.
Drivers (IoT DK)	bme680 folder icm426xx folder ak09915 folder vcnl4010 folder	These folders contain the factory drivers of the sensors.

Table 3: Source Files Specific to DA14585 IoT MSK

File Name	Description
user_iot.c	Contains all top level BLE callback functions for connection, disconnection and advertising.
user_dws.c user_dws_task.c	Provide the user space access to the DWS custom service.
user_sensors.c	Contains sensor initialization functions. Handles sensor data on application level. Main application's finite state machine function "user_iot_timer_handler" assists sensor calibration.
user_sensor_reports.c	The user space functions that are specific for sending reports (sensor and sensor fusion data) to the central device.
user_sensor_sfl.c user_space_raw.c	User space sensor data processing.
basic_acc_cal.c	Accelerometer calibration functions.
static_calibration.c basic_autocal.c sensor_calibration.c smartfusion_autocal.lib	The calibration functions for the three modes, static, basic auto, and SmartFusion auto.
user_sfl_util.c sensor_fusion.lib	Sensor fusion library API.

sensors_interface_api.c sensors_interface.c	Expose the Sensor Interface API to the application user. Core methods to implement the Sensor Interface module functionality.
icm426xx_impl.c bme680_imp.c ak09915_impl.c vnc14010_impl.c	Driver functions for each sensor that act as middleware for the "driver adaptation layer". These drivers make use of the actual manufactured drivers and use only functions that are really needed for this implementation, not all available features.

The project contains a header file for each source file that contains function prototypes and definitions. Some header files contain information that is essential for the operation and configuration of the IoT project. The header files reside in the folder `projects\target_apps\iot\iot_585\src\config`. The header files are listed in [Table 4](#).

Table 4: Header Files for the Configuration of DA14585 IoT MSK

File Name	Description
da1458x_config_basic.h (SDK)	Basic configuration of DA14585 for security, watchdog, and print functions. Furthermore in this file users can control the sensors that are included in the project.
da1458x_config_advanced.h (SDK)	Advanced features of DA14585, such as low power clock source.
user_config.h (SDK)	Configure data related to sleep modes, advertise, and parameter update.
user_app_iot_config.h (IoT DK)	Application specific definitions: sensor calibration parameters, motion wakeup, advertising time, and LED signaling.
user_callback_config.h (SDK)	Application callback functions.
user_modules_config.h (IoT DK)	User related module activation/deactivation.
user_periph_setup.h (IoT DK)	Peripheral related definitions (GPIO).
user_profiles_config.h (IoT DK)	Included profiles.
user_dws_config.h (IoT DK)	Definition of DWS Service.

5.2.3 Application Configuration

A group of compilation switches allows control of the application's behavior. The most important switches are listed in [Table 5](#).

Table 5: Configuration Parameters

Name	Default value	Description
user_config.h		
app_default_sleep_mode	ARCH_EXT_SLEEP_ON	This controls whether the processor to enter sleep mode or not. Use ARCH_SLEEP_OFF when connecting a SW debugger.
USER_DEVICE_NAME	"IoT-585"	The advertising name string.
connection_param_configuration	- Min-Max connection interval: 20 ms to 40 ms, - Latency: 4 (events missed), - Supervision timeout: 2 s.	Recommended settings.
da1458x_config_basic.h		
CFG_PRINTF	DISABLED	Enables/disables the use of UART debug messages.
CFG_WDOG	ENABLED	Enables the Watchdog timer.
VCNL4010_OPTO_SENSOR_AVAILABLE	ENABLED	Enables/disables the opto sensor
AK099XX_MAGNETO_SENSOR_AVAILABLE	ENABLED	Enables/disables the magnetometer sensor
ICM4XX_ACCEL_SENSOR_AVAILABLE	ENABLED	Enables/disables the accelerometer - gyroscope sensor
BME680_ENVIRONM_SENSOR_AVAILABLE	ENABLED	Enables/disables the environmental sensor
IAQ_ENABLED	ENABLED	Enables/disables the indoor air quality algorithm
IAQ_RESTORE_STATUS_ENABLE	ENABLED	Enables/disables saving the indoor air quality algorithm state
CFG_RANGE_EXT	BYPASS	Configures the range extender's power value. For more information please refer to [6]
da1458x_config_advanced.h		
CFG_LP_CLK	LP_CLK_XTAL32	LP_CLK_XTAL32: Use external XTAL32 as a low power clock (recommended). LP_CLK_RCX20: Use internal RCX20 as a low power clock.
CFG_NVDS_TAG_BD_ADDRESS	{0x99, 0x00, 0x00, 0xCA, 0xEA, 0x80}	Defines the BD address if it is not programmed in OTP.

Name	Default value	Description
user_app_iot_config.h		
FAST_ADV_INTERVAL	160 (= 100 ms)	The advertising interval. Unit: 0.625 ms.
ADV_TIME_OUT	6000 (= 60 s)	The time that the device advertises before it goes to sleep. Unit: 10 ms.
ALWAYS_ADVERTISE	DISABLED	If enabled the system will remain in advertising state and never go to sleep mode.
USE_FAST_ACC_CAL	ENABLED	If enabled the accelerometer calibration becomes available.
GYRO_SDC_ENABLED	ENABLED	If enabled the gyroscope drift compensation is applied.
USE_SPI_FLASH_CONFIG	ENABLED	If enabled external flash will be active to read & store configuration parameters.
MAGNETO_CAL_ENABLED	ENABLED	Enables the calibration function of magnetometer. For more information see section 5.9.1

5.3 Operation Overview

5.3.1 General Description

A general view of the application flow is presented in Figure 9.

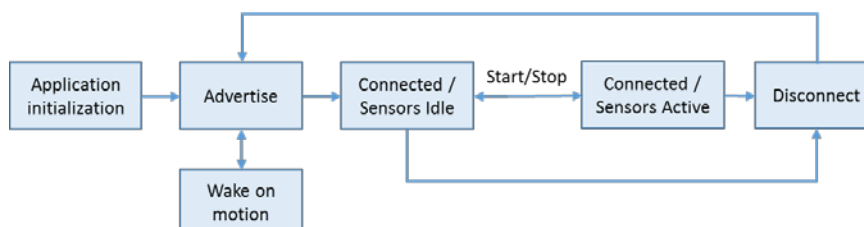


Figure 9: General Application Flow of DA14585 IoT MSK

5.3.2 Application initialization

This block refers to the entry point of the user application using the function `user_iot_app_on_init()` located in "user_iot.c" file. This handles:

- Initialize processor peripherals, `periph_init()`.
- Initialize radio extender GPIOs, `init_ext_gpio()`.
- Disable all external sensors and turn off all LEDs to reduce power consumption, `clr_led(ALL_LED)`.
- Clear sensor interface variables, `wkup_ad_init()`.
- Load sensor configuration from flash if available, `user_periph_sensors_initialize()`.
- Start BLE parameter updating.

5.3.3 Advertise

In this state, DA14585 IoT MSK will be visible to client applications.

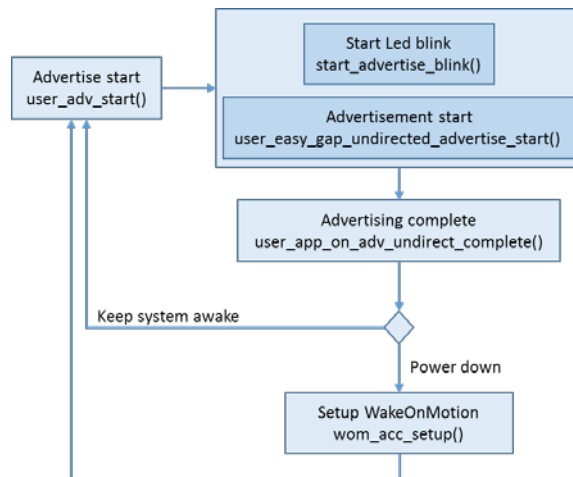


Figure 10: DA14585 IoT MSK in Advertise state

This block has the following operations:

- **"Advertise start"** block calls the `user_adv_start()` function. The following actions take place:
 - Start a timer to produce a LED blink using `start_advertise_blink()`. While the system remains in advertising state, the LED will blink periodically according to the `ADVERTISE_LED_ON_TIME` and `ADVERTISE_LED_OFF_TIME`.
Actual advertisement begins by sending a `gapm_start_advertise_cmd` to the GAP layer using the `user_easy_gap_undirected_advertise_start()` function.
- **"Advertising complete"**: Advertisement time period is determined by the value set on `ADV_TIME_OUT`.
 - When the timer expires, `user_app_on_adv_undirect_complete()` is executed to terminate advertising.
 - Routine `user_adv_start()` will be executed again if sleep mode is set to `ARCH_SLEEP_OFF`.
 - If sleep mode is set to `ARCH_EXT_SLEEP_ON`, the system will attempt to power off and enable wake on motion interrupt. This action will not apply if:
 - The user has explicitly set the system to stay "awake" by setting the `ALWAYS_ADVERTISE` switch.
 - The motion sensor is excluded from this project and therefore there are no means to wake the processor.
- **"Setup WakeOnMotion"**. The accelerometer for this purpose is configured for low power operation with the "anymotion" interrupt function set. The processor is allowed to go to Extended Sleep mode and wakes up only when it receives an interrupt from the accelerometer. Then the interrupt handler `wkup_intr_non_connected_cb()` is executed, which initiates advertising.

5.3.4 Connected/Sensors Idle

On connected state, the `user_on_connection()` function will execute all the necessary procedures in order to set up the system for run-time operation:

- Stops the advertising timer, `user_stop_adv_timer()`.

- Starts the blink timer to indicate connection event, `start_connection_blink()`.
- Sets the sensors to the default parameters, `user_periph_sensors_initialize()`.
- Suspend the sensors, `user_periph_sensors_suspend()`.
- Starts the BLE parameter update procedure, `app_easy_gap_param_update_start()`.
- Starts the main FSM.

5.3.5 Connected/Sensors Active

The sensors will enter active state when a "Start" command is sent from the application. This will switch the sensors FSM from `ACQUISITION_STATE_IDLE` to `ACQUISITION_STATE_INIT` using the `sensor_enable` variable.

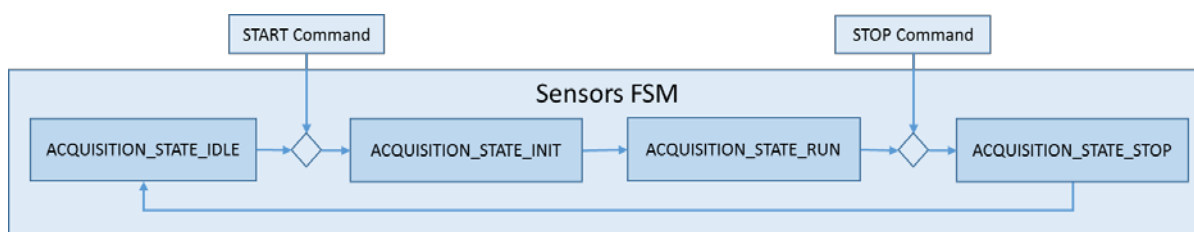


Figure 11: Activating sensors

In state `ACQUISITION_STATE_INIT` the following actions will take place:

- Initialize sensors control variable.
- Initialize gyro static drift compensation, if enabled.
- Initialize magneto calibration, if enabled.
- Initialize sensor fusion algorithm.
- Initialize all external sensors, if enabled.
- Switch to `ACQUISITION_STATE_RUN`.

After the system has switched to `ACQUISITION_STATE_RUN`, it will begin retrieving data from all active sensors and forward these to the client application. During this process, if `one_shot_cal_active` is enabled, magnetometer calibration will take place. This is a separate operation and will be discussed in section 5.9.

A general description for the data path in the "run" state is presented in the Figure 12. Note that "**Sensor Interface**" has a significant role in this operation and will be described in detail in section 5.5.

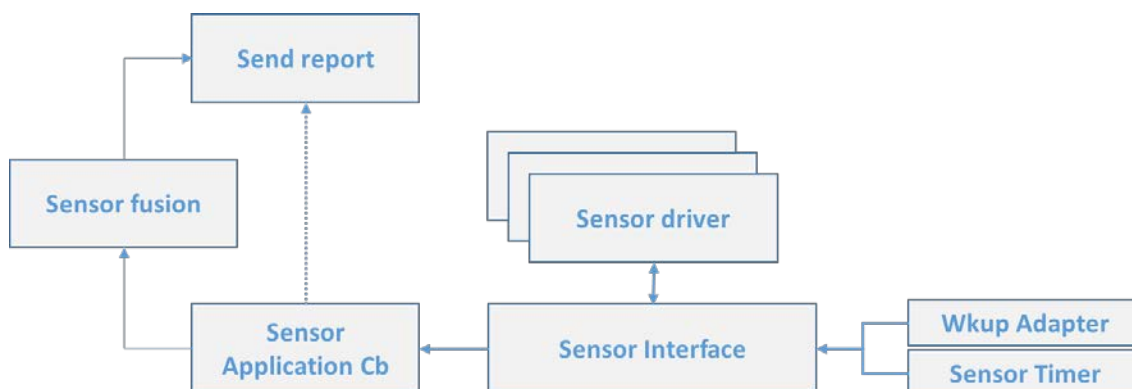


Figure 12: Overview of Sensors Data Path

Data ready indication for each sensor is triggered either from timer events or interrupt events. In both cases this indication is forwarded to the "**Sensor Interface**" to handle. This entity is responsible for discovering which sensor has created the event. This information will be used to select the correct sensor driver to retrieve data. The "**Sensor Interface**" is also aware of which application's callback function is bound to this sensor and will forward the data. The callback function is defined by users and from that point users will decide how to handle this information. In this project, the information is sent to client application using the reporting mechanism.

This path differentiates if the "**Sensor fusion**" library is used. Data from these devices (accelerometer, gyroscope, and magnetometer) will be processed in the Sensor fusion library and the outcome will be fed to the reporting mechanism.

5.3.6 Connected/Sensors Stopped

A "Stop" command terminates sensor operations and switch the system to ACQUISITION_STATE_STOP state.

If calibration takes place during the "run" state, stopping the sensors will initiate briefly the following:

- Save magnetometer or accelerometer calibration results in flash memory
- Inform the client application that calibration has finished.

Note this will be described extensively in section 5.9.1.

5.3.7 Disconnect

On this state, function user_on_disconnect will execute the following:

- Clear all LEDs and suspend sensors to minimize power consumption.
- Reinitiate the advertising procedure.

5.4 Wkup_adapter

This module is an extension to the "WakeUP Capture Timer" to further extend the possibilities of interrupt handling. The need for such an adapter emerges with the IoT project development where multiple interrupts are required to be supported as well as a simplified method to handle these is needed. Please note that currently this module will support interrupts only from port 1 and 2.

This module exposes the following functions to users:

- `wkup_ad_register_gpio`: This function will add a new entry to the `wkup_ad_entry` table. Available parameters are:
 - `sel_port`, `sel_pins`, `pol_pins`: Denote the port, pin, and polarity of the interrupt respectively.
 - `cb`: is the callback function that is called when the interrupt is activated.
 - `cb_inv`: is the callback function that is called when the interrupt with reverse polarity is activated.
- `wkup_ad_init`: This function will initialize the `wkup_ad_entry` table.
- `wkup_ad_remove_gpio`: This function will remove a specific entry from the table that matches the pin, port conditions.
- `wkup_ad_clear_all_gpio`: This function will clear all entries on the `wkup_ad_entry` table and further disable all active interrupts.
- `wkup_ad_cb`: This is the main callback function of the module. This routine will be executed for every registered interrupt event. Furthermore the appropriate callback function will be executed returning the port and pin that activate the interrupt.

Note 1 Currently this module supports interrupts in port 1 and 2. This hardware configuration does not support other ports.

5.5 Sensor Interface

5.5.1 General Description

The purpose of this module is to simplify integration and operation of any sensor. Users following the methods described should ignore the sensor complexity and treat currently installed or new sensors with the same approach.

Sensors are distinguished into two main categories according to their operation mode:

- **TIMED**: These sensors are configured to send data in time intervals defined by users.
- **INTERRUPT**: These sensors are configured to send data when an interrupt is triggered.

SI will treat these sensors differently. Each category has its own "Data Base" and "Data Available" mechanism ([Figure 13](#)).

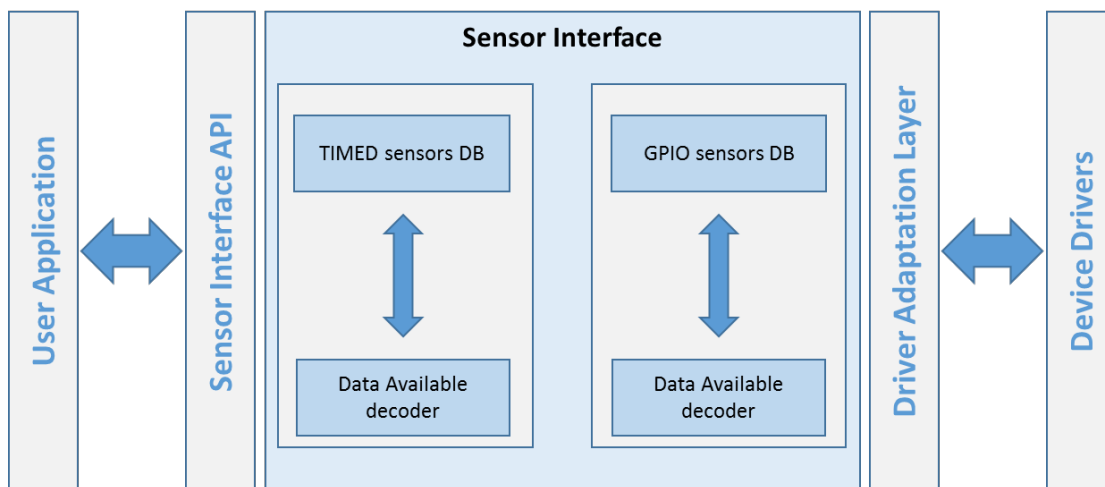


Figure 13: Sensors Interface Overview

5.5.2 Driver Adaptation Layer

"Driver Adaptation Layer" provides an interface for the SI to interconnect with device drivers. This layer exposes certain functions that are common for all device types. Users should create these functions:

- `<device>_setup`:
 - This function makes use of the "Driver specific settings" to configure the desirable operation of the device.
- `<device>_force_read`:
 - This function puts the device in forced mode.
- `<device>_read`
 - This function retrieves data from the device.
- `<device>_disable`:
 - This function stops the device operation and put it in minimum consumption mode.

5.5.3 Sensor Interface API

The "Sensor Interface API" is part of the "**Sensor Interface**" module and exposes the following functionalities to users:

- `si_reset()`.
 - The reset operation will clear all information in both TIMED and GPIO sensors table in the "**Sensor Interface**" module.
 - This action will take place when all sensors are suspended to avoid unintentional operation of any device. On system startup this information is already initialized.
- `si_register_sensor()`.
 - This command will register a new sensor in the module.
 - The "**Sensor Interface**" module will follow the path described in [Figure 14](#) to achieve this:

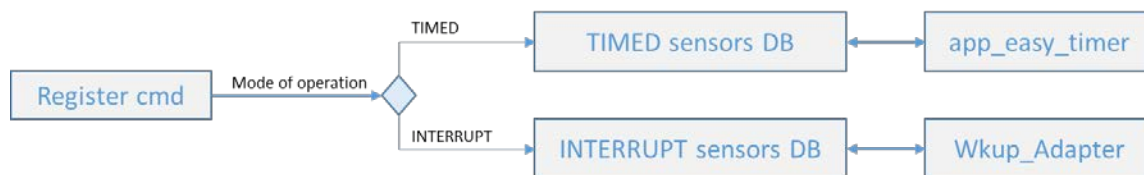


Figure 14: SI Registration Path

The first thing to inspect is the mode of operation. After this each device will be registered in the corresponding DB. TIMED based sensors will further make use of the "app_easy_timer" library. INTERRUPT based sensors will register necessary data to the "wkup_adapter" module that keeps track of all HW interrupts.

Each device type has separate and common configuration parameters.

- Available settings for TIMED devices:
 - operation_mode: This value will make clear this sensor will be used as a TIMED device and should be set as TIMER.
 - read_delay: The time period when users apply a forced mode to the device prior to reading the actual data.
 - periodic_read_interval: The time period when users expect to read data from the device.
 - force_read_fn: A callback function defined by users to setup the device in forced mode.
 - Figure 15 illustrates the timeline of a TIMED device to help users further understand the difference between force_read_fn and normal read_fn.

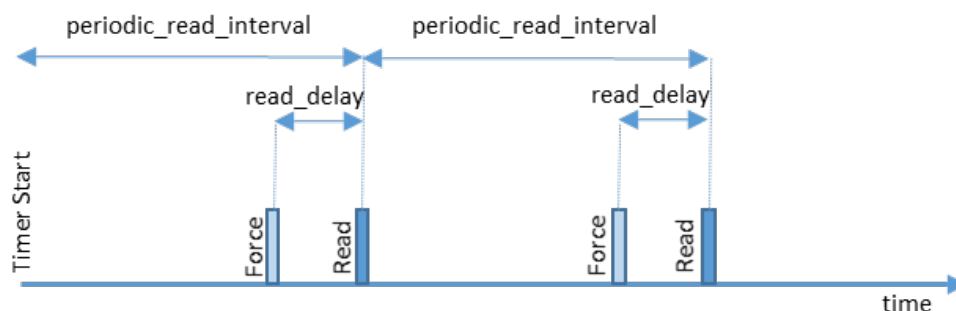


Figure 15: TIMED Sensors Timeline

- Available settings for INTERRUPT devices:
 - operation_mode: This value will make clear this sensor will be used as an INTERRUPT device and should be set as INTERRUPT.
 - sensors_pin_conf: The user should provide the pin, port, and polarity settings for the "wkup_adapter" to enable the interrupt.
- Common settings:
 - "Driver specific settings". Users are required to pass all necessary device specific information to the driver layer using this variable.
 - sensor_id. Each sensor should have a unique ID.
 - set_sensor_config. A user-defined callback function to setup the device in driver space. This function will make use of the previously defined Driver specific settings.

- `data_size`: The actual size of the data that are expected during a read operation.
- `read_fn`: A user defined callback function to read the actual data from the device.
- `pre_data_read_fn`: A user defined callback function that will be executed before the normal read.
- `si_send_command()`:
Provided that a sensor is already registered, the following commands are available:
 - `FORCE_TO_READ` will immediately execute the user-defined callback function `force_read_fn`. This operation applies to TIMED registered sensors.
 - `READ` will immediately execute the user-defined callback function `read_fn`. This operation applies to INTERRUPT registered sensors.

5.6 Device Drivers

5.6.1 Environmental Sensor

BOSCH BME 680 is an environmental sensor capable of:

- Measuring pressure
- Measuring humidity
- Measuring temperature
- Detecting volatile organic compounds

This IoT Sensors reference application also includes the "Bosch Software Environmental Cluster" library that can provide indoor air quality (IAQ) output.

Used functions:

- `bme680_init` performs software reset and reads the chip-id and calibration data from the sensor.
- `bme680_set_sensor_settings` is used to set the oversampling, filter and temperature, pressure, humidity, and gas selection settings in the sensor.
- `bme680_get_profile_dur` retrieves the profile duration of the sensor.
- `bme680_set_sensor_mode` sets the power mode of the sensor. In this implementation, two modes, forced and sleep, are used.
- `bme680_get_sensor_data` returns the sensor data with compensated values.

These driver files are located in "projects\target_apps\common\src\drivers\bme680".

Communication interface: I2C

5.6.2 Motion Sensor

TDK ICM-42605 is a 6-axis motion tracking device with the following features:

- 3-axis gyroscope
- 3-axis accelerometer
- User-programmable interrupts
- Wake-on-motion interrupt for low power operation of applications processor

Used functions:

- `SetupInvDevice426` setups the communication interface of the device, FIFO size, and callback function to handle data read.

- `ConfigureInvDevice` is used to configure the sampling rate and sensitivity of each sensor.
- `HandleInvDeviceFifoPacket426` is the call back function that retrieves available data from the sensor.
- `wom_acc_setup` pauses sensor operations and puts ICM-42605 in "Wake-on-Motion" state with minimum power consumption.
- `icm426xxInterrupt_wom_cb` is the callback function that executes right after a motion event.
- `DisableBothModules` set both sensors (accelerometer and gyroscope) in low power mode to reduce consumption.

These driver files are located in "projects\target_apps\common\src\drivers\icm426xx".

Communication interface: SPI

5.6.3 Magneto Sensor

Asahi Kasei AK09915 is 3-axis electronic compass IC with highly sensitive Hall sensor technology.

Used functions:

- `ak09915_init` configures operation modes of the sensor: single measurement, continuous, and power down.
- `ak09915_single_measurement_mode` retrieves data from the sensor.
- `ak09915_power_down_and_stop` stop all activities and set the sensor to low power mode to reduce consumption.

These driver files are located in "projects\target_apps\common\src\drivers\ak09915".

Communication interface: SPI

5.6.4 Optical Sensor

VISHAY VCNL4010, is an optical device with the following features:

- Proximity sensor
- Ambient light sensor
- Built-in infrared emitter
- Programmable LED drive current

Used functions:

- `vcnl4010_config` determines which sensor is enabled, proximity, ambient, or both, and set the power level of the infrared emitter.
- `vcnl4010_set_force_mode_fn` puts the sensor into forced mode.
- `vcnl4010_read_after_force_fn` retrieves data from the sensor.
- `vcnl4010_disable_sensor` sets the sensor to low power mode to reduce consumption.

These driver files are located in "projects\target_apps\common\src\drivers\vcnl4010".

Communication interface: I2C

5.6.5 GPIO Extender

ON Semiconductor FXL6408UMX is a low power GPIO expander with the following features:

- Eight independently configurable I/O ports.

- Low-power quiescent current of 1.5 μ A.

This device handles the three LEDs as well as the power output of the power amplifier.

Used functions:

- `set_ctrl_pwm_bp` sets range extender PWM bypass. For more information please refer to [6].
- `clr_ctrl_pwm_bp` clears range extender PWM bypass. For more information please refer to [6].
- `init_ext_gpio` initializes GPIO extender GPIOs.
- `set_led` turns on one single LED or all LEDs.
- `clr_led` turns off one single LED or all LEDs.

These driver files are located in "projects\target_apps\common\src\drivers\gpio_extender".

User functions are located in the "user_iot_dk_utils.c" file.

Communication interface: I2C.

5.6.6 Power Amplifier

Skyworks SKY66111-11 is a fully integrated RF Front End Module (FEM) designed for Smart Energy applications.

Used functions:

- No user API is available for this device. Power settings can be changed in "da1458x_config_basic.h" from RANGE_EXT_MODE to act as bypass or full power.

The driver files are located in "SDK_585\sdk\platform\driver\range_ext\sky66111".

5.7 Sequence Diagrams

This section outlines the sequence of events and processes when data is transferred from the sensors to the GATT in Dialog's wearable devices. The sequence diagrams also describes the data reporting from sensor fusion and environmental sensors.

5.7.1 Sensor fusion data reporting

Generation of sensor fusion data is triggered by the "Data ready" IRQ signal of the inertial sensor (accelerometer and gyro). Inertial sensor is registered in Sensor Interface as an interrupt driven device, hence the sensor interface calls subsequently the read function of the driver adaptation layer for sampling the sensor data and the application callback function to deliver sampled data to the application.

The application parses the obtained inertial sensor data and stores them in a dynamically allocated area. It combines the first accelerometer/gyro sample with stored magneto sensor sample and starts subsequent executions of Sensor Fusion algorithm in `user_sensor_fusion_process` function.

Then it adds the latest raw data samples of accelerometer/gyro/magneto sample and the Sensor Fusion result in a DWS report notification and passes it to GATT layer to be transmitted over the BLE interface.

Finally, it initiates the magneto data sampling by issuing a `FORCE_TO_READ` command to Sensor Interface through the `si_send_command` API function of the Sensor Interface. Sensor interface sets the magneto sensor in single shot mode and it waits for the "data ready" IRQ of the magneto sensor. When the interrupt is asserted, the sensor interface will fetch the data and call application callback which then stores the magneto sample to combine with the next bunch of inertial sensor samples.

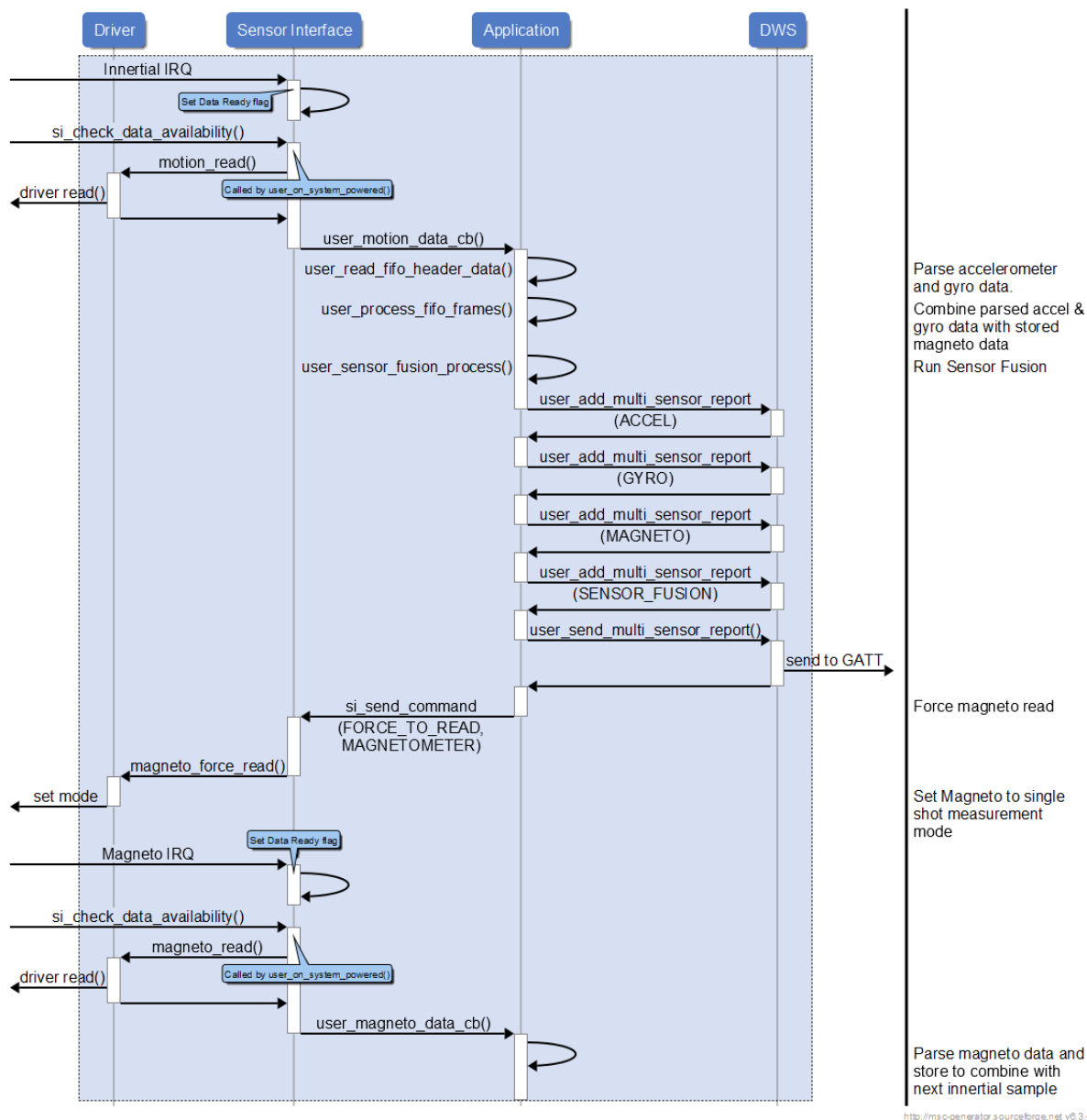


Figure 16: Sequence Diagram of Sensor Fusion Reporting

5.7.2 Environmental Data Reporting

Environmental sensor is registered in Sensor Interface in TIMER mode and forcing sensor to read and data read occurs in different time instances. Two functions of the adaptation layer of the environmental sensor are registered for this purpose: `environmental_force_read()` to force read operation on sensor and `environmental_read()` for reading sampled data.

The sensor interface will initiate two timer instances, expiring in a constant time gap equal to the `read_delay` declared at the registration of the sensor to Sensor Interface. Sensor interface calls `environmental_force_read()` on the expiration of the first timer and `environmental_read()` on the expiration of the second. When sensor data are available, the Sensor Interface will call the application callback function.

If "volatile organic compounds" sample is available, the application will run the air quality algorithm and add reports for temperature/humidity/pressure sensors. The Air Quality classification results in a DWS notification message and sends it to GATT.

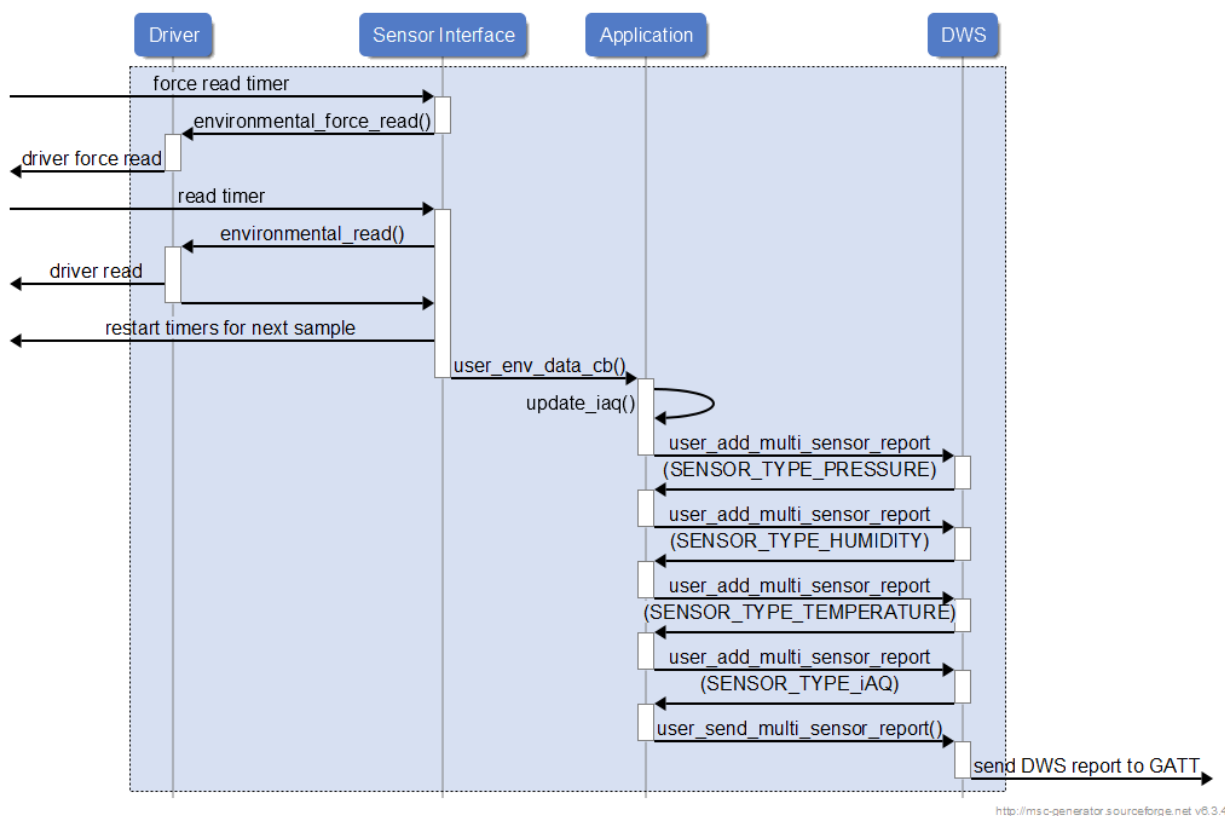


Figure 17: Sequence Diagram of Environmental Sensor Reporting

5.8 Dialog Wearable Service V2

The IoT Sensors reference application contains the Dialog Wearable Service version 2 (DWSv2) in order to transfer sensor and control data. This service includes a number of characteristics that are listed below. The DWSv2 provides a means to:

- Transfer raw and calibrated sensor data.
- Transfer sensor fusion data.
- Configure the device, such as setting of operating parameters.
- Control the device, such as start/stop and load/store to non-volatile memory.

The details of the DWS Service are outlined in [Table 6](#).

Table 6: DWSv2 Characteristics

Service/Characteristic	UUID	Properties (Note 2)	Size (B)	Description
wrbl_dws_svc	2EA7-8970-7D44-4BB-B097-2618-3F40-2400	RD	16	Service attribute

Service/Characteristic	UUID	Properties (Note 2)	Size (B)	Description
wrbl_dws_accel_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2401	NTF	25	Accelerometer Report. Legacy DWS compatibility, not used.
wrbl_dws_gyro_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2402	NTF	25	Gyroscope Report. Legacy DWS compatibility, not used.
wrbl_dws_mag_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2403	NTF	25	Magnetometer Report. Legacy DWS compatibility, not used.
wrbl_dws_baro_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2404	NTF	25	Barometer Report. Legacy DWS compatibility, not used.
wrbl_dws_hum_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2405	NTF	25	Humidity Report. Legacy DWS compatibility, not used.
wrbl_dws_temp_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2406	NTF	25	Temperature Report. Legacy DWS compatibility, not used.
wrbl_dws_sensf_char	2EA7-8970-7D4444BB-B097-2618-3F40-2407	NTF	25	Sensor Fusion Report. Legacy DWS compatibility, not used.
wrbl_dws_feat_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2408	RD	25	Device Features
wrbl_dws_control_char	2EA7-8970-7D44-4BB-B097-2618-3F40-2409	WR	32	Control Point
wrbl_dws_control_reply_char	2EA7-8970-7D44-4BB-B097-26183F40-240A	NTF	32	Command Reply
wrbl_dws_multi_sens_char	2EA7-8970-7D44-4BB-B097-26183F40-2410	NTF	109	Sensors Report

Note 2 RD: read, WR: write, NTF: notify, IND: indicate.

5.8.1 Features Report Structure

Upon connection, the central device reads the features characteristic (wrbl_dws_feat_char) in order to determine the capabilities and the firmware version of the connected device.

Table 7: Features Report Structure

Offset (B)	Name	Description
0	accelerometer_en	0: Accelerometer not present 1: Accelerometer present
1	gyro_en	0: Gyroscope not present 1: Gyroscope present
2	magn_en	0: Magnetometer not present 1: Magnetometer present

Offset (B)	Name	Description
3	pressure_en	0: Barometer not present 1: Barometer present
4	humidity_en	0: Humidity sensor not present 1: Humidity sensor present
5	temp_en	0: Temperature sensor not present 1: Temperature sensor present
6	s_fusion_en	0: Sensor fusion capability not present 1: Sensor fusion capability present
7 to 22	version[]	Version number, ASCII
23	device_type	Device version: 0: DA14580 IoT 2: DA14585 IoT MSK

5.8.2 Multi Sensor Report and Sensor Report

All sensor data are encapsulated in reports named "sensor reports" specific for each sensor type. The sensor reports are concatenated in a multi sensor report that is transferred as a BLE notification using `wrbl_dws_multi_sens_char` to the central device.

Table 8: Multi Sensor Report

Preamble (1 Byte)	Timestamp (1 Byte)	Sensor Reports (Up to 107 bytes)
Always 0xA5	Integer Number that increments after each report.	Concatenated sensor reports (Table 9).

Table 9: Sensor Report

Report ID (1 Byte)	Sensor State (1 Byte)	Calibration state (1 Byte)	Sensor Data (N Bytes)
1 to 24 (Table 10)	Value depending on sensor type.	Value depending on sensor type.	Depends on Sensor Type

Table 10: Report Types/Report ID's

Report ID	Report type
1	ACCELEROMETER_REPORT_ID
2	GYROSCOPE_REPORT_ID
3	MAGNETOMETER_REPORT_ID
4	PRESSURE_REPORT_ID
5	HUMIDITY_REPORT_ID
6	TEMPERATURE_REPORT_ID
7	SENSOR_FUSION_REPORT_ID
8	COMMAND_REPLY_REPORT_ID
9	AMBIENT_LIGHT_REPORT_ID

Report ID	Report type
10	PROXIMITY_REPORT_ID
11	GAS_REPORT_ID
12	iAQ_REPORT_ID
13	BUTTON_REPORT_ID
14-24	Reserved

5.8.2.1 Sensor Report for Accelerometer, Gyroscope, and Magnetometer

Table 11: Report Structure for Accelerometer, Gyroscope, and Magnetometer

Report field	Field size (B)	Description
ucReportId	1	1, 2 or 3 (Table 10)
snr_state	1	Sensor state (see Table 12)
cal_state	1	Calibration state, used only for Magnetometer. 0: Calibration State Disabled 1: Calibration State Initialized 2: Calibration State Bad 3: Calibration State OK 4: Calibration State Good 5: Calibration State Error
val_x	2	X axis value for the selected sensor
val_y	2	Y axis value for the selected sensor
val_z	2	Z axis value for the selected sensor

Table 12: Bitfield Structure for snr_state

Bit(s)	Field	Description
0	in_data_valid	Input (pre calibration) data valid flag
1	out_data_valid	Output (post calibration) data valid flag
2	cal_enabled	Calibration enabled flag
3	cal_settled	Calibration settled flag
4	cal_converged	Calibration converged flag
5:7	cal_mode	Calibration mode

5.8.2.2 Sensor Report for Temperature, Humidity, Gas, and Barometric Pressure

Table 13: Environmental Sensor Report

Report field	Field size (B)	Description
ucReportId	1	4, 5, 6, 11 (Table 10)
ucSensorState	1	Always 2 (Sensor Ready)
ucSensorEvent	1	Always 3 (Update Value)

Report field	Field size (B)	Description
Val32	4	Sensor value

5.8.2.3 Sensor Report for Indoor Air Quality (IAQ)

Table 14: Indoor Air Quality (IAQ) Report

Report field	Field size (B)	Description
ucReportId	1	12 (Table 10)
ucSensorState	1	Always 2 (Sensor Ready)
ucSensorEvent	1	Accuracy 0-3: 0: Unreliable 1: Low Accuracy 2: Medium Accuracy 3: High Accuracy
Val32	4	Sensor value 0-500

5.8.2.4 Sensor Report for Ambient Light and Proximity

Table 15: Sensor Report for Ambient Light and Proximity

Report field	Field size (B)	Description
ucReportId	1	9,10 (Table 10)
ucSensorState	1	Always 2 (Sensor Ready)
ucSensorEvent	1	Always 3 (Update Value)
Val32	4	Sensor value

5.8.2.5 Sensor Report for Button

Table 16: Sensor Report for Button

Report field	Field size (B)	Description
ucReportId	1	13 (Table 10)
ucSensorState	1	Button Status 0-1: 0: Released 1: Pressed
ucSensorEvent	1	Always 3 (Update Value)
Val32	4	Reserved

5.8.2.6 Sensor Report for Sensor Fusion

Table 17: Sensor Report for Sensor Fusion

Report field	Field size (B)	Description
ucReportId	1	7 (Table 10)
ucSensorState	1	Always 2 (Sensor Ready)

Report field	Field size (B)	Description
mc _{al} _state	1	Magnetometer calibration state: 0: Calibration State Disabled 1: Calibration State Initialized 2: Calibration State Bad 3: Calibration State OK 4: Calibration State Good 5: Calibration State Error
va _l _w	2	W sensor fusion value
va _l _x	2	X sensor fusion value
va _l _y	2	Y sensor fusion value
va _l _z	2	Z sensor fusion value

5.8.3 Report Structures for Configuration and Control

The DWSv2 Service provides the wrbl_dws_control_char (WR) and wrbl_dws_control_reply_char (NTF) characteristics for configuring and controlling the device. The device may also send unsolicited messages (such as STOP) to signal events.

Typically, the central device issues a command using the control characteristic and waits for a reply from the notification reply characteristic. The replies issued by the IoT sensor always start with byte 0x08 (COMMAND_REPLY_REPORT_ID, omitted from the following tables), followed by the Command ID and the command data.

Table 18: Report Structure for Commands

Report ID (1 byte)	Command ID (1 byte)	Command data (N bytes)
COMMAND_REPLY_REPORT_ID =8	See following tables.	Depending on Command ID, varies in length and field types.

Table 19: Commands

Command	Offset (B)	Description	Value
Start	0	Command ID	1
Start Command Reply	0	Command ID	1
	1	Running Status	1: Running
Stop	0	Command ID	0
Stop Command Reply	0	Command ID	0
	1	Running Status	0: Stopped
Read Parameters from Flash Memory	0	Command ID	2
Reset to Defaults (RtD)	0	Command ID	3
Store Basic Configuration in Flash Memory	0	Command ID	4
Store Calibration Coefficients and Control Configuration in Flash Memory	0	Command ID	5
Return Running Status	0	Command ID	6

Return Running Status Reply	0	Command ID	6
	1	Running Status	0: Stopped 1: Running
Reset Sensor Fusion and Calibration Configuration command	0	Command ID	7
Basic Configuration			

5.8.3.1 Start Command

Table 20: Start Command

Offset (B)	Description	Value
0	Command ID	1

Table 21: Start Command Reply

Offset (B)	Description	Value
0	Command ID	1
1	Running Status	1: Running

5.8.3.2 Stop Command

Table 22: Stop Command

Offset (B)	Description	Value
0	Command ID	0

Table 23: Stop Command Reply

Offset (B)	Description	Value
0	Command ID	0
1	Running Status	0: Stopped

5.8.3.3 Read Parameters from Flash Memory

Table 24: Read Flash Command

Offset (B)	Description	Value
0	Command ID	2

5.8.3.4 Reset to Factory Defaults

Table 25: Reset to Defaults (RtD) Command

Offset (B)	Description	Value
0	Command ID	3

5.8.3.5 Store Basic Configuration in Flash Memory

Table 26: Store Basic Configuration Command

Offset (B)	Description	Value
0	Command ID	4

5.8.3.6 Store Calibration Coefficients and Control Configuration in Flash Memory

Table 27: Store Calibration and Control Command

Offset (B)	Description	Value
0	Command ID	5

5.8.3.7 Return Running Status

Table 28: Return Running Status Command

Offset (B)	Description	Value
0	Command ID	6

Table 29: Return Running Status Reply

Offset (B)	Description	Value
0	Command ID	6
1	Running Status	0: Stopped 1: Running

5.8.3.8 Reset Sensor Fusion and Calibration Configuration

Table 30: Reset Sensor Fusion and Calibration Configuration command

Offset (B)	Description	Value
0	Command ID	7

5.8.3.9 Basic configuration
Table 31: Basic Configuration Command

Offset (B)	Description	Value
0	Command ID	10
1	Sensor Combination	Bit 0: Accelerometer Enable Bit 1: Gyroscope Enable Bit 2: Magnetometer Enable Bit 3: Environmental Sensor Enable Bit 4: Gas Sensor Enable Bit 5: Proximity Sensor Enable Bit 6: Ambient Light Sensor Enable Note: In order Sensor Fusion to operate only certain combinations are allowed regarding accelerometer, gyroscope, and magnetometer. The combinations allowed are: • Gyroscope only. • Gyroscope and Accelerometer. • Accelerometer and Magnetometer. • Accelerometer, Gyroscope and Magnetometer.
2	Accelerometer Range	0x03: 2G 0x05: 4G 0x08: 8G 0x0C: 16G
3	Accelerometer Rate	0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz 0x09: 200 Hz
4	Gyroscope Range	0x00: 2000 deg/s 0x01: 1000 deg/s 0x02: 500 deg/s 0x03: 250 deg/s 0x04: 125 deg/s
5	Gyroscope Rate	0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz 0x09: 200 Hz
6	Magnetometer Rate	Valid only if SF is off. 0: Accelerometer ODR/1 1: Accelerometer ODR/2 3: Accelerometer ODR/4 7: Accelerometer ODR/8

Offset (B)	Description	Value
7	Environmental Sensors Rate	1: 0.5 Hz 2: 1 Hz 4: 2 Hz
8	Sensor Fusion Rate	0: SF Off 10: 10 Hz 15: 15 Hz 20: 20 Hz 25: 25 Hz 50: 50 Hz 100: 100 Hz
9	Sensor Fusion Mode	Reserved
10	Sensor Fusion Raw Data Enable	0: Disabled 1: Enabled Using this bit the device may send decimated raw sensor data at SF ODR.
11	Reserved	NA
12	Gas Sensor rate	Not used, the gas sensor rate is always 0.33Hz (Low Power Mode).
13	Proximity/Ambient Light Mode	Not used, the mode is always polled.
14	Proximity/Ambient Light Rate	0: Sensor Off 1: 10Hz 2: 5Hz 3: 2Hz 4: 1Hz 5: 0.5Hz 6: 0.2Hz

5.8.3.10 Read Basic Configuration

Table 32: Read Basic Configuration Command

Offset (B)	Description	Value
0	Command ID	11

Table 33: Read Basic Configuration Command reply

Offset (B)	Description	Value
0	Command ID	11
1	Sensor Combination	Bit 0: Accelerometer Enable Bit 1: Gyroscope Enable Bit 2: Magnetometer Enable Bit 3: Environmental Sensor Enable Bit 4: Gas Sensor Enable Bit 5: Proximity Sensor Enable Bit 6: Ambient Light Sensor Enable Note: In order Sensor Fusion to operate only certain combinations are allowed regarding Accelerometer, Gyroscope and magnetometer. The combinations allowed are: • Gyroscope only. • Gyroscope and Accelerometer. • Accelerometer and Magnetometer. • Accelerometer, Gyroscope and Magnetometer.
2	Accelerometer Range	0x03: 2G 0x05: 4G 0x08: 8G 0x0C: 16G
3	Accelerometer Rate	0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz 0x09: 200 Hz
4	Gyroscope Range	0x00: 2000 deg/s 0x01: 1000 deg/s 0x02: 500 deg/s 0x03: 250 deg/s 0x04: 125 deg/s
5	Gyroscope Rate	0x06: 25 Hz 0x07: 50 Hz 0x08: 100 Hz 0x09: 200 Hz

Offset (B)	Description	Value
6	Magnetometer Rate	Valid only if SF is off. 0: Accelerometer ODR/1 1: Accelerometer ODR/2 3: Accelerometer ODR/4 7: Accelerometer ODR/8
7	Environmental Sensors Rate	1: 0.5 Hz 2: 1 Hz 4: 2 Hz
8	Sensor Fusion Rate	0: SF Off 10: 10 Hz 15: 15 Hz 20: 20 Hz 25: 25 Hz 50: 50 Hz 100: 100 Hz
9	Sensor Fusion Mode	Reserved
10	Sensor Fusion Raw Data Enable	0: Disabled 1: Enabled Using this bit the device may send decimated raw sensor data at SF ODR.
11	Reserved	NA
12	Gas Sensor rate	Not used, the gas sensor rate is always 0.33Hz (Low Power Mode).
13	Proximity/Ambient Light Mode	Not used, the mode is always polled.
14	Proximity/Ambient Light Rate	0: Sensor Off 1: 10Hz 2: 5Hz 3: 2Hz 4: 1Hz 5: 0.5Hz 6: 0.2Hz

5.8.3.11 Set Sensor Fusion Coefficients Command

Table 34: Set Sensor Fusion Coefficients Command

Offset (B)	Description	Value
0	Command ID	12
1	BETA A LSB	Sensor Fusion Beta A Gain (LSB)
2	BETA A MSB	Sensor Fusion Beta A Gain (MSB)
3	BETA M LSB	Sensor Fusion Beta M Gain (LSB)
4	BETA M MSB	Sensor Fusion Beta M Gain (MSB)
5:8	TEMPERATURE_REPORT_ID	Reserved

5.8.3.12 Read Sensor Fusion Coefficients

Table 35: Read Sensor Fusion Coefficients Command

Offset (B)	Description	Value
0	Command ID	13

Table 36: Read Sensor Fusion Coefficients Command Reply

Offset (B)	Description	Value
0	Command ID	13
1	BETA A LSB	Sensor Fusion Beta A Gain (LSB)
2	BETA A MSB	Sensor Fusion Beta A Gain (MSB)
3	BETA M LSB	Sensor Fusion Beta M Gain (LSB)
4	BETA M MSB	Sensor Fusion Beta M Gain (MSB)
5:8	Reserved	NA

5.8.3.13 Set Calibration Coefficients

Table 37: Set Calibration Coefficients Command

Offset (B)	Description	Value
0	Command ID	14
1	Sensor Type	2: Magnetometer
2	Q Format	Integer value
3:8	Offset Vector (3 × int16)	Integer value
9:26	Matrix 3 × 3 × int16	Signed fixed point value

5.8.3.14 Read Calibration Coefficients

Table 38: Read Calibration Coefficients Command

Offset (B)	Description	Value
0	Command ID	15

Table 39: Read Calibration Coefficients Command Reply

Offset (B)	Description	Value
0	Command ID	15
1	Sensor Type	Magnetometer = 2
2	Q Format	Integer value
3:8	Offset Vector 3 × 1 int16	Integer value
9:26	Matrix 3 × 3 int16	Signed fixed point value

5.8.3.15 Set Calibration Control Flags

Table 40: Set Calibration Control Flags Command

Offset (B)	Description	Value
0	Command ID	16
1	Sensor Type	2: Magnetometer
2:3	Calibration Control Flags	Byte 2: see Table 41 Byte 3: see Table 42
4:15	Calibration Parameters	See Table 43

Table 41: Calibration Control Flags #1

Calibration mode	Bit 0 reserved	Bit 1 reserved	Bit 2 offset apply	Bit 3 matrix apply	Bit 4 offset update	Bit 5 matrix update	Bit 6 init from static	Bit 7 offset post apply
Static	X	X	1: Yes	1: Yes	0: No	0: No	0: No	1: Yes
Basic Auto	X	X	1: Yes	1: Yes	1: Yes	1: Yes	0: No	1: Yes
SmartFusion Auto	X	X	1: Yes	1: Yes	1: Yes	1: Yes	1: Yes	1: Yes

Table 42: Calibration Control Flags #2

Calibration mode	Bit 0 reserved	Bit 1 reserved	Bit 2 reserved	Bit 3 reserved	Bit 4 reserved	Bit 5 reserved	Bit 6 converged (read only)	Bit 7 settled (read only)
Static	X	X	X	X	X	X	X	0: No
Basic Auto	X	X	X	X	X	X	0: No	1: Yes
SmartFusion Auto	X	X	X	X	X	X	1: Yes	1: Yes

Table 43: Calibration Parameters

Offset (B)	Static	Basic	SmartFusion	Description
0	Reserved	ref_mag	ref_mag	Reference magnitude
1	Reserved	mag_range	mag_range	Magnitude range
2	Reserved	mag_alpha	mag_alpha	Magnitude filter coefficient
3	Reserved	mag_delta_thresh	mag_delta_thresh	Magnitude gradient threshold
4	Reserved	Reserved	mu_offset	Offset update rate
5	Reserved	Reserved	mu_matrix	Matrix update rate
6	Reserved	Reserved	err_alpha	Overall error filter coefficient
7	Reserved	Reserved	err_thresh	Overall error threshold

5.8.3.16 Read Calibration Control

Table 44: Read Calibration Control Flags Command

Offset (B)	Description	Value
0	Command ID	17

Table 45: Read Calibration Control Flags Command Reply

Offset (B)	Description	Value
0	Command ID	17
1	Sensor Type	2: Magnetometer
2:3	Calibration Control Flags	Byte 2: see Table 41 Byte 3: see Table 42
4:15	Calibration Parameters	See Table 43

5.8.3.17 Fast Accelerometer Calibration

Table 46: Fast Accelerometer Calibration Command

Offset (B)	Description	Value
0	Command ID	18

Table 47: Fast Accelerometer Calibration Reply

Offset (B)	Description	Value
0	Command ID	18
1	Fast Calibration Status	0: Stopped 1: Started

5.8.3.18 Set Calibration Modes

Table 48: Set Calibration Modes Command

Offset (B)	Description	Value
0	Command ID	19
1	Calibration Mode for Accelerometer	Not used, reserved for future use.
2	Calibration Mode for Gyroscope	Not used, reserved for future use.
3	Calibration Mode for Magnetometer	0: None 1: Static 2: Continuous Auto 3: Auto One Shot
4	Auto Calibration Mode for Accelerometer	Not used, reserved for future use.
5	Auto Calibration Mode for Gyroscope	Not used, reserved for future use.
6	Auto Calibration Mode for Magnetometer	0: Basic Auto Calibration 1: Smart Auto Calibration

5.8.3.19 Read Calibration Modes

Table 49: Read Calibration Modes Command

Offset (B)	Description	Value
0	Command ID	20

Table 50: Read Calibration Modes Command Reply

Offset (B)	Description	Value
0	Command ID	20
1	Calibration Mode for Accelerometer	Not used, reserved for future use.
2	Calibration Mode for Gyroscope	Not used, reserved for future use.
3	Calibration Mode for Magnetometer	0: None 1: Static 2: Continuous Auto 3: Auto One Shot
4	Auto Calibration Mode for Accelerometer	Not used, reserved for future use.
5	Auto Calibration Mode for Gyroscope	Not used, reserved for future use.
6	Auto Calibration Mode for Magnetometer	0: Basic Auto Calibration 1: Smart Auto Calibration

5.8.3.20 Read Device Sensors

Table 51: Read Device Sensors Command

Offset (B)	Description	Value
0	Command ID	21

Table 52: Read Device Sensors Command reply

Offset (B)	Description	Value
0	Command ID	21

Offset (B)	Description	Value
1-17	Sensor Type	Physical Sensors : 0: None 1: Accelerometer 2: Gyroscope 3: Magnetometer 4: Barometer 5: Humidity Sensor 6: Temperature Sensor 7: GAS Sensor 8: Proximity Sensor 9: Button 10-24: Reserved Virtual Sensors: 64: Sensor Fusion 65: Reserved 66: Indoor Air Quality 67-74: Reserved

5.8.3.21 Read Software Version

Table 53: Read Application Software Version Command

Offset (B)	Description	Value
0	Command ID	22

Table 54: Read Application Software Version Command Reply

Offset (B)	Description	Value
0	Command ID	22
1-17	Version Number	Version number of application in ASCII representation, for example, "v6.160.2"

5.8.3.22 Start LED Blink

Table 55: Start LED Blink Command

Offset (B)	Description	Value
0	Command ID	23

5.8.3.23 Stop LED Blink

Table 56: Stop LED Blink Command

Offset (B)	Description	Value
0	Command ID	24

5.8.3.24 Set Proximity Hysteresis Limits

Table 57: Set Proximity Hysteresis Limits Command

Offset (B)	Description	Value
0	Command ID	25
1-2	Proximity Low Limit (proximity off)	0-65535
3-4	Proximity Low Limit (proximity on)	0-65535

5.8.3.25 Read Proximity Hysteresis Limits

Table 58: Read Proximity Hysteresis Limits Command

Offset (B)	Description	Value
0	Command ID	26

Table 59: Read Proximity Hysteresis Limits Command Reply

Offset (B)	Description	Value
0	Command ID	26
1-2	Proximity Low Limit (proximity off)	0-65535
3-4	Proximity Low Limit (proximity on)	0-65535

5.8.3.26 Calibration Complete

The calibration complete notification is only sent from the device to the central application when a calibration operation is completed.

Table 60: Calibration Complete Notification

Offset (B)	Description	Value
0	Command ID	27
1	Sensor Type	0: Accelerometer 1: Gyroscope 2: Magnetometer
2	Status	0: OK 1: Error

5.9 Sensor Calibration Library

5.9.1 Overview

The SmartFusion Sensor Calibration Library (SCL) provides a set of routines for calibrating MEMS sensors such as gyroscopes, accelerometers, and magnetometers. By applying these routines to captured sensor data, it is possible to compensate for the imperfections and distortion typically exhibited.

5.9.1.1 Modes of Operation

The routines provided by the Sensor Calibration Library have been designed with flexibility in mind and support various modes of operation depending on the specific characteristics of the sensors and the system requirements.

The supported calibration modes are as follows:

- Static Calibration Mode:
 - When the sensor distortions are measurable, stable, and consistent between devices, static calibration is the preferred mode of operation as it gives the best performance in terms of distortion correction.

- In this mode the calibration routine is initialized with static calibration coefficients which are then applied to the sensor data and do not change.
- These static calibration coefficients are typically calculated off-line by the device manufacturer by analyzing recordings of raw sensor data made under controlled conditions. The calibration coefficients can then be stored in either the device's firmware or non-volatile memory.
- Continuous Automatic Calibration Mode:
 - When the sensor distortions are unstable and/or inconsistent between devices, continuous automatic calibration is the preferred mode of operation as it allows the calibration coefficients to be determined automatically at runtime without requiring them to be built into the firmware or programmed into non-volatile memory.
 - In this mode the auto-calibration function continually monitors the sensor data for distortions and adapts the calibration coefficients to compensate for them.
- One-shot Automatic Calibration Mode:
 - When the sensor distortions are relatively stable in the short term, one-shot auto-calibration mode may be preferable.
 - In this mode the auto-calibration function is run upon device startup to determine the sensor distortions but is disabled once complete and suitable calibration coefficients have been calculated.

5.9.1.2 Calibration Routines

The Sensor Calibration Library provides a number of routines for both static and automatic calibration of three dimensional sensor data. Note that not all of these routines are appropriate for all types of sensor or can be used in all calibration modes.

The supported calibration routines are as follows:

- Static Calibration:
 - The static calibration routine applies user-defined static calibration coefficients in the form of a 3×3 transformation matrix and a 3-dimensional vector offset.
 - This routine is suitable for all types of sensor but is only appropriate for use in static calibration mode as the coefficients do not change.
- Basic Auto-Calibration:
 - The basic auto-calibration routine monitors the data captured from the sensor for basic offset and scaling distortions, calculating and applying appropriate calibration coefficients at runtime in order to compensate for them.
 - This routine is not designed to detect and compensate for more sophisticated types of distortions such as cross-axis, spherical and rotational distortions.
 - It presumes that the magnitude of the external stimulus to the sensor (such as magnetic or gravitational field strength) is constant and only varies according to device orientation.
 - Neither is it able to support scenarios where the distortions are not constant but vary over time.
- SmartFusion Auto-Calibration:
 - To address some of the shortcomings of the basic auto-calibration routine, a more sophisticated algorithm is also provided.
 - In addition to being able to calculate and compensate for basic offset and scaling distortions, this algorithm can also compensate for more sophisticated distortions such as magnetometer soft iron spherical distortions.

- It is also able to cope with gradual changes in sensor distortions and external stimulus over time, although some adaptation time is required.
- Static Drift Compensation:
 - This routine has been specifically designed to reduce gyroscope drift and is not appropriate for use with any other type of sensor.
 - Gyroscopes typically exhibit small biases, indicating slow rotation even when the device is stationary. This results in drift in calculated orientation when the gyroscope data is integrated.
 - Although these biases typically have large static components that can be compensated for using static calibration, there is often a residual bias that varies over time due to temperature or gravitational effects.
 - The static drift compensation routine provides tracks and removes these dynamic biases, eliminating drift when the device is stationary.
 - The algorithm also includes a noise gate to eliminate drift induced by random walk due to noise.

5.9.1.3 Calibration Procedure

It requires sampling an external stimulus, such as a magnetic or gravitational field, at a wide range of different orientations for the basic and SmartFusion auto-calibration routines to operate correctly. Therefore, it is necessary to rotate the device through to determine the distortions and calculate appropriate calibration coefficients. It is also important that these external stimuli remain constant in both magnitude and direction.

It is required to calibrate the magnetometer in a place where the magnetic field is reasonably strong and uniform. The rotation can be performed manually by randomly rotating the device until the calibration routine signals its completion.

It is important that the device isn't subjected to any lateral movement while being rotated during the accelerometer calibration, as lateral movements will distort the sampling of the gravitational field. It is therefore recommended to use a gimbal in conjunction with these routines to rotate the accelerometer around its center without any lateral movement.

When using the device in an environment where the external stimulus is constantly changing, such as when the magnetic field fluctuates with position or the device is subjected to lateral accelerations, it is recommended to perform an initial automatic calibration in one-shot mode under controlled situations and then switch to static calibration mode, using the resultant calibration coefficients.

5.9.2 API Usage

The various calibration routines provided by the Sensor Calibration Library are designed to work together and complement each other. Where they share common functionality, the routines use generic controls, parameters and code. The more advanced routines re-use the functionality of the basic routines. For example, the basic auto-calibration routine re-uses the static calibration routine to apply its calibration parameters to the sensor data.

More specifically, the parameter structures for all routines are designed to overlap so that they can share the same location in memory, thus sharing common parameters. This has been done to aid switching between calibration modes without unnecessary copying of parameter data between different routines and to minimize the memory footprint.

Wrapper code (`sensor_calibration.h|c`) has been provided, which implements the overlapping of the various calibration routines and provides a common interface through which they can be called.

5.9.2.1 Allocation

An instance of the appropriate calibration parameter structure (`static_calibration_params`, `basic_autocal_params`, `smartfusion_autocal_params`, or `static_drift_compensation_params`) or combined wrapper instance structure (`cal_instance`) shall be instantiated, either statically or on the heap, and shall be maintained during the life-cycle of sensor calibration processing.

5.9.2.2 Initialization

Before processing can be performed on an instance of a calibration routine, a subset of parameters within the appropriate parameter structure must be initialized. These parameters are as follows:

- **Generic:** Common to all
 - `in_data`: Pointer from which to read raw sensor input data
 - `out_data`: Pointer to which to write processed sensor output data
- **Static calibration:** In addition to the generic parameters:
 - `offset`: Offset vector coefficients to be subtracted from sensor data in same representation as raw sensor data
 - `matrix`: 3×3 matrix of signed fixed point coefficients to apply to sensor data.
 - `q_format`: Q format of matrix coefficients
 - `flags`: Control flags
 - `apply`: Flag to control whether or not calibration coefficients are applied
 - `matrix_apply`: Flag to control whether or not matrix coefficients are applied in addition to offset
 - `offset_post_apply`: Flag to control whether vector coefficients are applied before or after the matrix coefficients are applied
- **Basic Auto-calibration:** In addition to the static calibration parameters:
 - `ref_mag`: Reference magnitude indicating expected geomagnetic field strength in same representation as raw sensor data
 - `mag_range`: Q15 unsigned fixed point scaler indicating range +/- reference magnitude of valid sensor data
 - `mag_alpha`: Q15 unsigned fixed point coefficient controlling the filtering applied to the calculated sensor vector magnitude
 - `mag_delta_threshold`: Q15 unsigned fixed point threshold applied to the calculated gradient of the filtered sensor vector magnitude below which the algorithm is considered to have settled
 - `flags`: Control flags
 - `update`: Flag to control whether or not calibration coefficients are updated
 - `matrix_update`: Flag to control whether or not matrix coefficients are updated in addition to offset
- **SmartFusion Auto-calibration:** In addition to the basic auto-calibration parameters:
 - `mu_offset`: Q15 unsigned fixed point convergence speed of offset parameters
 - `mu_matrix`: Q15 unsigned fixed point convergence speed of matrix parameters
 - `err_alpha`: Q15 unsigned fixed point coefficient controlling the filtering applied to the overall error in calculated cost function

- `err_thresh`: Q15 unsigned fixed point threshold applied to the calculated overall error below which the algorithm is considered to have converged
- `flags`: Control flags
 - `init_from_static`: Flag to control whether or not calibration coefficients are initialized externally by user or should be reset by initialization routine
- **Static Drift Compensation**: In addition to the static calibration parameters:
 - `bias_thresh`: Threshold indicating the maximum range of dynamic shift in dynamic bias magnitude that can be tracked
 - `bias_alpha`: Q15 unsigned fixed point convergence speed of bias tracking
 - `bias_range_limit`: Q8 limit indicating maximum range of biases that can be tracked
 - `noise_gate_thresh`: Threshold indicating the magnitude of output data (after bias has been removed) below which the noise gate is applied
 - `flags`: Control flags
 - `update`: Flag to control whether or not the bias offset is updated
 - `init_from_static`: Flag to control whether or not calibration coefficients are initialized externally by user or should be reset by initialization routine. Note: This can be used to specify the static component of the bias allowing a tighter range of `bias_thresh` to be specified so that slower movements can be tracked.
- **Sensor Calibration Wrapper**: If the wrapper instance structure is used then the `mode` field should be initialized to indicate which calibration routine should be used. The valid modes specified by the `cal_mode` enumeration are:
 - `CAL_NONE`: No calibration routine is applied.
 - `CAL_STATIC`: The static calibration routine is applied.
 - `CAL_BASIC_AUTOCAL`: The basic auto-calibration routine is applied.
 - `CAL_SMARTFUSION_AUTOCAL`: The SmartFusion auto-calibration routine is applied.
 - `CAL_STATIC_DRIFT_COMPENSATION`: The static drift compensation routine is applied.

Once the calibration parameters have been initialized, it is necessary to call the appropriate routine's initialization function (if it has one) or the `cal_init()` function, when the wrapper is being used.

Note 3 It is presumed that un-initialized parameters will be reset to zero. It is therefore advised to use the `memset()` function to reset the parameter structure before initialization.

When using the wrapper, the `controls` field is a union of a 16-bit word and the overlapped calibration routine flags bit field structure, allowing the flags to be set individually or all at once.

For reasons of more optimal code generation for the ARM M0, the pointers to the input/output data vectors reference 32-bit signed integer types. However, the magnitude of the vectors that these values represent should not exceed a signed 16-bit range ($-32768 \leq |v| \leq 32767$).

Similarly the fixed point matrix coefficients also use 32-bit types, but should not exceed a signed 16-bit range. In cases where the matrix coefficients represent values greater than one, they should be scaled to a 16-bit range and the `q_format` parameter adjusted accordingly. For example, to represent coefficients in the range of ± 2.0 , `q_format` should be set to 14 and the coefficients scaled by 214. Values of ± 2.0 (32768 once scaled) should saturate at 32767 to prevent overflow.

Depending on the algorithm used to calculate the calibration coefficients, the offset vector may need to be applied before or after applying the matrix. The static calibration routine supports both methods, but this should be indicated by setting the `offset_post_apply` appropriately. When set, the offset vector will be applied after applying the matrix.

When applying static calibration it is necessary to initialize the calibration coefficients (offset and matrix) to be applied.

When applying SmartFusion auto calibration, in some cases it may be preferred to backup and restore the calculated calibration coefficients between instantiations, rather than starting from scratch each time. In this case it is necessary to prevent the calibration routine's initialization function from resetting the coefficients by setting the `init_from_static` flag.

In order to work correctly, the auto calibration routines need to know the expected magnitude of the vector produced by the sensor, once distortions have been removed. This reference magnitude is provided by initializing the `ref_mag` parameter. The value of this parameter should ideally be determined by performing an external measurement of the gravitational/geomagnetic field strength and expressing this in the same format and sensitivity as generated by the sensor. Failing this, the value can be set according to estimated or published values. Alternatively this value can be set to zero which enables a mode in which the calibration attempts to determine the sensor vector magnitude for itself.

As there is inevitably some margin of error in setting of the reference magnitude and potentially quite a lot of variability in the uncalibrated sensor vector magnitude, the `mag_range` parameter has been provided to specify the tolerance for the reference magnitude. This should be set in order to encompass the full range of expected acceptable magnitude values. For example, when the expected magnitude is 1000 and the desired tolerance $\pm 10\%$ (a range from 900 to 1100), the `mag_range` should be set to 0.1 (3277 in Q15 fixed point). In cases where the actual magnitude lies outside this range, the auto calibration routine will never complete. Conversely when the range is set too wide the calibration routine will detect completion too early and imperfect calibration will result. In case of noisy sensor data, the magnitude range can also be used to filter out outliers to some extent.

The SmartFusion auto calibration routine includes functionality for detecting when it has achieved convergence and the sensor distortions have been sufficiently removed. This is useful for determining when calibration is complete or when it has subsequently become unsettled. As the level of attainable convergence is sensitive to the amount of noise that exists in the sensor data, tuning parameters have been provided to tune the performance. The `err_alpha` parameter tunes the amount of filtering applied to the calculated overall error and can be increased in situations where the amount of noise in the sensor data is higher. The `err_thresh` parameter sets the threshold at which convergence is detected and should be increased in situations where higher sensor noise has reduced the level of convergence that is attainable.

5.9.2.3 Processing

Sensor calibration processing is performed either by calling the appropriate process function on an instantiation of the related parameter structure or by calling function `cal_process()` when using the wrapper.

Prior to calling the process function, it is necessary to indicate to the algorithm whether or not the sensor data is valid and has been updated by setting the `in_data_valid` flag. This is done to prevent invalid data from getting into the calibration routine and corrupting its operation. Examples of invalid data include null data while the sensor is starting up or saturated sensor data. Detection of these conditions are sensor specific, so this must be done prior to calling the calibration routine.

In the case of SmartFusion auto-calibration it is possible to speed up the rate of convergence by calling the process function multiple times between samples. In this case the `in_data_valid` flag should only be set when the sensor data is updated.

In some cases it may be desirable to disable the application of the calibration coefficients at runtime or selectively only apply offset correction. The `apply` flag enables/disables the application of the calibration coefficients altogether. The `matrix_apply` flag enables/disables the application of just the matrix coefficients. It is not possible to apply the matrix in isolation.

In some cases (for instance upon completion of one-shot mode) it may be desirable to disable the updating of the calibration coefficients by the auto calibration routine at runtime or selectively only update the offset coefficients. The `update` flag enables/disables the updating of the calibration coefficients altogether. The `matrix_update` flag enables/disables the update of just the matrix coefficients. It is not possible to update the matrix in isolation.

Once the sensor calibration routine's processing cycle is complete, the calibrated sensor data can be read from the location referenced by the `out_data` pointer. The `out_data_valid` flag can also be read to determine whether the calibration routine detected the sensor data to be within its valid range of acceptable magnitudes (as specified by `ref_mag` and `mag_range`).

So that the state of the calibration routine can be determined, flags have been provided which indicate when calibration is complete or has become unsettled. In the case of basic auto-calibration, the `settled` flag can be used and in the case of the SmartFusion auto-calibration routine the `converged` flag can be used. In one-shot mode, calibration can be stopped when the appropriate flag is set.

5.10 Sensor Fusion Library

5.10.1 Overview

The SmartFusion Sensor Fusion Library provides the SmartFusion Attitude and Heading Reference System (AHRS) for fusing sensor data from MEMS gyroscopes, accelerometers and magnetometers in order to determine and track the absolute orientation of the device in which they are mounted.

The algorithm presumes that the sensor data supplied to it has been calibrated and all distortions have been compensated for. When this not the case the performance will be compromised and drift artefacts may be observed.

5.10.1.1 Modes of operation

The SmartFusion AHRS algorithm support different modes of operation depending on what types of sensor information are available. The supported modes are as follows:

- Gyroscope, Accelerometer and Magnetometer (GAM) mode:
With information from all sensors, the algorithm is able to track the absolute orientation of the device and compensate for any drift in the gyroscope data and noise in the accelerometer and/or magnetometer data.
- Gyroscope and Accelerometer (GA) mode:
With information from only the gyroscope and the accelerometer, the algorithm is able to track the orientation of the device and compensate for any drift in the pitch and roll components of the gyroscope data (but not the heading!) and noise in the accelerometer data. The reference heading is taken to be whatever the heading is at initialization, but this will drift over time.
- Gyroscope Only (G) mode:
With information from only the gyroscope, the algorithm is able to track the orientation of the device but will be unable to compensate for any drift in the gyroscope data. The reference orientation is taken to be whatever the orientation is at initialization, but this will drift over time.
- Accelerometer and Magnetometer (AM) mode:
With information from only the accelerometer and the magnetometer, the algorithm is able to track the absolute orientation of the device, but is less able to compensate for noise in the accelerometer/magnetometer data.

Note 4 Gyroscope and Magnetometer (GM) mode is not supported.

5.10.2 API

5.10.2.1 Memory allocation

An instance of the SmartFusionAHRS_param structure must be instantiated, either statically or on the heap, and must be maintained during the lifecycle of sensor fusion processing.

5.10.2.2 Initialization

Before processing can be performed on an instance of the SmartFusion AHRS algorithm, a subset of the parameters within the SmartFusionAHRS_param structure must be initialized. These parameters are as follows:

- `controls`: Algorithm control flags.
- `g_vec_ptr`: Pointer from which to read the gyroscope input data.
- `m_vec_ptr`: Pointer from which to read the magnetometer input data.
- `a_vec_ptr`: Pointer from which to read the accelerometer input data.
- `g_scale`: Gyro scaling factor.
- `beta_a`: Scaling factor controlling the relative weight of accelerometer data.
- `beta_m`: Scaling factor controlling the relative weight of magnetometer data.
- `q`: Output quaternion representing the absolute real world orientation of the device.

The `controls` field is a union of an 8-bit word and the SmartFusionAHRS_flags bit field structure to allow the algorithm control flags to be set, individually or all at once. These flags should be initialized to indicate which sensors are available by setting the appropriate `g_vec_valid`, `a_vec_valid` and `m_vec_valid` flags to '1' or '0' to indicate the mode of operation.

The pointers to the gyroscope, accelerometer and magnetometer data vectors reference 32-bit signed integer types. However, the magnitude of the vectors that these values represent should not exceed a signed 16-bit range ($-32768 \leq |v| \leq 32767$). In case data for a particular sensor is unavailable (that is, when using the algorithm in GA, G and AM modes), the associated pointer should be set to null (0x00000000).

`g_scale`, `beta_a` and `beta_m` are all Q15 unsigned fixed point parameters representing positive scaling factors in the range of 0 to 1.0 (32768).

To aid in the setting of the `g_scale` parameter, the SmartFusionAHRS_set_gyro_scale() function has been provided. This function takes the following parameters:

- `gyro_fullscale_degrees`: The gyroscope sensitivity specified in degree/s per full-scale value (range is -32768 to +32767). For example, if set to 2000 then a gyro value of 1 corresponds to a rate of rotation of $2000 / 32768 = 0.061$ degree/s.
- `gyro_sampling_rate`: Rate at which the gyroscope is sampled in Hz.

Suitable values for `beta_a` and `beta_m` should be selected to match the requirements in terms of the maximum rate of rotation and tolerance to noise. Increasing these values will allow faster rates of rotation to be tracked at the expense of tolerance to accelerometer/magnetometer noise.

`q` must be initialized to the reference orientation [32767, 0, 0, 0] (i.e. upright and facing north).

5.10.2.3 Processing

Sensor fusion algorithm processing is performed by calling the SmartFusionAHRS_update() function on an instantiation of the SmartFusionAHRS_param structure. Prior to calling this function, the contents

of the vectors referenced by `g_vec_ptr`, `m_vec_ptr` and `a_vec_ptr` should be updated with the appropriate sensor data.

In cases where the sensors are sampled at different rates and data from individual sensors are not available at every processing cycle, the flags for these inputs should be set to '1' when they have data available and '0' when not.

The algorithm uses a right-handed coordinate system, where the x-axis is aligned with north, the y-axis is aligned with east and the z-axis is aligned with down. The gyroscope, accelerometer and magnetometer data must be converted to this coordinate space for the algorithm to function correctly.

Note 5 Positive gyroscope values represent a clockwise rotation around the associated axis (when looking in the direction it is pointing).

The algorithm also presumes that the sensors have been sampled synchronously at a regular interval and that all distortions (for example, magnetometer hard/soft iron distortions) have been properly compensated for. Algorithm performance will be degraded when this is not the case.

Once processing is complete, an updated orientation estimate can be read from `q`. This parameter represents the orientation in unit quaternion form, consisting of four elements in Q15 signed fixed point representation.

5.11 IoT MSK Android and iOS Application

5.11.1 Installing from Apple AppStore

Dialog IoT Sensors app for iOS devices is available to download from Apple AppStore.

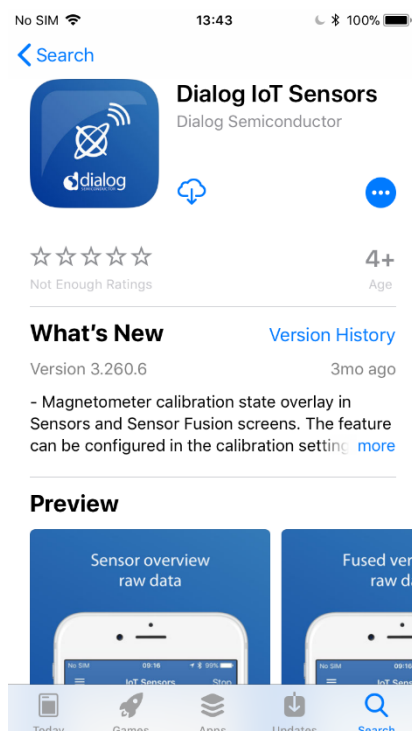


Figure 18: Installing Dialog IoT Sensors App from Apple AppStore

5.11.2 Installing from the PlayStore

Dialog IoT Sensors app for Android devices is available to download from Google PlayStore.

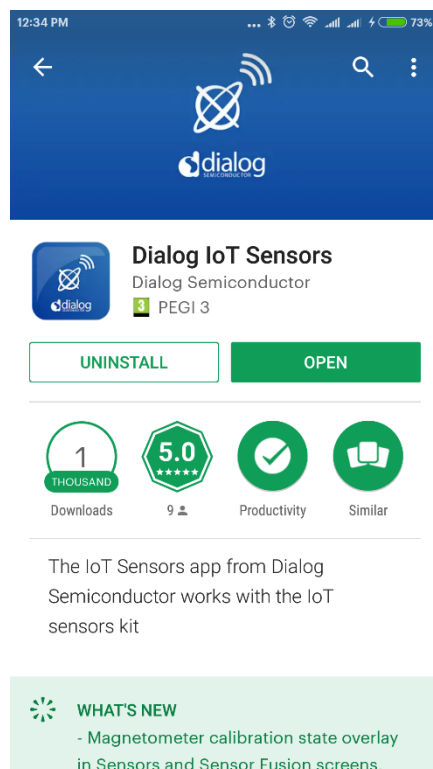


Figure 19: Installing Dialog IoT Sensors App from Google PlayStore

5.11.3 Scan Screen

Once DA14585 IoT MSK receives power, users will see the advertising LED blinking, indicating that the device is currently visible to client applications. After starting the app, the first screen users see is the "scan screen" (Figure 20).

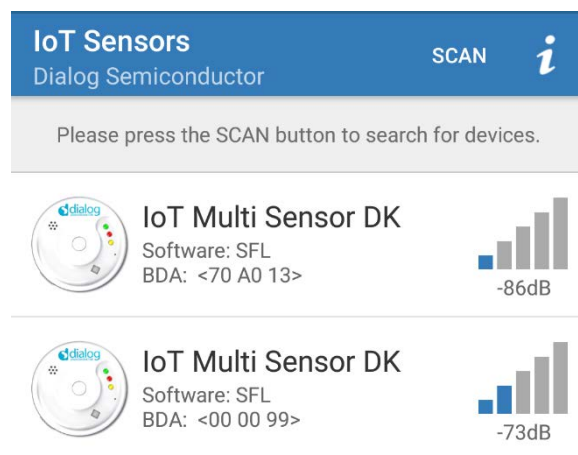


Figure 20: Scan screen of Dialog IoT Sensors App

Only IoT MSK are listed on this page. The application will hide BLE devices that do not have the advertise values related to this DK. Two fields are listed on the scan screen: the device type and the last three bytes of the BD address. Tap on a device in the list to connect to it. Note that a listed device might not be active (and the LED does not blink) when the advertise timeout has expired. To update the scan list, touch the screen to scroll downwards (iOS) or press the **Scan** button (Android).

5.11.4 Side Menu

From the side menu users can navigate to the rest of the screens that present:

- Environmental sensors
- IMU sensors
- Sensor Fusion 3D
- Cloud specific settings
- Settings
- Information
- Disclaimer
- Magnetometer status

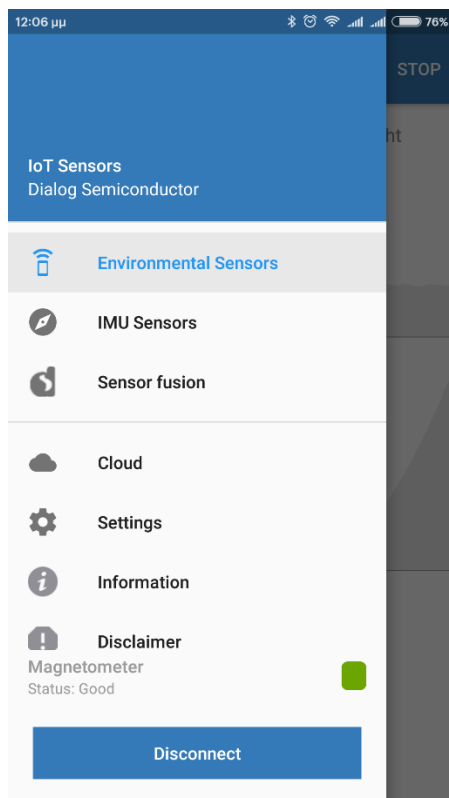


Figure 21: Side Menu

Also a "**Disconnect**" button is available to disconnect the IoT MSK device. This action will route the application to the scan screen.

5.11.4.1 Sensor Screen: Environmental Sensors

Upon successful connection, the environmental sensors screen (Figure 22) appears. This screen presents the following values:

- Temperature
- Humidity
- Pressure
- Ambient light
- Air Quality: GOOD/AVERAGE/LITTLE BAD/BAD/VERY BAD/GET OUT. The accuracy of the displayed Air quality status is also displayed. (Note 6)
- Proximity status: ON if an object is close to the surface of the sensor/ OFF if there is no object. (Note 7)
- Button status

In this screen the sensors have received automatically a "START" command and the sensors are in ACQUISITION_STATE_RUN state as described in 5.3.5. Users can stop the sensors at any time by pressing the "STOP" button on the top-right of the screen.

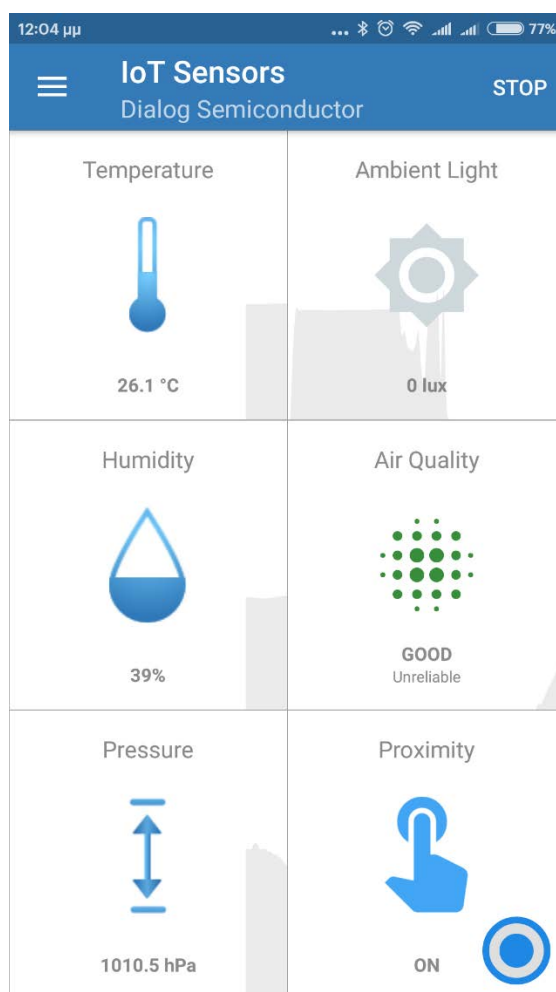


Figure 22: Environmental Sensor Screen of Dialog IoT Sensors App

Note 6 The Air quality algorithm is self-calibrated but it needs to take some values of poor air quality to reach quickly high accuracy of values. This can be achieved by enclosing the IoT Multi Sensors DK in a box and blow through a hole in it. CO2 is a big percentage of human breath.

Note 7 Proximity sensor status is affected by the top cover of the plastic enclosure of the IoT Multi Sensors DK. If the DK is located in the enclosure the values of the proximity hysteresis must be carefully selected, otherwise Proximity sensor status will be always ON. This means that in this case the distance of the object from the sensor to detect ON status will be very small.

5.11.4.2 Sensor screen: IMU sensors

This screen (Figure 23) will present raw values from the following sensors:

- Accelerometer
- Gyroscope
- Magnetometer

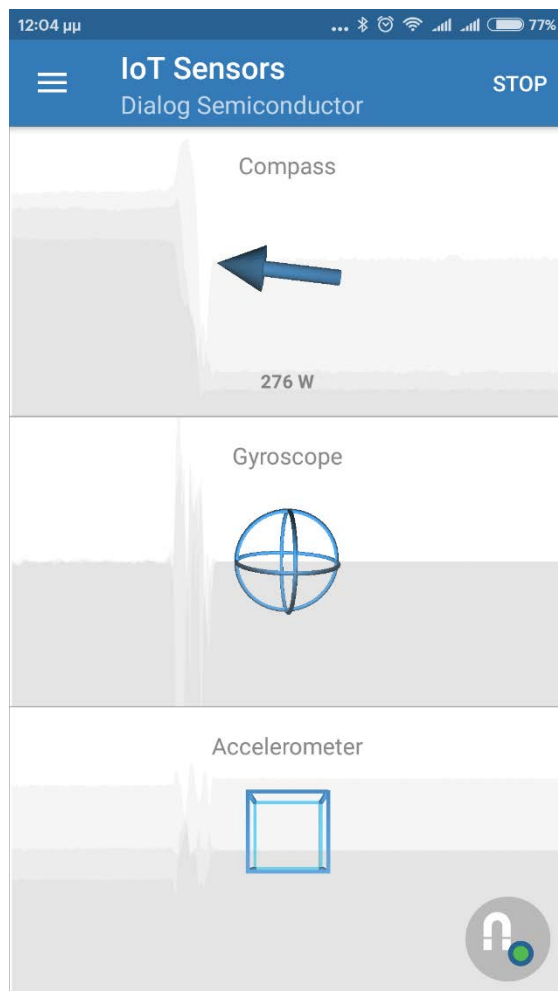


Figure 23: IMU Sensor Screen of Dialog IoT Sensors App

5.11.4.3 Sensor Fusion 3D screen

Users can navigate to the 3D screen from the Sensor screen (SFL project only). This is a 3D illustration of the position of the IoT sensor board. Notice that when we move DA14585 IoT MSK, the 3D drawing immediately adjusts itself to the new position.



Figure 24: 3D Screen of Dialog IoT Sensors App

5.11.5 Basic Settings Screen

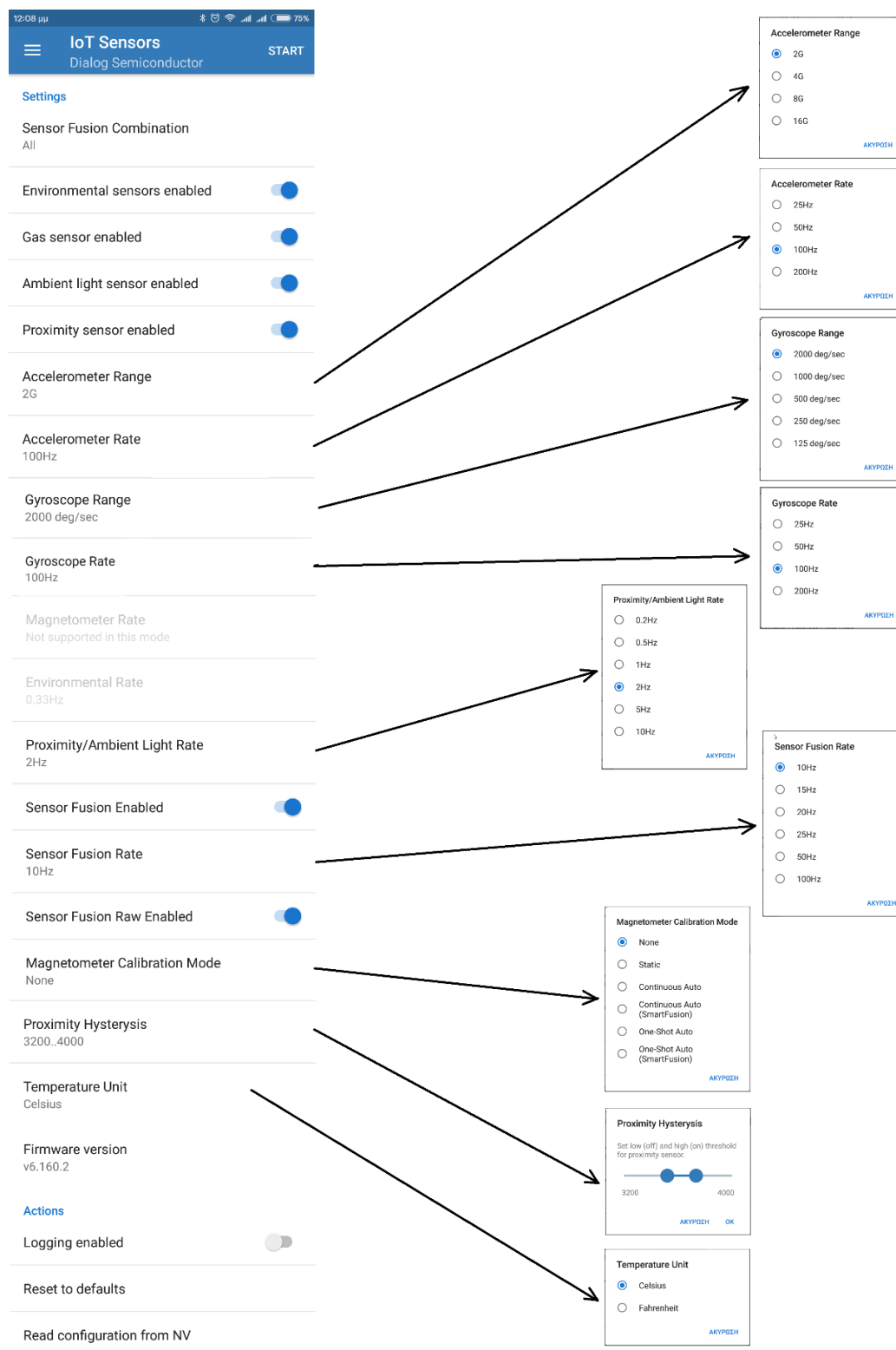


Figure 25: Basic settings of Dialog IoT Sensors App

In the basic settings screen, users can enable/disable available sensors and execute Sensor Fusion Library. Users can set the basic configuration parameters of the sensors like, range and sampling rate of accelerometer and gyro, sampling rate of Sensor Fusion, magneto, environmental and Proximity/Ambient Light sensors and hysteresis of proximity sensor state. Users can also enable/disable the magneto calibration and select the desired method or mode.

Finally the selected settings can be stored in nonvolatile memory or the configuration values can be reset to defaults.

It is important to note the following points:

- Do not change the settings while the IoT MSK is in running state. First press the STOP button.
- A modified setting will be applied after the next START command. The new setting will not be retained upon disconnection, unless the 'Store configuration to NV' button has been pressed.
- All modified settings only exist in RAM until the '**Store configuration to NV**' button is pressed. Press the '**Read Configuration from NV**' button to restore the previous settings. To restore the default settings press the '**Reset to defaults**' button.

5.11.6 Magnetometer Calibration Settings Screen

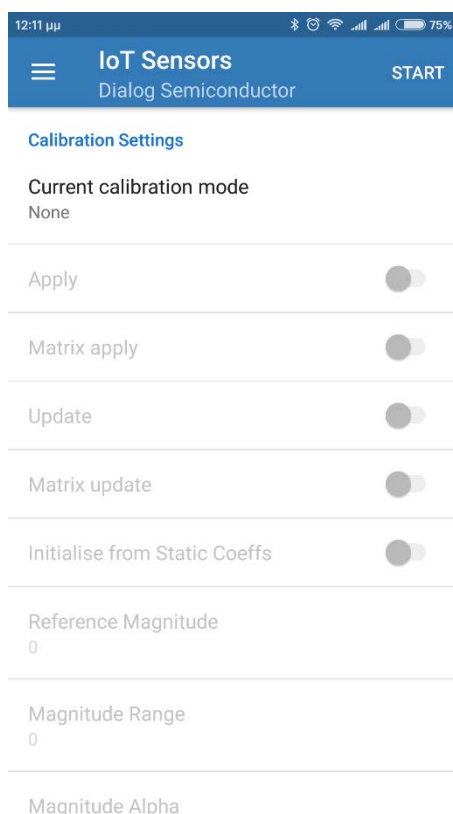


Figure 26: Calibration Settings of Dialog IoT Sensors App

The calibration page (Figure 26) contains the controls and parameters for the various calibration modes. Only the controls and parameters for the currently enabled calibration mode can be modified and those that apply to other modes are greyed out.

The controls and parameters are:

- **Apply:** Enables/Disables application of calibration coefficients on raw sensor data.

- **Matrix apply:** Enables/Disables application of calibration matrix on raw sensor data. If disabled, only offset vector is applied. This control only applies to basic auto-calibration.
- **Update:** Enables/Disables updating of calibration coefficients.
- **Matrix update:** Enables/Disables updating of calibration matrix. If disabled, only offset vector is updated. This control only applies to basic auto-calibration.
- **Initialize from Static Coeffs:** If enabled, the calibration coefficients algorithms are restored from the previous instantiation of the auto-calibration rather than being reset to default values.
- **Reference Magnitude:** The expected magnitude of the magnetometer vector.
- **Magnitude Range:** The expected range of magnitude of the magnetometer vector.
- **Magnitude Alpha:** Coefficient for the magnetometer vector magnitude filtering.
- **Magnitude Gradient Threshold:** Threshold applied to rate of change of magnetometer vector magnitude for detecting settling.
- **Offset Mu:** The rate of convergence for the offset coefficients.
- **Matrix Mu:** The rate of convergence for the matrix coefficients.
- **Error Alpha:** Coefficient for the overall error filtering.
- **Error Threshold:** Threshold applied to the filtered overall error for detecting convergence.

Note 8 The fixed point parameters are expressed in their raw integer form. For a more detailed description of the behavior of these controls and parameters, please refer to section 5.9.

5.11.6.1 Magnetometer Calibration File

It is possible to send the calibration coefficients for the currently enabled calibration mode from the application to the device. The coefficients must be loaded from file using the associated button. The coefficients can then be stored in Flash memory for subsequent initialization of the calibration algorithm. Similarly, the current set of calibration coefficients can also be saved to file.

The format of the file uses an **INI** format of the form:

```
[magnetometer calibration]
sensor_type    = 2
q_format       = 14
offset_vector  = 0, 0, 0
matrix         = 16384, 0, 0, 0, 16384, 0, 0, 0, 16384
```

All values are integers with the exception of the matrix coefficients which are fixed point values (with precision specified by `q_format`) expressed in raw integer form. The `sensor_type` should always be set to 2 (representing magnetometer).

The `offset_vector` coefficients are listed in [x, y, z] order.

The matrix coefficients are listed in [row, column] order: [0, 0], [0, 1], [0, 2], [1, 0], [1, 1], [1, 2], [2, 0], [2, 1], [2, 2].

5.11.7 SmartFusion Coefficients Screen

The SFL coefficients screen (Figure 27) contains parameters for the sensor fusion algorithm. These parameters are:

- **Beta A:** Accelerometer scaling factor
- **Beta M:** Magnetometer scaling factor

Note 9 These are fixed point parameters that are expressed in their raw integer form. For a more detailed description of the behavior of these parameters, please refer to section 5.10.

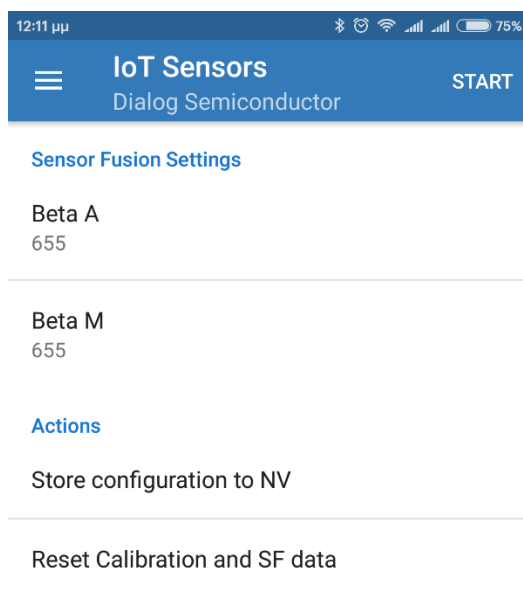


Figure 27: Sensor Fusion Settings of Dialog IoT Sensors App

5.11.8 Fast Accelerometer Calibration Screen



Figure 28: Accelerometer Calibration Settings of Dialog IoT Sensors App

Accelerometer offsets around 0 G may be observed when placing the device on a flat horizontal surface. In order to compensate for these offsets, Dialog IoT Sensors app includes a fast calibration method. This method is simple but less accurate, but it is fast and can be performed easily. Please note that the calibration should be repeated when the range of the accelerometer changes, since the offsets stored are static and not scaled on range change.

The fast calibration method includes the following steps:

1. Place the device on a stable flat surface with the LEDs facing upwards.
2. Connect the device to the IoT Sensor application.
3. Press STOP.
4. Navigate to the Fast Accelerometer Page.
5. Press Start Calibration.
6. The calibration procedure will be concluded within few seconds and the “**Calibrated**” message will appear. The calibration offsets are now saved in Flash memory and the sensor board has returned to its idle status.

Note 10 The accelerometer calibration offset will be reset when the device is returned to defaults.

6 Smart Tag Reference Application

6.1 Introduction

Dialog's Bluetooth® low energy proximity tag reference application, Smart Tag, provides an ideal starting point to develop a proximity tag application with the shortest time-to-market and lowest development cost and effort. The design comes with a complete software solution of the full proximity application and profiles source codes. Dialog also provides full-featured Android and iOS applications to manage the proximity tag's settings, trigger alerts, check the battery status, and play a fun 'Seek & Find' game, all in source code.

The Smart Tag reference application functions in the Bluetooth Low Energy Proximity Reporter Role defined in the Bluetooth specification [15]. The Proximity profile defines the behavior of a Bluetooth device when it moves away from a peer device, which covers the use case where a connection loss causes an immediate alert. This alert notifies the user that the devices have become separated.

The Smart Tag application is designed to run on Dialog's MSK HW reference design.

As a starting point, developers are suggested to get familiar with the DA14585 datasheet [5], the software developer's guide [1], and the software platform reference document [2].

6.2 Software Features

6.2.1 Profiles and Services

Besides the Proximity Reporter profile, the Smart Tag application implements the following profiles and services for monitoring and supporting additional features:

- Proximity profile, Reporter role
 - Link Loss service
 - Immediate Alert service
 - Tx Power service
- Battery Service, Server role
- Data Information service, Server role

- Find Me profile, Locator role
- SUOTA, Server role

6.2.2 Alerts

The Smart Tag application supports two types of alerts for user notifications (High level and Mild level), as described in [Table 61](#).

Table 61: Alert Types

Alert type	Description
High level	• LED blinking with buzzer tone with the following pattern: ○ LED: 150 ms on, 150 ms off. ○ Buzzer: 150 ms on, 150 ms off, alternating between 392 Hz ("G" note) and 440 Hz ("A" note) • Triggered when peer device writes immediate alert with 'High Alert', or Smart Tag disconnects from peer and Link Loss is set to 'High Alert'.
Mild level	• LED blinking with buzzer tone with the following pattern: ○ LED: 500 ms on, 500 ms off ○ Buzzer: 500 ms on, 500 ms off, 440 Hz ("A" note) • Triggered when peer device writes immediate alert with 'Mild Alert', or Smart Tag disconnects from peer and Link Loss is set to 'Mild Alert'.

6.2.3 Advertising and Sleep Phases

Smart Tag advertises in undirected mode with different intervals for specific advertising phases:

- Advertising phase (200 ms interval): For the first minute after start-up or disconnection.
- Advertising phase (1000 ms interval): For three minutes following the 200 ms interval phase.
- Deep Sleep phase: After four minutes the Smart Tag stops advertising and enters continuous Deep Sleep mode.

In Advertising mode the Smart Tag blinks the green LED with the following pattern: 50 ms on - 1000 ms off. On connection the LED stops blinking, while on disconnection the LED starts blinking again since it goes again in advertising mode.

The Smart Tag application supports Extended Sleep mode during Connectable and Connected states, and Deep Sleep mode during the Deep Sleep Phase. The supported sleep modes of the DA14585 are explained in [section 8.6](#) in [\[1\]](#).

6.2.4 Push-Button Interface

The actions that are triggered on a push-button event depend on the operating state of the Smart Tag as described in [Table 62](#):

Table 62: Push-Button Interface

Operating State	Action on Button Press
Alert is active	Stop alert
Deep Sleep phase	Wake up Smart Tag and start advertising

Operating State	Action on Button Press
Connected and 'find me' locator discovered, immediate alert service on peer device	If the alert is not active, writes alert characteristic of immediate alert service on peer device. Stops the alert in peer device, if alert is active
Advertising phase	Long press (currently set to 3 s): bonding data are deleted from SPI Flash memory. When long press is detected, an 880 Hz tone ('A' note, 5th octave) will be played for 125 ms (Note 11)
All other states	None

Note 11 When deleting the bonding data from the Smart Tag SPI Flash and the Android device is paired with the Smart Tag device, the Smart Tag device needs to be removed from the list of paired devices of the Android device (usually via menu **Settings > Bluetooth > Forget Device**).

6.2.5 Security

According to the Bluetooth Core specification, the purpose of bonding is to create a relation between two Bluetooth devices based on a common link key (a bond). The link key is created and exchanged (pairing) during the bonding procedure and is expected to be stored by both Bluetooth devices to be used for future authentication.

Since the MSK reference design does not have a keyboard or display, it only supports the 'Just Works' pairing method. Upon completion of a successful bonding procedure, the Smart Tag application stores the security information (LTK, EDIV, and RAND which will be also referred as 'bonding data') in the SPI Flash memory for reuse on subsequent reconnections of the bonded device. For more information regarding the bonding procedure refer to [\[3\]](#), section 5.6.

When the Smart Tag is in Advertising mode, users can 'forget' a bonded central device by keeping the button pressed until a tone is heard. This indicates that the security information has been deleted from the SPI Flash memory and a new central device can now pair with the Smart Tag device.

Note 12 The Smart Tag transmits a Security Request command to the central device in order to trigger the pairing procedure, upon receiving the connection request from the central device. However, this command could be ignored by the central device, when it does not wish to start a pairing procedure.

6.2.6 Battery Level

In Connected state, the Smart Tag software samples the battery level and updates the value of the battery level characteristic. The notification capability of the characteristic is disabled at the beginning of each connection. To enable value update notifications, the peer device must write the configuration attribute of the characteristic with the corresponding value.

6.3 Software Architecture

Group	File Name	Description
user_platform	i2c_gpio_extender.c	User drivers for the I2C GPIO extender
user_app	user_smarttag_proj.c user_smarttag_utils.c	Application code
utilities	user_iot_dk_utils.c battery.c	User drivers for battery and MSK HW peripherals

6.4 Operation Overview and State Machines

This section provides information about important functions of the application and a detailed description of the Finite State Machines (FSM) that are used.

6.4.1 Application Configuration Parameters

The main parameters of the Smart Tag application software, which can be adjusted to customize certain functionality of the application, are listed in [Table 63](#).

Table 63: Smart Tag Application Configurable Parameters

Parameter	Description	Current Value
APP_SPI_POWEROFF_DELAY	Upon application initialization, the timer APP_FLASH_POWEROFF_TIMER is set with this value to delay the SPI Flash power down mode and allow the developer to use the SmartSnippets Flash Programmer application to connect to the Smart Tag device and re-program the SPI Flash device.	1 s
APP_SLEEP_DELAY	Upon application initialization, the function app_set_startup_sleep_delay(APP_SLEEP_DELAY) is called to modify the system startup sleep delay. This delay allows the developer to have an active JTAG interface and to use the debugger to connect to the Smart Tag device.	5 s
APP_FIRST_ADV_PHASE_DUR	This parameter sets the APP_ADV_TIMER to control the advertising intervals. Refer to section 6.2.3 for details.	60 s
APP_FIST_ADV_PHASE_INTVAL	This parameter sets the advertising interval for the first advertising phase.	200 ms
APP_SECOND_ADV_PHASE_DUR	This parameter sets the APP_ADV_TIMER to control the advertising intervals. Refer to section 6.2.3 for details.	3 min
APP_SECOND_ADV_PHASE_INTVAL	This parameter sets the advertising interval for the second advertising phase.	1 s
APP_BOND_DB_DATA_OFFSET	The SPI Flash start address where the bonding data are stored.	0x32000
APP_ADV_BLINK_ON_DUR	Controls the LED 'on' duration during advertising.	50 ms
APP_ADV_BLINK_OFF_DUR	Controls the LED 'off' duration during advertising.	1 s

6.4.2 Application Task State Machine

The FSM of the application task of Smart Tag consists of the following states ([Table 64](#)):

Table 64: Application Task: FSM States

State	Description
APP_DISABLED	Application task initiated. Waiting for GAPM_DEVICE_READY_IND message
APP_DB_INIT	Database initialization in progress
APP_CONNECTABLE	Advertising or Continuous Extended Sleep
APP_CONNECTED	Device connected to Proximity Monitor

[Figure 29](#) graphically illustrates the FSM. The state transitions are described in [Table 65](#).

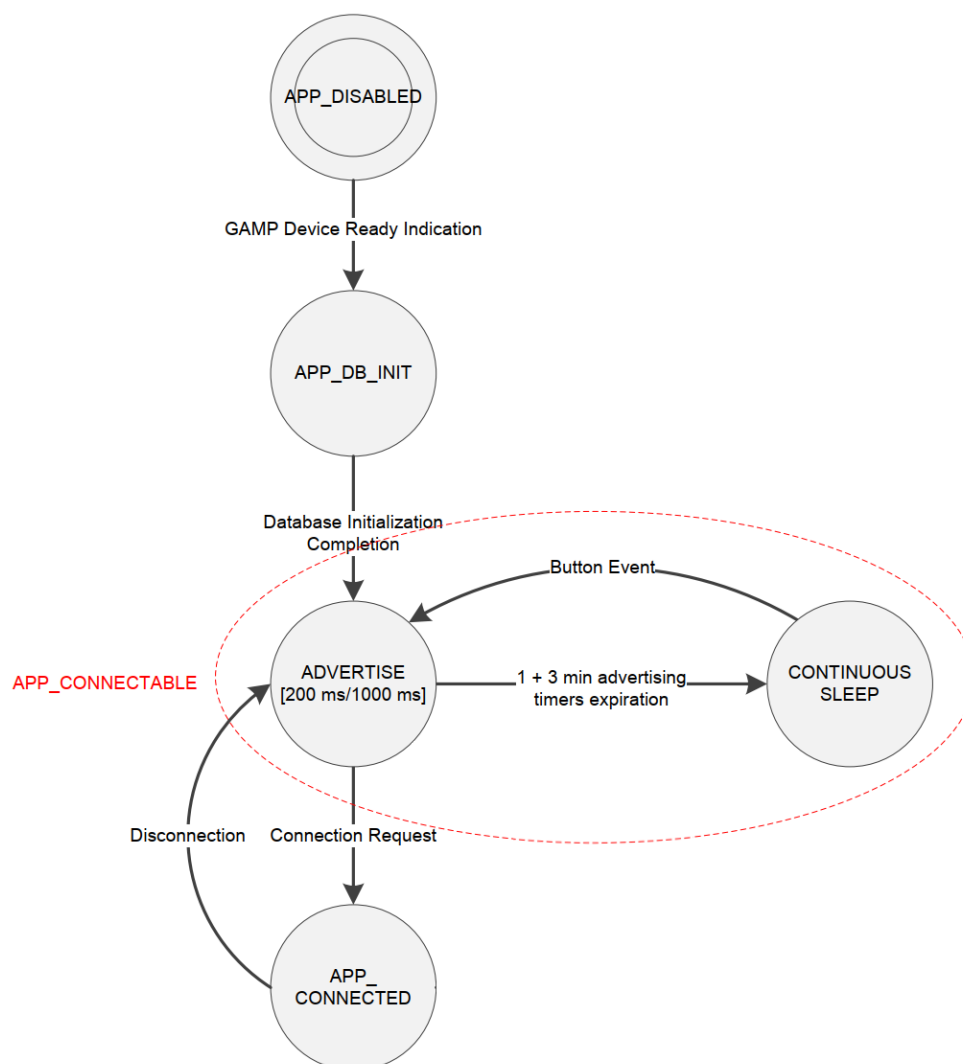


Figure 29: Smart Tag Application Task FSM

Table 65: State Transitions of the Application Task FSM

State Transition		Event
From	To	Action
APP_DISABLED		GAMP_DEVICE_READY_IND message reception.
	APP_DB_INIT	Start database initialization of supported profiles.
APP_DB_INIT		Database initialization procedure is completed.
	ADVERTISE (APP_CONNECTABLE)	Start advertising in 200 ms advertising interval.
ADVERTISE (APP_CONNECTABLE)		Timer for 1000 ms advertising interval expired.
	CONTINUOUS_SLEEP (APP_CONNECTABLE)	Device stops advertising and/or active alerts and switches to Deep Sleep mode.

State Transition		Event
From	To	Action
CONTINUOUS_SLEEP (APP_CONNECTABLE)		Button press event.
	ADVERTISE (APP_CONNECTABLE)	Device exits Deep Sleep mode and restarts advertising in 200 ms interval.
ADVERTISE (APP_CONNECTABLE)		Connection Request has been received.
	APP_CONNECTED	Enable profiles and start battery polling.
APP_CONNECTED		Device disconnects.
	ADVERTISE (APP_CONNECTABLE)	Start advertising in 200 ms advertising interval.

6.4.3 Callback Functions

A set of callback functions is defined in `user_callback_config.h` which consist of the entry points of the application.

- `user_app_on_init()`:
 - The function `user_app_on_init()` is the main entry point of the application task.
 - It is used to initialize the parameters of high level profiles and low level hardware modules.
- `user_app_on_set_dev_config_complete()`:
 - The function `user_app_on_set_dev_config_complete()` is called after the device configuration is complete.
 - This means that the device database of the supported profiles has been created and the device can enter Advertising mode.
 - Also, in this function the Flash power-off timer is initialized. This timer allows the user to use the [SmartSnippets](#) tool to program the Flash memory by leaving the Flash memory in power-up mode for $(APP_SPI_POWEROFF_DELAY * 10)$ ms. The default time is 1 s.

6.4.4 Advertising

- `user_advertise_operation()`
 - The function `user_advertise_operation()` starts or restarts the Advertising mode.
 - It is called upon completion of the database initialization, upon device disconnection, and upon expiration of the advertising timer to start the advertising interval of the second phase.
 - The function also initializes the advertising timer for the variable advertising interval feature and the blink timer for Advertising mode LED blinking.
 - Finally, function `user_undirected_advertise_start()` constructs and sends the command `GAPM_START_ADVERTISE_CMD` to the `GAPM_TASK` in order to initiate the advertising mode.
- `user_adv_timer_handler()`
 - The function `user_adv_timer_handler()` is called upon expiration of the advertising timer.
 - Its main functionality is to program the correct advertising phase.
 - For Continuous Sleep state, the function disables all timers and alerts before setting the device in Deep Sleep mode.

- `user_blink_timer_handler()`
 - The function `user_blink_timer_handler()` handles the expiration of the LED blinking timer, restarts the timer, and inverts the state of the LED.

6.4.5 Connection

- `user_app_on_connection()`
 - The function `user_app_on_connection()` is called upon reception of a connection request from a central role device.
 - When this function is called, the Smart Tag application stops the LED blinking timer, enables the profiles and services, selects the proper sleep mode, and enables the device profiles.
- `user_on_disconnect()`
 - The function `user_on_disconnect()` is called upon reception of a `GAPC_DISCONNECT_IND` message, which indicates that the connection does not exist anymore.
 - When this function is called, the battery level polling is stopped and the advertising procedure starts again.

6.4.6 Security

- `user_app_on_pairing_request()`
 - The function `user_app_on_pairing_request()` is called during the pairing process.
 - It informs the peer device about the security capabilities of the device. The Smart Tag application uses 'Just Works' mode with bonding capability.
 - It also checks whether the Smart Tag device is already paired with another device by looking for bonding data stored in SPI Flash memory. When the Smart Tag device is already paired with another central device, it will not accept the new pairing request. Smart Tag only supports bonding with one central device at a time.
- `default_app_on_ltk_exch()`
 - The function `default_app_on_ltk_exch()` is the SDK 5 default function and is called upon reception of a `GAPC_BOND_REQ_IND` message with request set to `GAPC_LTK_EXCH` (Long Term Key Exchange).
 - This function generates the LTK and sends it to the host.
- `user_app_on_pairing_succeed()`
 - The function `user_app_on_pairing_succeed()` is called upon reception of a `GAPC_BOND_IND` message with status `GAPC_PAIRING_SUCCEED`.
 - The function stores the security information into SPI Flash memory and completes the connection establishment phase.
- `user_app_on_encrypt_req_ind()`
 - The function `user_app_on_encrypt_req_ind()` is called to initiate a secure connection upon the reception of a `GAPC_ENCRYPT_REQ_IND` message.
 - In order to validate the connecting host, this function uses the parameters `RAND` and `EDIV` to check whether the Smart Tag has already stored the bonding data in SPI Flash memory. If not, the request is rejected and the peer is disconnected.
- `user_app_on_encrypt_ind()`
 - The function `user_app_on_encrypt_ind()` is called upon the reception of a `GAPC_ENCRYPT_IND` message to indicate that encryption is completed.

- The database is updated with values from the SPI Flash memory.

6.4.7 Push button

In the application initialization function `user_app_on_init()`, a wakeup interrupt (IRQ) is enabled on the GPIO that is allocated to the push-button interface. This is done via the API functions `wkupct_register_callback()` and `wkupct_enable_irq()` of the wakeup module driver. The callback function `user_button_press_cb()` is registered to enable a wakeup interrupt when the button is pressed.

- `user_button_press_cb()`
 - The function `user_button_press_cb()` is the callback function of the application, called from the `WKUP_QUADEC_IRQn()` interrupt handler of the wakeup module driver when the button is pressed.
 - The function checks the state of the application and triggers the required action, as described in [Table 62](#).
 - It also calls the API functions `wkupct_register_callback()` and `wkupct_enable_irq()` of the wakeup module driver, to register the `user_button_release_cb()` callback function and enable a wakeup interrupt when the button is released.
 - Finally, the function sends a wakeup message to the `TASK_APP` to start the button press timer, which is used to detect a long key press for deleting the bonding data stored in the SPI Flash memory.
- `user_button_release_cb()`
 - The function `user_button_release_cb()` is the callback function of the application, called from the `WKUP_QUADEC_IRQn()` interrupt handler of the wakeup module driver when the button is released.
 - The function calls the API functions `wkupct_register_callback()` and `wkupct_enable_irq()` of the wakeup module driver, to register the `app_button_press_cb()` callback function and enable a wakeup interrupt when the button is pressed.
 - Finally, the function sends a wakeup message to the `TASK_APP` to stop the button press timer.
- `user_wakeup_handler()`
 - The function `user_wakeup_handler()` is called upon reception of the wakeup message and calls function `user_advertise_operation()` to start advertising.
 - It also starts/stops the button press timer, depending on the button status (`user_button_status`).

6.4.8 Proximity Reporter and Alerts

- `app_proxr_enable()`
 - The function `app_proxr_enable()` enables the Proximity Reporter profile upon connection.
- `user_proxr_alert_ind_handler()`
 - The function `user_proxr_alert_ind_handler()` is the message handler of a `PROXR_LLS_ALERT_IND` message, which is sent by the Proximity Reporter profile to trigger an alert on the device.
 - This function calls functions `user_proxr_alert_start()` or `user_proxr_alert_stop()` to start or stop an alert, depending on the alert level received in `PROXR_ALERT_IND`.
- `user_proxr_alert_start()`

- The function `user_proxr_alert_start()` initiates user alert indications. It updates the alert state parameters and depending on the alert level it starts the PWM engine by calling the function `user_proxr_pwm_enable()` to generate the alert melody.

Note 13 The LED functionality is controlled from within the functions that program the PWM tones, as explained in section 6.4.9.

- `user_proxr_alert_stop()`
 - The function `user_proxr_alert_stop()` stops user alert indications. It clears the alert state parameters, turns off the alert LED, and stops the APP_PXP_TIMER.

6.4.9 PWM Engine

This section describes how the Smart Tag uses the PWM0 and PWM1 (TIMER 0) outputs to create the alert melodies. Details on the PWM0 and PWM1 can be found in section 7.12 in [1].

- `user_proxr_pwm_enable()`
The function `user_proxr_pwm_enable()` initializes TIMER 0:
 - Enables the TIMER 0 peripheral clock by calling the `set_tmr_enable()` PWM API driver function.
 - Calls `set_tmr_div()` to sets the TIMER 0 clock division factor to 8 (16 MHz clock source).
 - Calls `timer0_init()` to initialize the PWM with the desired PWM mode, TIMER 0 'on' time division option, and clock source selection.
 - In this example, the timer tick period is configured to:
 $(1/16 \text{ MHz}) \times 8 \text{ (clock division)} \times 10 \text{ (TIM0_CLK_DIV_BY_10)} = 5 \mu\text{s}$
 - Sets the TIMER 0 'on', 'high', and 'low' times by calling function `timer0_set()`.
 - Registers a callback function for SWTIM_IRQn interrupts by calling `timer0_register_callback()`. The callback function pointer is an input parameter to this function. In the Smart Tag application three different melodies/tones are needed, which are handled by the following callback functions:
 - `high_alert_pwm_callback()`: programs the high alert melody.
 - `mild_alert_pwm_callback()`: programs the mild alert melody.
 - `button_pwm_callback()`: programs the button long press tone.
 - Enables the SWTIM_IRQn, by calling the `timer0_enable_irq()` function.
 - Starts TIMER 0 by calling the `timer0_start()` function.
- `high_alert_pwm_callback()`
The function `high_alert_pwm_callback()` is a callback function for the SWTIM_IRQn interrupt that handles the high alert melody. The following parameters configure the alert melody:
 - The melody is defined in the constant array `alert_high_notes[]`.
 - In this array developers can define a new sequence of notes and pauses. Note values represent the frequency of the musical notes. For example, '880' means 880 Hz which is the 'A' note of the 5th octave.
 - Each time this callback function is called, a note or a pause from the array will be programmed to the PWM engine by calling the functions `timer0_set_pwm_high_counter()` and `timer0_set_pwm_low_counter()`.
 - The duration of the note/pause is determined by the parameter value `ALERT_HIGH_DURATION` that is passed to the function `timer0_set_pwm_on_counter()`. At the end of this duration, an

interrupt will be triggered and the callback function will be executed again to program the next note/pause from the array.

- In order for the LED to blink synchronously with the melody, the LED is controlled within `high_alert_pwm_callback()`. When a pause is programmed, the LED is turned off. When a note is played, the LED is turned on.

- `mild_alert_pwm_callback()`

The function `mild_alert_pwm_callback()` is a callback function for the `SWTIM_IRQn` interrupt that handles the mild alert melody. This function is similar to the `high_alert_pwm_callback()` function, described in the previous section, with one extra parameter:

- `ALERT_MILD_EXTRA_DELAY`: This parameter determines how many times the function will be called without programming a note/pause. This is needed in case that a note/pause duration needs to be quite long and cannot be set by the `timer0_set_pwm_on_counter()` range.
- The notes/pauses for the mild alert melody are defined in the constant array `alert_mild_notes[]`. The duration of the note/pause is determined by the parameter value `ALERT_MILD_DURATION`.

- `button_pwm_callback()`

The function `button_pwm_callback()` is a callback function for the `SWTIM_IRQn` interrupt that handles the long button press alert. This alert is a single tone alert (`button_press_notes = PWM_TONE_A_5TH`) and the `TIMER 0` 'on' counter is not programmed again, because no further interrupts are needed to program more notes/pauses.

6.4.10 SmartTag Sequence Diagram

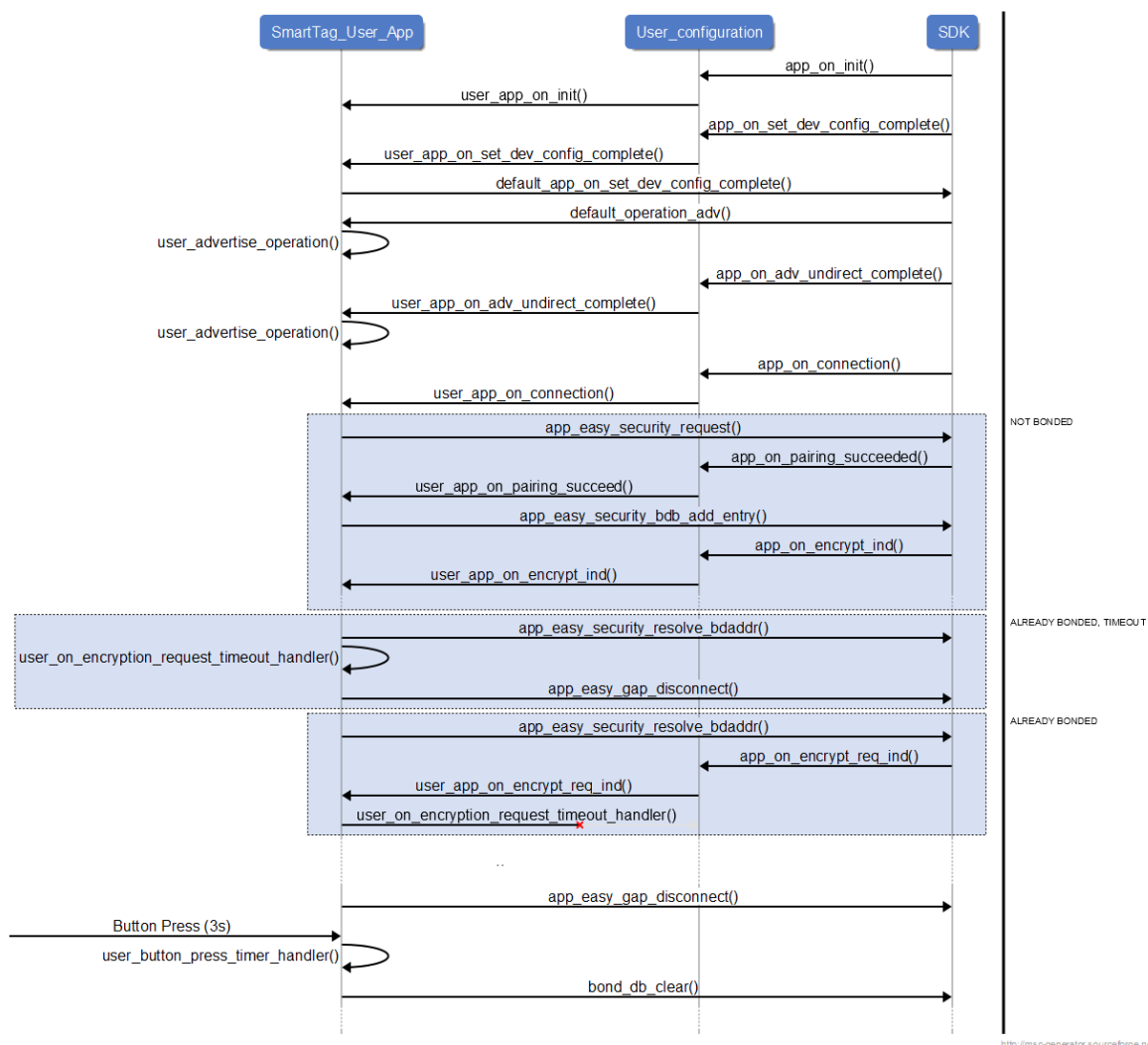


Figure 30: Smart Tag Reference Application Sequence Diagram

6.5 Application of SmartTags in Android and iOS

6.5.1 Overview

The Smart Tag proximity reference application comes with a proximity monitoring demo app called SmartTags for mobile devices using iOS and Android. The Dialog SmartTags app can discover, connect, and control devices with a Smart Tag within the Bluetooth RF range of the iOS and Android devices.

The features of SmartTags include:

- Proximity reporter devices discovery
- Connection to discovered devices
- RSSI polling of connected devices
- Immediate alert

- Distance alert based on measured RSSI value
- Link Loss alert
- Battery level monitoring
- Disconnection
- Seek and Find game based on the current RSSI value

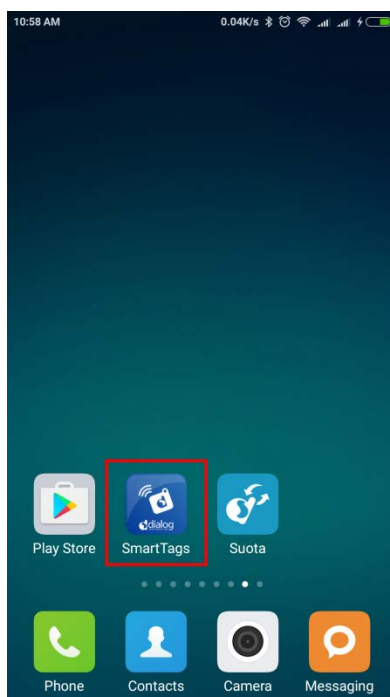


Figure 31: SmartTags APP Icon

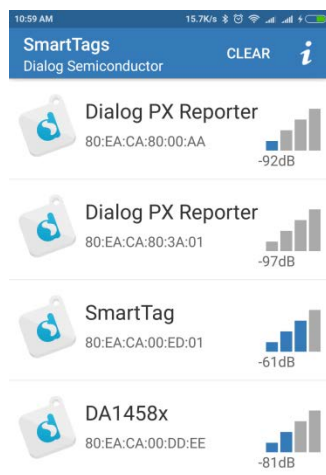
6.5.2 Installation

The Dialog SmartTags app is available in Apple App Store and Google Play Store. Search for "Dialog SmartTags" and click the install icon (Figure 32). The application will then be downloaded and installed without further user interaction.


Figure 32: Installation of Dialog SmartTags App

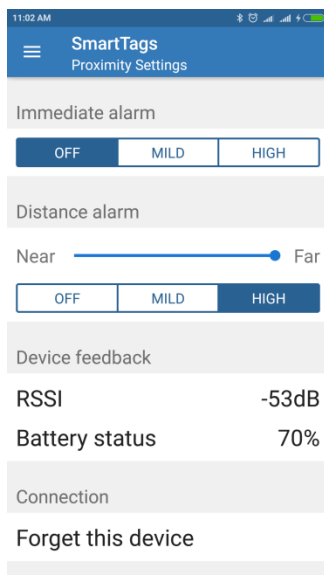
6.5.3 Device List

Tap on the Dialog SmartTags icon to start the application. A list of all devices supporting the Proximity Reporter profile, including Smart Tag, will be shown on the screen. A device with a smart tag will get displayed automatically when it gets within the Bluetooth RF range of the iOS/Android device. Tab on a device icon in the list to connect to it. If a device is not paired (section 6.2.5), users will be asked by a request to pair the device. If a device is already paired, it will be marked as "Connected" in the list.


Figure 33: Device List

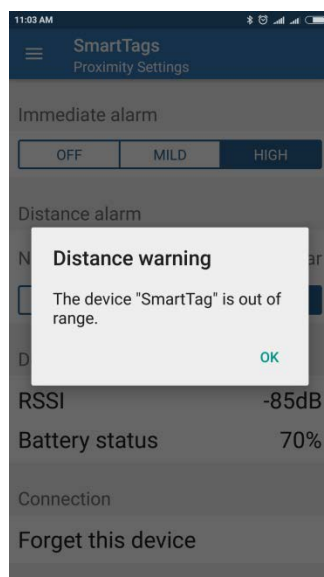
6.5.4 Device Details

The following controls (Figure 34) are provided for monitoring and control of the device:

**Figure 34: Device Details**

- Immediate Alarm

Click on the **MILD** or **HIGH** button of the "**Immediate alarm**" section to trigger an immediate alert. The alert can be terminated by clicking on the **OFF** button.
- Distance Alarm
 - Distance and link loss alerts are configured by the "**Distance alarm**" section controls. The SmartTags application polls the RSSI of the connected devices to estimate the distance between the iOS/Android system and the Smart Tag device.
 - The **Near/Far** slider controls the RSSI threshold for triggering a distance alert. When the measured RSSI level of a connected device moves below the configured threshold or a device is disconnected, the SmartTags app will trigger an alert on both Smart Tag and iOS or Android devices. In case of an alert, the screen displays a message as shown in [Figure 35](#). Click **OK** to terminate the alert on either device.
 - The alert level depends on the selected button (**OFF**, **MILD**, and **HIGH**). When the **OFF** button is selected, no alert will be triggered.

**Figure 35: Distance Alert Window**

- Device Feedback
 - The "**Device feedback**" section provides status information for the device connected to the SmartTags proximity reporter, showing the current RSSI value and the battery level of the device with a Smart Tag.
- Connection
 - Users can disconnect a device from the Dialog SmartTags app by clicking on **Forget this Device**. Please note that this control does not erase the bonding data. It just disables auto connection of the specific IoT MSK device.
- Side Menu
- The side menu appear when the top left menu icon is clicked ([Figure 36](#)). From the side menu users have access to the Seek & Find game, the information of the device ([Figure 37](#)), and the disclaimer statement ([Figure 38](#)).

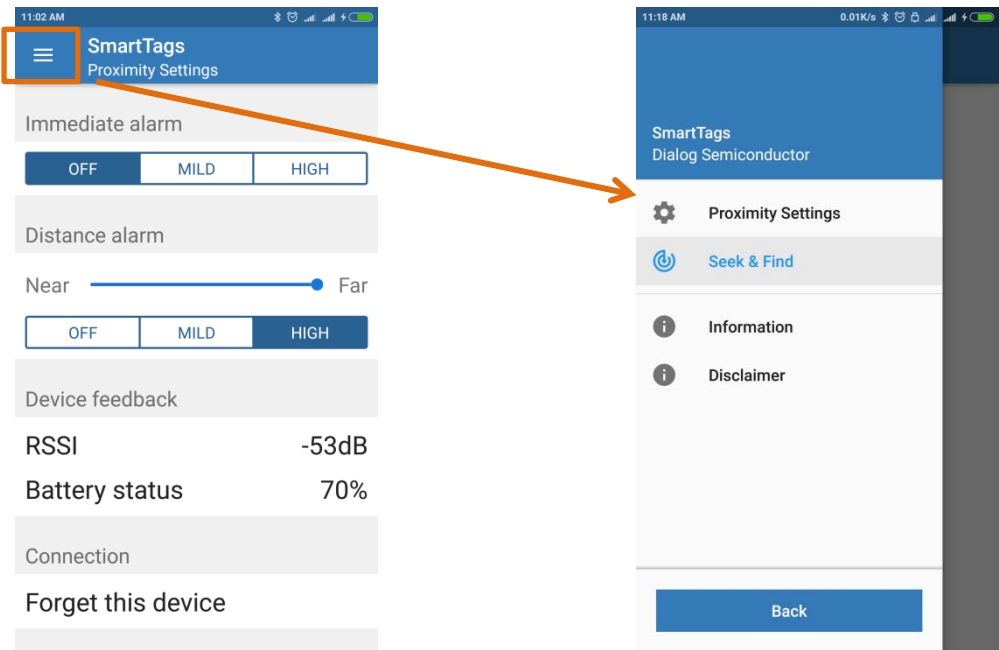


Figure 36: How to Access the Side Menu

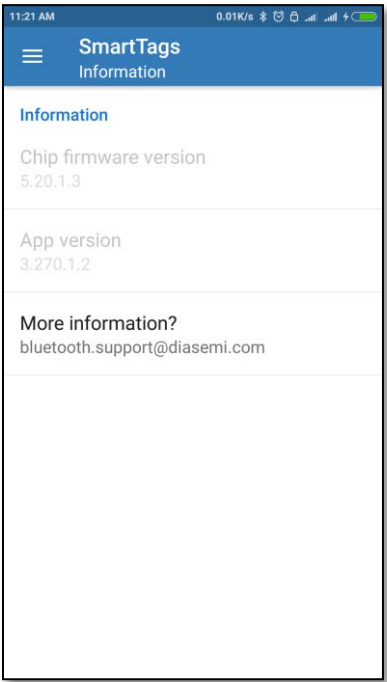
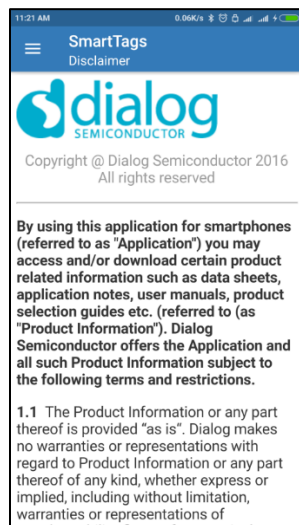


Figure 37: Device Information

**Figure 38: Disclaimer**

- Seek & Find Game
 - By clicking the Seek & Find icon (
 - [Figure 36](#)), the Dialog SmartTags app will start a Warmer/Colder game with the selected device.
 - The application changes states and displays an image according to the RSSI-based estimate of the distance of a device with a Smart Tag. The app changes to a warmer state when it approaches a device with a Smart Tag, while it changes to a colder state when it becomes further from a device with a Smart Tag. The various “Seek & Find” screens are shown in [Figure 39](#).

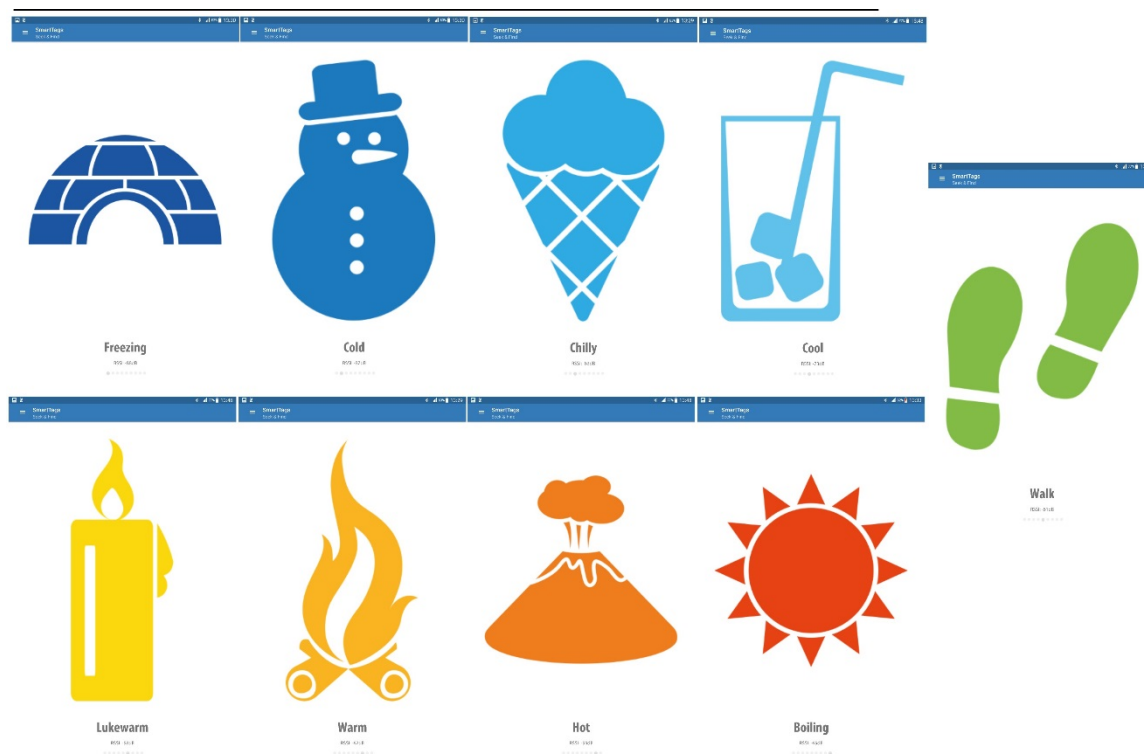


Figure 39: Seek & Find screens

7 Beacon Reference Applications

7.1 Introduction

This section describes the Bluetooth® Low Energy Beacon reference application design based on DA14585. It serves as a user guide for developers to customize the beacon for any desired purpose.

Dialog Semiconductor is a member of the iBeacon™ program. Please contact your local sales office to learn more about the possibilities we can offer relating to this standard. The Dialog Beacon reference application also supports the AltBeacon protocol as well as the Eddystone™ protocol, with supported modes being the Eddystone -UID, the Eddystone -URL, and the Eddystone -TLM (unencrypted).



Figure 40: Beacon Protocol Logos

The Beacon reference software can be downloaded from the [Dialog support portal](#) and run on the MSK reference hardware. The Beacon software also runs on the Dialog DA14585/14586 Development kit (Expert/Pro/Basic).

7.2 What is a Beacon?

Beacons are battery powered devices that advertise a particular Bluetooth low energy payload with identifying information. In short, it is a device that just says (Figure 41):



Figure 41: Bluetooth Low Energy Beacon

Although section 4 contains a startup guide that explains how to run the Beacon reference software out of the box and how to configure the main parameters, this section provides to software developers all design details to customize the Dialog Beacon reference application for more advanced use cases, such as:

- Adaptive modification of advertising data
- Choosing from various beacon formats
- Interleaving connectable advertising events
- Software Updates Over The Air (SUOTA)

7.3 Beacon Example

Table 66 shows a beacon configuration that is chosen for an international museum application, where each exhibit has its own unique beacon. The specific advertising data transmitted by each beacon is picked up by mobile phones in proximity of the exhibit. The mobile phone will then direct users to additional information regarding the exhibit.

In this museum example, the organization uses a Universal Unique Identifier (UUID). The museum location is indicated by the Major Field and the exhibit number within the museum by the Minor field.

Table 66: Example of Advertising Data from a Museum Beacon

Museum Location		London	Paris	Amsterdam
UUID		1234A567-3854-ABED-8FAC-56E783159AE2		
Major		10	20	30
Minor	Exhibit #1	1	1	1
	Exhibit #2	2	2	2
	Exhibit #3	3	3	3

When the beacon sends '20' as a major value and '1' as a minor value, the application on the smartphone/tablet will guide visitors to additional information on exhibit #1 from the Paris museum. This additional information might come from the smartphone application or from the internet. As a result, visitors will receive only specific information that is of interest when standing close to the exhibit (Figure 42).



Figure 42: Description of the Exhibit on a Smartphone

The smartphone application can also provide a distance indication to the beacon using the RSSI value.

There are endless other applications using Beacon apart from the museum application, for example, in retail, advertising, and sports events.

7.4 Beacon Formats

There are various beacon formats. The beacon formats supported by DA14585 IoT MSK are presented.

7.4.1 iBeacon

iBeacon™ is a protocol developed by Apple but used by various vendors. It is a closed format that exposes a universally unique identifier (UUID) as well as other configurable values. An iBeacon frame is presented in Figure 43.

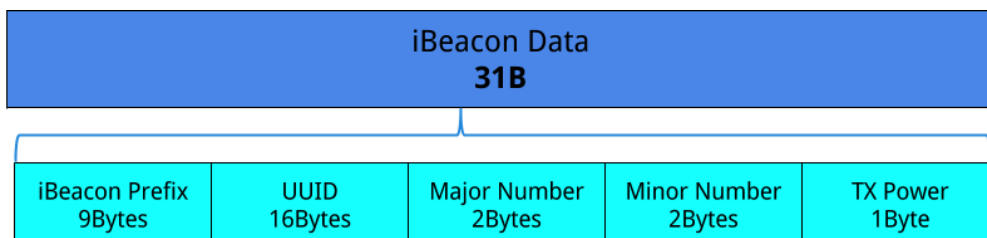


Figure 43: iBeacon Frame

An iBeacon frame consists of:

- **iBeacon Prefix:** A 9-byte constant preamble identifying iBeacon
- **UUID:** A 16-byte string used to differentiate a large group of related beacons.
- **Major:** A 2-byte string meant to distinguish a smaller subset of beacons within the larger group.
- **Minor:** A 2-byte string meant to identify individual beacons.
- **Tx Power:** A 1-byte value representing the RSSI at 1 m from the advertiser.

7.4.2 AltBeacon

AltBeacon is an open beacon format for proximity beacons. The emitted message contains information that the receiving device can use to identify the beacon and to compute its relative distance to the beacon. The receiving device may use this information as a contextual trigger to

execute procedures and implement behaviors that are relevant to being in proximity to the transmitting beacon (see [13]). Figure 44 presents the AltBeacon frame structure.



Figure 44: AltBeacon Frame

Table 67 provides more detailed information on the various field of an AltBeacon frame.

Table 67: AltBeacon Protocol Fields

Field Name	Description	Accepted Values
AD LENGTH [MFG SPECIFIC]	Length of the type and data portion of the Manufacturer Specific advertising data structure.	0x1B
AD TYPE [MFG SPECIFIC]	Type representing the Manufacturer Specific advertising data structure.	0xFF
MFG ID	The beacon device manufacturer's company identifier code.	The little endian representation of the beacon device manufacturer's company code as maintained by the Bluetooth SIG assigned numbers database.
BEACON CODE	The AltBeacon advertisement code.	The big endian representation of the value 0xBEAC.
BEACON ID	A 20-byte value uniquely identifying the beacon.	The big endian representation of the beacon identifier. For interoperability purposes, the first 16 bytes of the beacon identifier should be unique to the advertiser's organizational unit. Any remaining bytes of the beacon identifier may be subdivided as needed for the use case. Note 14
REFERENCE RSSI	A 1-byte value representing the average received signal strength at 1m from the advertiser.	A signed 1-byte value from 0 to -127.
MFG RESERVED	Reserved for use by the manufacturer to implement special features.	A 1-byte value from 0x00 to 0xFF. Interpretation of this value is to be defined by the manufacturer and is to be evaluated based on the MFG ID value.

Note 14 In Dialog beacon reference design, the Beacon ID is divided into a 16-byte UUID, a 2-byte ALT_val1, and a 2-byte ALT_val2.

7.4.3 Eddystone

Eddystone™ is a protocol specification that defines a BLE message format for proximity beacon messages. It describes several different frame types that may be used individually or in combination to create beacons suitable for a variety of applications (see [9]). Figure 45 shows the Eddystone modes supported by the Dialog Beacon reference design.

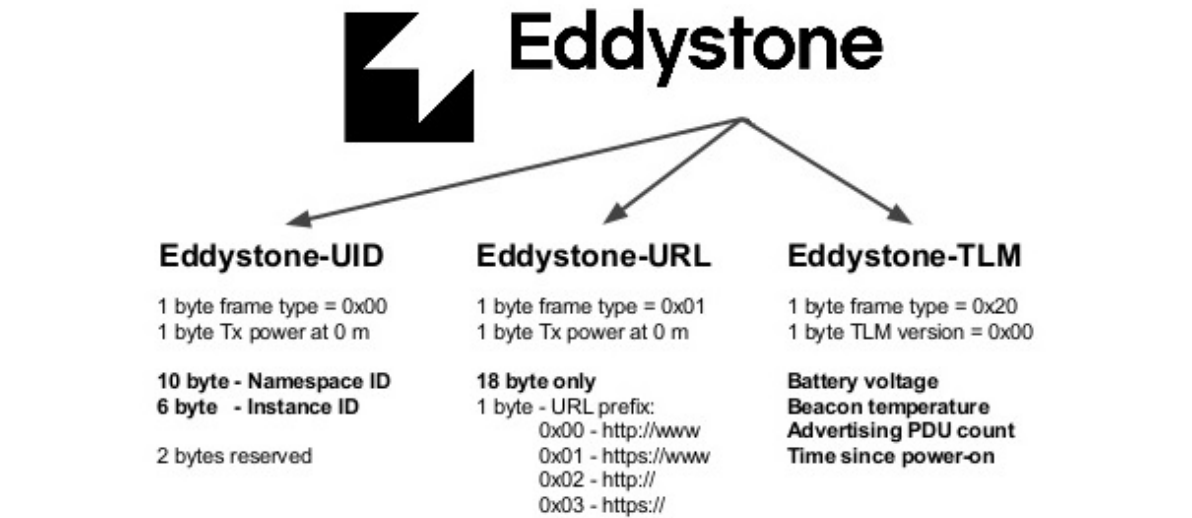


Figure 45: Eddystone Modes Supported by the Dialog Beacon Reference Design

Figure 46 describes the various fields of the different Eddystone Beacon modes.

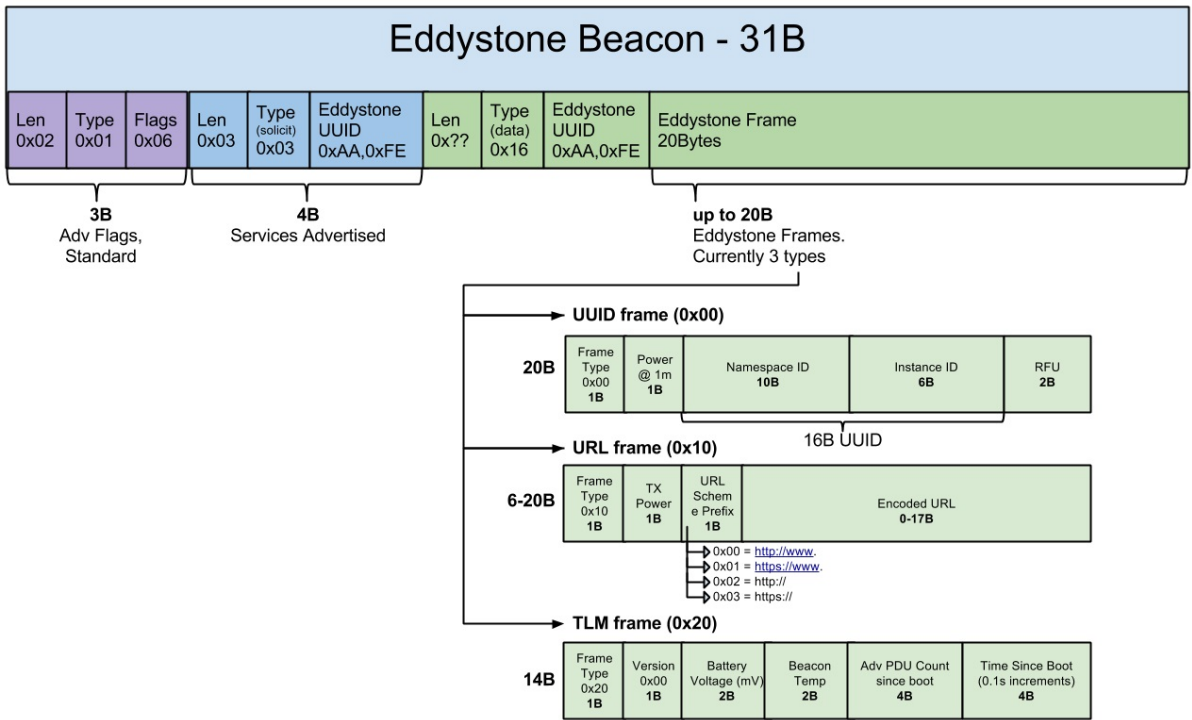


Figure 46: Eddystone Different Mode Frames Analyzed

The specific type of Eddystone frame is encoded in the high-order four bits of the first octet in the Service Data associated with the Service UUID. Permissible values are shown in Table 68.

Table 68: Eddystone Frame Types

Frame Type	High Order 4 bits	Byte Value
UID	0000	0x00
URL	0001	0x10
TLM	0010	0x20
EID	0011	0x30
RESERVED	0100	0x40

Note 15 The four low-order bits are reserved for future use and shall be 0000.

Note 16 Although the core Bluetooth data types are defined in the standard as little-endian, Eddystone's multi-value bytes defined in the Service Data are all big-endian.

7.4.3.1 Eddystone-UID

The Eddystone-UID frame broadcasts an opaque and unique 16-byte Beacon ID composed of a 10-byte namespace and a 6-byte instance. The Beacon ID is useful in mapping a device to a record in external storage. The namespace portion of the ID can be used to group a particular set of beacons, while the instance portion of the ID identifies individual devices in the group. The division of the ID into a namespace and an instance can also be used to optimize BLE scanning strategies, for example by filtering only on the namespace.

The UID frame is encoded in the advertisement as a Service Data block associated with the Eddystone service UUID. The frame layout is shown in [Table 69](#).

Table 69: Eddystone UID Frame

Byte Offset	Field	Description
0	Frame Type	Value = 0x00
1	Ranging Data	Calibrated Tx power at 0 m
2	NID[0]	10-byte Namespace
3	NID[1]	
4	NID[2]	
5	NID[3]	
6	NID[4]	
7	NID[5]	
8	NID[6]	
9	NID[7]	
10	NID[8]	
11	NID[9]	
12	BID[0]	6-byte Instance
13	BID[1]	
14	BID[2]	
15	BID[3]	
16	BID[4]	

Byte Offset	Field	Description
17	BID[5]	
18	RFU	Reserved for future use, must be 0x00
19	RFU	Reserved for future use, must be 0x00

More on information on the Eddystone-UID specification can be found in [\[10\]](#).

7.4.3.2 Eddystone-URL

The Eddystone-URL frame broadcasts a URL using a compressed encoding format in order to fit more within the limited advertisement packet. Once decoded, the URL can be used by any client with access to the internet.

Table 70: Frame Specification

Byte Offset	Field	Description
0	Frame Type	Value = 0x10
1	TX Power	Calibrated Tx power at 0 m
2	URL Scheme	Encoded Scheme Prefix
3+	Encoded URL	Length 1 to 17

For URLs longer than 17 bytes, a [URL shortener](#) is recommended.

Table 71 URL Scheme Prefix

Decimal	Hex	Expansion
0	0x00	http://www.
1	0x01	https://www.
2	0x02	http://
3	0x03	https://

The HTTP URL scheme is defined by [RFC 1738](#). The encoding consists of a sequence of characters. Character codes excluded from the URL encoding are used as text expansion codes. When a user agent receives the Eddystone-URL, the byte codes in the URL identifier are replaced by the expansion text according to [Table 72](#).

Table 72: Eddystone-URL HTTP URL Encoding

Byte Offset	Field	Description
0	0x00	.com/
1	0x01	.org/
2	0x02	.edu/
3	0x03	.net/
4	0x04	.info/
5	0x05	.biz/
6	0x06	.gov/
7	0x07	.com

Byte Offset	Field	Description
8	0x08	.org
9	0x09	.edu
10	0x0A	.net
11	0x0B	.info
12	0x0C	.biz
13	0x0D	.gov
14..32	0x0E...0x20	Reserved for future use
127..255	0x7F...0xFF	Reserved for future use

URLs are written only with the graphic printable characters of the US-ASCII coded character set. The octets 00 to 20 and 7F to FF hexadecimal are not used. See “Excluded US-ASCII Characters” in [RFC 3986](#).

IMPORTANT NOTE

In short, the URL-prefix is represented by one byte (see [Table 71](#)), followed by the URL in ASCII, and succeeded by the extension (see [Table 72](#)), if applicable.

Below an example of a URL frame is presented:

```
uint8_t url_adv_data[] =
{
    0x03, // Length of Service List
    0x03, // Param: Service List
    0xAA, 0xFE, // Eddystone ID
    0x0E, // Length of Service Data
    0x16, // Service Data
    0xAA, 0xFE, // Eddystone ID
    0x10, // Frame type: URL
    0xC5, // Power
    0x00, // http://www.
    'd', 'i', 'a', 's', 'e', 'm', 'i',
    0x07, // .com
};
```

For the development of the Dialog Beacon reference design, an Eddystone-URL generator was used, however it is not essential to create a URL frame if one follows the specification. More information on the Eddystone-URL specification can be found in [\[11\]](#).

7.4.3.3 Unencrypted Eddystone-TLM

Eddystone Beacons may transmit data about their own operation to clients. This data is called telemetry and is useful for monitoring the health and operation of a fleet of beacons. Since the Eddystone-TLM frame does not contain a beacon ID, **it must be paired with an identifying frame which provides the ID, either of type Eddystone-UID or Eddystone-URL.**

Table 73: Eddystone TLM Frame Specification

Byte offset	Field	Description
0	Frame Type	Value = 0x20
1	Version	TLM version, value = 0x00

Byte offset	Field	Description
2	VBATT[0]	Battery voltage, 1 mV/bit
3	VBATT[1]	
4	TEMP[0]	Beacon temperature
5	TEMP[1]	
6	ADV_CNT[0]	Advertising PDU count
7	ADV_CNT[1]	
8	ADV_CNT[2]	
9	ADV_CNT[3]	
10	SEC_CNT[0]	Time since power-on or reboot (up timer)
11	SEC_CNT[1]	
12	SEC_CNT[2]	
13	SEC_CNT[3]	

More information on the unencrypted Eddystone-TLM specification can be found in [\[12\]](#).

7.5 Beacon Parameters

The following subsections describe how to modify the basic parameters of the beacon: advertising data and advertising interval.

7.5.1 Advertising Data

The data to be advertised is derived from a structure by the name of `user_beacon_config_tag`. The struct contains fields that can serve any beacon format or mode. In [Table 74](#), the `user_beacon_config_tag` struct is analyzed.

Table 74: Format of Struct `user_beacon_config_tag`

Type	Name	Size (B)	Description
uint8_t	uuid[16]	16	UUID value if iBeacon, AltBeacon or Eddystone-UID
uint16_t	major_ALT_val1	2	iBeacon Major value or AltBeacon value 1
uint16_t	minor_ALT_val2	2	iBeacon Minor value or AltBeacon value 2
uint16_t	company_id	2	Manufacturer ID
uint16_t	adv_int	2	Advertising Interval
uint8_t	power	1	Reference value of the received signal strength (RSSI) measured at 1 m. The value is represented in signed format. (Note 17)
uint8_t	beacon_type	1	Beacon Type (iBeacon, AltBeacon, Eddystone UID/URL)
uint8_t	url_prefix	1	http://www. or https://www. or http:// or https://
uint8_t	url[19]	19	The url preceded by the length of service data and succeeded by the extension (.com, .net and others)
uint8_t	TLM_version	1	The TLM version of the TLM service (if TLM is used). (Note 18)
uint8_t	TLM_used	1	Flag that shows if TLM is used or not. (Note 18)

Note 17 To estimate the distance to a transmitting beacon, the receiving device uses and calibrates RSSI by the measured power parameter, which is included in the advertising data. The measured power parameter is a 1-byte value in signed representation. The value depends on the RF transmit power and antenna implementation of the hardware. For the Dialog Beacon, the RSSI level measured at 1 m was -59 dBm (see Appendix F.1).

Note 18 Eddystone-TLM is not an "independent" Eddystone beacon mode. It must be paired with either Eddystone-UID or Eddystone-URL (see section 7.4.3.3).

Two methods are used to populate the contents of the struct:

- Using the `user_default_beacon_config` struct (defined in the code) and programming the desired values.
- Reading the contents of a device configuration struct in the Flash memory.

The two methods are explained in detail in section 7.5.1.1 and 7.5.1.2.

7.5.1.1 Using the `user_default_beacon_config` Struct

If a Flash memory is not available or not to be used, the contents of the `user_beacon_config_tag` struct are populated with the contents of the `user_default_beacon_config` struct. Below an example of a populated `user_default_beacon_config` struct is presented:

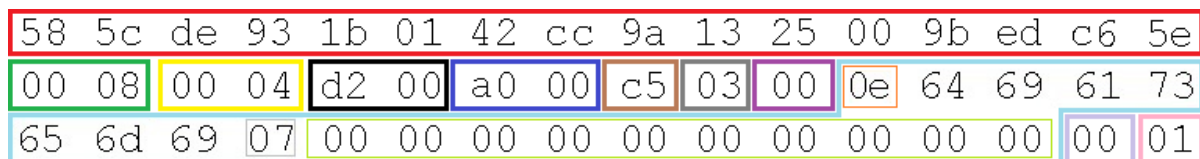
```
const struct user_beacon_config_tag user_default_beacon_config = {
    .uuid = {0x58, 0x5C, 0xDE, 0x93, 0x1B, 0x01, 0x42, 0xCC, 0x9A, 0x13, // 10-byte Namespace
             0x25, 0x00, 0x9B, 0xED, 0xC6, 0x5E}, // 6-byte Instance
    .major_ALT_val1 = 0x0800, // Major/Alt1 Value
    .minor_ALT_val2 = 0x0400, // Minor/Alt2 Value
    .company_id = DIALOG_COMP_ID, // Beacon company ID
    .adv_int = BEACON_ADVERTISING_INTERVAL, // Advertising interval
    .power = 0xC5, // Tx Power
    .beacon_type = xxx_BEACON_TYPE, // Not used
    .url_prefix = HTTPWWW, // Populated if used
    .url = {0x0E, 'd', 'i', 'a', 's', 'e', 'm', 'i', DOTCOM}, // Populated if used
    .TLM_version = 0x00, // Populated if used
    .TLM_used = 0x01 // Populated if used
};
```

Note 19 The fields `major_ALT_val1` and `minor_ALT_val2` are in big endian format.

This example shows how users can use the same struct to alternate between different beacon types. Depending on the application, some of the fields of the struct are used and some are not. For example, if the application uses the Eddystone-URL beacon type (see 7.4.3.2), the fields `".uuid"`, `".major_ALT_val1"`, and `".minor_ALT_val2"` are not used. If the beacon type is different, for example, Eddystone-UID (see 7.4.3.1), the fields `".url_prefix"` and `".url"` would not be used. The advertised data packet is synthesized according to the beacon type.

7.5.1.2 Reading Advertising Data from Flash

If a Flash memory is to be used, the program reads the advertising data values from a device configuration struct that is written in Flash memory. Appendix D describes how to create an image to burn into Flash memory. The device configuration struct is by default located at address 0x30000, but this can easily be configured by the value set in the product header (see section D.1.1). The data are written in Flash in the order of the `user_beacon_config_tag` struct. Figure 47 shows an example of the contents of a configuration struct in a Flash memory.

**Figure 47: Example of a Device Configuration Struct in Flash Memory**

In [Figure 47](#), from left to right (see [Table 74](#) for more information):

- Red is the uuid
- Green is the major_ALT_val1 value
- Yellow is the minor_ALT_val2 value
- Black is the company_id
- Blue is the adv_int
- Brown is the Tx power
- Grey is the beacon_type
- Purple is the url_prefix
- Light blue is the url in ASCII (here it is "diasemi") with:
 - Orange is the length of the service data
 - Light grey is the url postfix (here it is the code for .com)
 - Light green bytes are not needed (because this specific url is shorter) and are set to 0x00
- Light purple is the TLM_version
- Pink is the TLM_used flag

Note 20 The values are in opposite endianness compared to the code.

7.5.2 Advertising Interval

The BLE Beacon sends advertising packets at a certain time interval. This is called the advertising interval. According to the BLE specification, the advertising interval can be set from 20 ms up to 10.24 s.

At a shorter advertising interval, the radio of the beacon will be enabled more often, resulting in higher average power consumption. A shorter advertising interval will also lead to an increased number of packets per second at the receiver and therefore result in a more accurate RSSI reading.

If the advertising data derive from Flash, then the advertising interval takes the value of the adv_int field of the device configuration struct, written in flash ([Figure 47](#), Blue field).

If no Flash memory is used, the advertising interval parameter can be changed at the location shown in [Table 75](#).

Table 75: Advertising Interval Location

Parameter	Macro	Project	File Name
Beacon advertising interval	BEACON_ADVERTISING_INTERVAL	All beacons	user_config.h
SUOTA advertising interval	SUOTA_ADVERTISING_INTERVAL	All beacons	user_config.h

7.6 Software Features

This section explains the advanced software features of the Dialog BLE Beacon. The following configurations are supported:

- Non-connectable advertising (Beacon mode)
 - Allows users to advertise data with the lowest power consumption.
- Connectable advertising (Peripheral mode)
 - Allows users to connect to a central device to run SUOTA, and use official BLE and custom 128-bit profiles.
- Dynamically change beacon data
 - Values `major_ALT_value1` and `minor_ALT_value2` are periodically updated with measured data. In the case of the "altbeacon_dynamic" project, these are data from the environmental sensor.
 - TLM data (up timer, temperature, battery readings) are updated.
- Support of SPI Flash memory
 - Power off/on for power saving.
 - Storage of the beacon configuration (product header, configuration structure).
 - Storage of an updated image received through SUOTA.
 - Storage of boot loader.
- SUOTA
 - After a connection has been established, the firmware can be updated.
 - Dual image boot loader.
- Custom 128-bit profiles
 - Environmental Data Notifications proprietary profile. This service makes data from the environmental sensor (temperature, humidity, pressure) available to the user.
 - Device configuration proprietary profile. The beacon configuration can be read and modified by the central device.

7.7 Software Architecture

Figure 48 presents how the Beacon reference applications are structured.

Table 76 shows the source files of the Beacon reference applications.

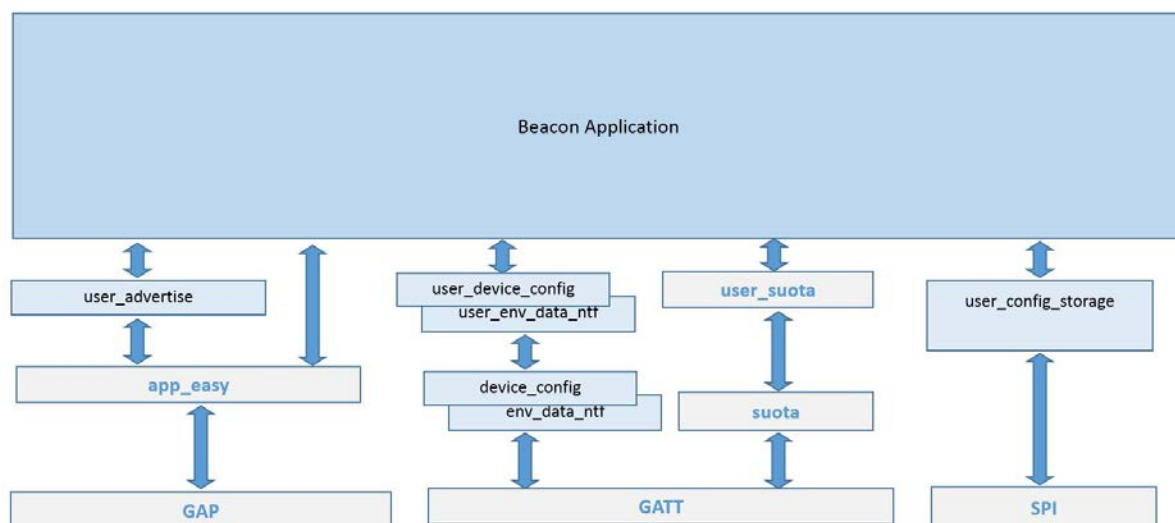


Figure 48: Beacon SW System Overview

Table 76: Source Files of Beacon Reference Applications

Group	File Name	Project	Description
user_platform	user_periph_setup.c	All beacons	Peripheral modules initialization, GPIO pins assignment
user_config_storage	user_config_storage.c	All beacons	Application configuration data storage API
device_config	device_config.c device_config_task.c user_device_config.c user_device_config_task.c	Eddystone, iBeacon	Device Configuration profile
env_data_ntf	env_data_ntf.c user_env_data_ntf.c env_data_ntf_task.c user_env_data_ntf_task.c	Eddystone	Environmental Data Notifications Profile
user_drivers	i2c_gpio_extender.c user_iot_dk_utils.c battery.c	All beacons (battery.c not in AltBeacon)	User drivers for the I2C GPIO extender, battery and MSK HW peripherals
bme680_drivers	bme680.c bme680_impl.c.c	Eddystone, AltBeacon	Environmental Sensor Drivers
sensor_inteface	sensors_inteface.c sensors_inteface_api.c environmental_bme680.c	Eddystone, AltBeacon	Sensors interface API
utilities	wkup_adapter.c sensors_periph_interface.cc rc32.c	All beacons	Various utilities for wakeup, crc and peripherals
user_beacon	user_eddy_uid_url_tlm.c	Eddystone	Application Code

Group	File Name	Project	Description
	user_altbeacon_dynamic.c	AltBeacon	
	user_ibeacon_suota_button.c	iBeacon	
user_adv_api	user_advertise.c	All beacons	User Advertise SW module

7.8 Operation Overview

Figure 49 presents the system operation.

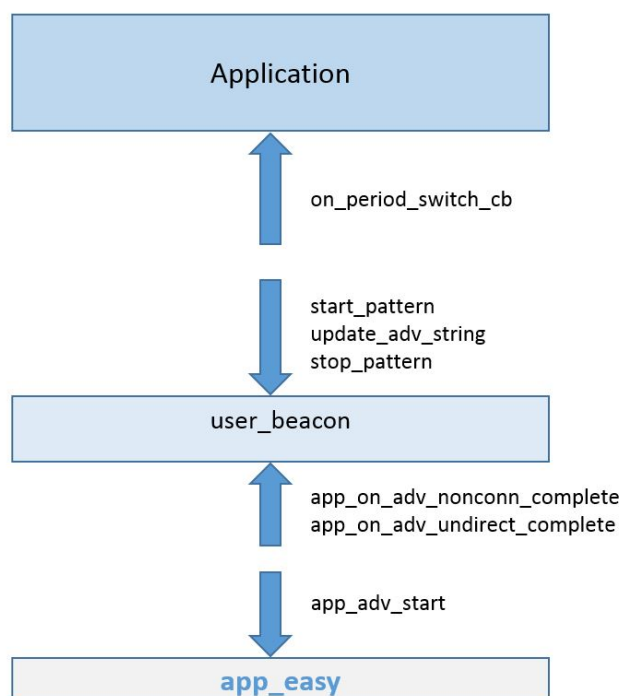


Figure 49: Operation Overview

7.8.1 Configuration Switches

The code contains various configuration switches which include or exclude functionalities from the code. They are divided into two categories:

- Software configuration switches
- Profile configuration switches

The various configuration switches are outlined in [Table 77](#) and [Table 78](#).

Table 77: List of Software Configuration Switches

Switch	Used in	Description
CFG_CONFIG_STORAGE	AltBeacon, Eddystone, iBeacon	Read/Write configuration from/to Flash memory enabled.
CFG_DYNAMIC_BEACON_DATA	AltBeacon	If defined, the major/minor values of the beacon are updated dynamically with data from the Environmental sensor.
CFG_DEV_CNF_HDR_CRC32_SUPPORT	AltBeacon, Eddystone, iBeacon	If defined, a CRC32 value of the configuration struct data is calculated and compared to the CRC_word included in the configuration struct header.
USE_EDDYSTONE_UID/USE_EDDYSTONE_URL	Eddystone	Eddystone-UID or Eddystone-URL packets will be advertised depending on which will be defined

Table 78: List of Profile Configuration Switches

Switch	Used in	Description
CFG_PRF_DEVICE_CONFIG	Eddystone, iBeacon	Custom 128-bit "Device Configuration" profile is enabled.
CFG_PRF_ENV_DATA_NTF	Eddystone	Enables custom 128-bit "Environmental Data Notifications" profile.
CFG_PRF_SUOTAR	iBeacon	Dialog SUOTA profile is enabled. CFG_SPI_FLASH switch must be defined.
CFG_PRF_DISS	Eddystone, iBeacon	Device Information Service Server (DISS) profile.
CFG_PRF_BASS	Eddystone, iBeacon	Battery Service profile. Battery readings are updated using an advanced battery reporting mechanism.

7.9 User Advertise SW Module

The user advertise software module provides users with an easy API to create configurable advertising strings with easily configurable advertising parameters (for example, advertising interval). Below the structural components of the user advertise module are outlined.

7.9.1 Style

The main block of the user advertise SW module is called a **style**. Below, the style type is presented:

```

/// Advertising Style
typedef struct user_adv_style
{
    /// Advertising interval
    uint16_t adv_int;
    /// Advertising length
    uint8_t adv_len;
}

```

```

    /// Counter of advertising events
    uint16_t cnt_upd_adv_string;
    /// Advertising mode
    uint8_t adv_mode;
    /// Advertising data
    uint8_t adv_data[31];
} user_adv_style_t;

```

An advertising style contains information about:

- the advertising interval,
- the length of the advertising string,
- the amount of advertising events of that specific style before switching to another style,
- the advertising mode (undirected or non-connectable)
- a 31-byte array containing the actual data to be advertised.

7.9.2 Pattern

An array of styles is called a **pattern**. The maximum number of styles allowed per pattern is configurable and is defined by the MAX_STYLES_PER_PATTERN macro.

The value of the cnt_upd_adv_string field of each style shows how many advertising events will take place before a style switches to the next. Below an example of a pattern is presented.

```

user_adv_style_t ibeacon_suota_pattern[2] =
{
    {
        .adv_int = iBEACON_ADV_INT,
        .adv_len = iBEACON_ADV_LEN,
        .cnt_upd_adv_string = iBEACON_EVENTS,
        .adv_mode = NON_CONNECTABLE_MODE,
        .adv_data = <pointer to a data array>
    },
    {
        .adv_int = SUOTA_ADVERTISING_INTERVAL,
        .adv_len = USER_ADVERTISE_DATA_LEN,
        .cnt_upd_adv_string = SUOTA_EVENTS_TO_SWITCH,
        .adv_mode = UNDIRECTED_MODE,
        .adv_data = USER_ADVERTISE_SUOTA,
    }
};

```

In this example, a pattern is declared, consisting of two styles. The data for the second style of the pattern are derived from the array shown below:

```
#define USER_ADVERTISE_DATA      "\\x05"\  
                                  ADV_TYPE_COMPLETE_LIST_16BIT_SERVICE_IDS\  
                                  ADV_UUID_DEVICE_INFORMATION_SERVICE\  
                                  ADV_UUID_SUOTAR_SERVICE\  
                                  "\\x0E"\  
                                  ADV_TYPE_COMPLETE_LOCAL_NAME\  
                                  USER_DEVICE_NAME
```

Figure 50 presents how to use a user advertise SW block.

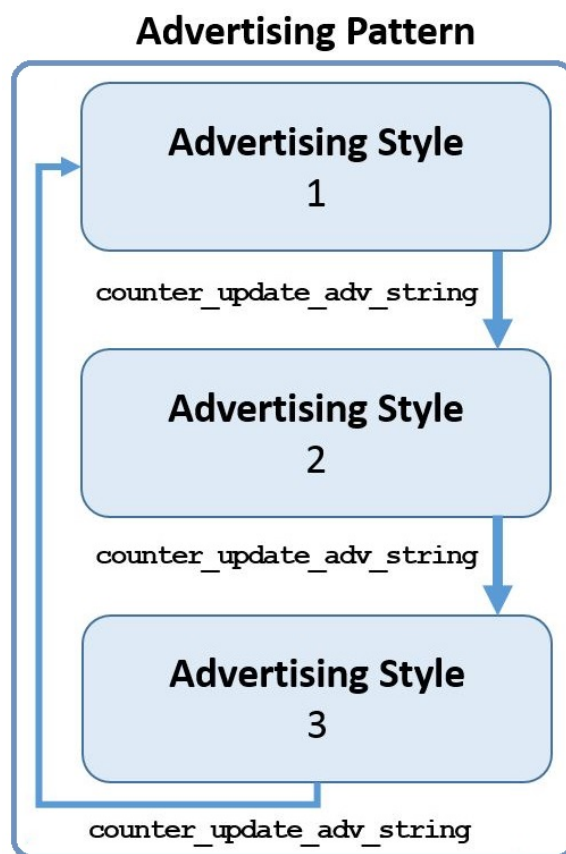


Figure 50: User Advertise usage example

In this example, the pattern contains three styles. When `cnt_upd_adv_string` of the first style is reached, in other words, `< cnt_upd_adv_string >` events have been advertised, the application switches to the second style. The same happens when the `cnt_upd_adv_string` of the second style is reached. However, when `< cnt_upd_adv_string >` events of the third advertising style have been advertised, the application starts advertising again from the first style. The row of events, or the times a style is advertised, depends on the application.

7.9.3 User Advertise SW Module Callbacks

The User Advertise SW Module makes two callbacks to the application and users can use these callbacks for any desired usage. In the provided examples, a callback is used to inform the application of advertising events and the other callback is called every time the `user_on_ble_powered` (Note 21) system callback is called. More information on the User Advertise SW module API is included in the doxygen documentation that is included in the release.

Note 21 `user_on_ble_powered` updates the advertising string.

7.10 Device Configuration Service

The Device Configuration Service is a custom 128-bit profile developed by Dialog Semiconductor. The profile provides a generic interface to a peripheral device for reading and writing configuration parameters of the application, irrespective of the number, type, and size of the parameters.

7.10.1 Device Configuration Service Specification

The Device Configuration Service exposes four characteristics. The characteristics of the service are outlined in Table 79.

Table 79: Characteristics of the Device Configuration Service

Characteristic Name	Qualifier	Properties	Size (B)
Configuration structure version	Mandatory	Read	66
Write configuration	Mandatory	Write	1
Read command	Mandatory	Write	1
Read response	Mandatory	Indicate	67

- Configuration structure version:
 - It identifies the type and version of the configuration structure of the application.
 - It is used as a convention between device configuration server and client for the format of the configuration data.
 - The Device configuration client should be aware of the ID and size of each configuration parameter.
- Write configuration:
 - The Device configuration client writes the configuration data into this characteristic.
 - The data format is shown in Figure 51:

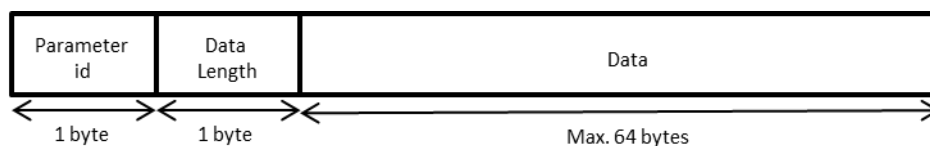


Figure 51: Data Format in Write Configuration

- The maximum configuration data size supported by the Device configuration service is 64 bytes. Up to 256 parameters can be supported. The client can read the parameter data after the end of a write operation.
- Read Command:

- The Device configuration client writes a parameter ID into this characteristic, requesting the command server to return the current values of the parameters.
- Read Response:
 - The Device configuration server sends an indication including the configuration data of a parameter, whenever the client requests it by writing the parameter ID to the read command characteristic.
 - The format of the indication data is shown in [Figure 52](#).

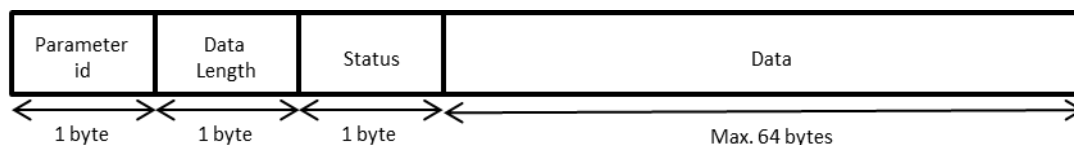


Figure 52: Indication Data Format in Read Response

7.11 Environmental Data Notifications Service

The Environmental Data Notifications Service is a custom 128-bit profile developed by Dialog Semiconductor. The profile provides a generic interface to a peripheral device providing data from the environmental sensor (temperature, humidity, and pressure).

7.11.1 Environmental Data Notifications Service Specification

The Environmental Data Notifications Service exposes five characteristics. The characteristics of the service are outlined in [Table 80](#).

Table 80: Characteristics of the Environmental Data Notifications Service

Characteristic Name	Qualifier	Properties	Size (B)
Temperature	Mandatory	Read	2
Pressure	Mandatory	Read	4
Humidity	Mandatory	Read	4
External Event	Mandatory	Read	1
Sampling Interval	Mandatory	Read/Write	2

- Temperature:
 - The "Temperature" characteristic carries the temperature value read by the environmental sensor.
 - Executing a read command, the current temperature value from the sensor will be transmitted. The same will happen upon a button press, if characteristic notifications are enabled.
- Pressure:
 - The "Pressure" characteristic carries the pressure value read by the environmental sensor.
 - Executing a read command, the current pressure value from the sensor will be transmitted. The same will happen upon a button press, if characteristic notifications are enabled.
- Humidity:
 - The "Humidity" characteristic carries the humidity value read by the environmental sensor.
 - Executing a read command, the current humidity value from the sensor will be transmitted. The same will happen upon a button press, if characteristic notifications are enabled.

- External Event:
 - The "External Event" characteristic carries the external event state (in our case the state of the button).
 - Executing a read command, the current button state will be transmitted. The same will happen upon a button press, if characteristic notifications are enabled.
- Sampling Interval:
 - The "Sampling Interval" characteristic carries the value of the timer interval during which the environmental sensor samples environmental data.
 - These data are used to update the values of the aforementioned characteristics (temperature, pressure, and humidity).

7.12 Beacon Configuration

Depending on the defined software configuration switches, the configuration data can be modified by a central device over the Device Configuration Service or written into SPI Flash memory during the production phase.

The Beacon application reference software uses a configuration storage module, implemented in file `user_config_storage.c`, to fetch and store configuration data from and into the non-volatile memory. The configuration storage module provides a list of common API functions to any DA14585 application. The current version only supports SPI Flash memory.

7.12.1 Beacon Configuration Memory Map

Configuration storage uses the memory map of the dual image boot loader. More specifically, it uses the first four bytes of the 'Reserved' field (byte offset 12) in the Product Header, as described in [D.1.1](#), to define the memory address of the configuration area. The address shall point to the start of an SPI Flash sector and no other information shall be stored in the same sector.

Device configuration data are stored at the start of the configuration area and consist of the device configuration header, followed by the device configuration struct. The size of the configuration data is 112 bytes and the format is outlined in [Table 81](#).

Table 81: Configuration Data Format

Byte #	Field
Device Configuration Header	
0, 1	Signature (0x70, 0x53)
2	Valid flag
3	Number of items
4 to 7	CRC
8 to 23	Version
24, 25	Data size
26	Encryption flag
27 to 63	Reserved
Device Configuration Struct	
64 to 79	UUID
80, 81	Major_ALT_val1

Byte #	Field
82, 83	Minor_ALT_val2
84, 85	Company ID
86, 87	Advertising Interval
88	Tx Power
89	Beacon Type (iBeacon / AltBeacon / Eddystone (UUID/URL))
90	URL prefix
91 to 109	URL
110	TLM version
111	TLM_used flag

The fields included in the header are:

- Signature:
 - A "magic" number identifying the configuration header.
 - Value: 0x70, 0x53.
- Valid flag:
 - A value of 0xAA denotes a valid image.
- Number of items:
 - The number of configuration parameters in the configuration data.
- CRC (If used):
 - The checksum calculated over the configuration data.
- Version:
 - Determines the configuration structure type and version.
 - Can be checked by the application to confirm that the expected data are stored.
- Data size:
 - Size of the configuration data (in bytes).
- Encryption flag:
 - Indicates whether the configuration data have been encrypted. Encryption of the configuration in memory is not supported for this software release.
- UID: The value in this field serves for different uses depending on the mode used.
 - If iBeacon is selected, this field contains the 16-byte UUID value.
 - If AltBeacon is selected, this field contains the 16-byte Beacon ID value ([Note 14](#)).
 - If Eddystone-UID is selected, this field contains the 10-byte namespace value followed by the 6-byte instance value.
- Major_ALT_val1: Contains the Major value for iBeacon or the Alt_val1 for AltBeacon.
- Minor_ALT_val2: Contains the Minor value for iBeacon or the Alt_val2 for AltBeacon.
- Company ID: The beacon device manufacturer's company identifier code, see [\[7\]](#).
- Advertising Interval: The advertising interval for the beacon (see section [7.5.2](#)).
- Tx Power: A 1-byte value representing the average RSSI at 1 m from the advertiser.
- Beacon Type: The beacon type included in the Configuration Struct.

- URL Prefix: If Eddystone-URL beacon type is selected, this field defines the URL prefix (see [Table 71](#)).
- URL: If Eddystone-URL beacon type is selected, this field contains the URL in ASCII, preceded by the length of service data and succeeded by the extension (.com, .net and others).
- TLM_version: TLM version is reserved for future development of this frame type. At present the value must be 0x00.
- TLM_used flag: This flag indicates whether the TLM service is used or not.

7.13 Battery Level Sampling

If BATTERY_SAMPLING_ENABLED has been defined, a battery level and voltage averaging mechanism is enabled.

A user driver has been developed so as to achieve more accurate battery readings. The main idea is to periodically capture an ADC sample from the VBAT3V signal immediately after the device wakes up, so as to avoid any possible battery consuming activities. These samples allow the driver to calculate average values for the battery voltage and battery level. The calculated average level and voltage values can then be used by the application.

Upon connection, if the BASS service is enabled, `app_batt_set_level(battery_return_avg_lvl())` is called which sets the battery level in the BASS service making the averaged battery level available when connected.

7.14 Beacon Examples for DA14585 IoT MSK

The Beacon reference applications based on DA14585 IoT MSK HW reference design comes with three distinct beacon examples that make use of all the different beacon types and features supported by Dialog Semiconductor. Below these examples are outlined.

7.14.1 AltBeacon

In this example a non-connectable AltBeacon string is advertised containing a 16-byte UID followed by 4 bytes (2 bytes for ALT_Val1 and 2 bytes for ALT_Val2). If the software-built flag DYNAMIC_BEACON_DATA is not enabled, then ALT_Val1 and ALT_Val2 are populated by the values set in the configuration struct in flash (if CONFIG_STORAGE is enabled) or by the default values hardcoded in the default configuration struct. If DYNAMIC_BEACON_DATA is enabled, then the 2-byte values are populated by environmental sensor data, refreshed every `periodic_read_interval`, set by users when initializing the environmental sensor.

[Figure 53](#) presents the advertising transition diagram of the AltBeacon example:

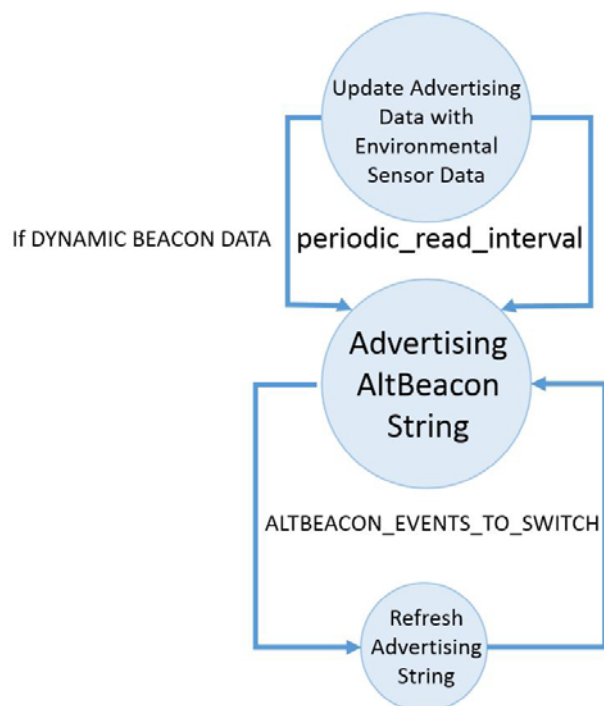


Figure 53: AltBeacon Example Transition Diagram

7.14.1.1 AltBeacon Example Sequence Diagram

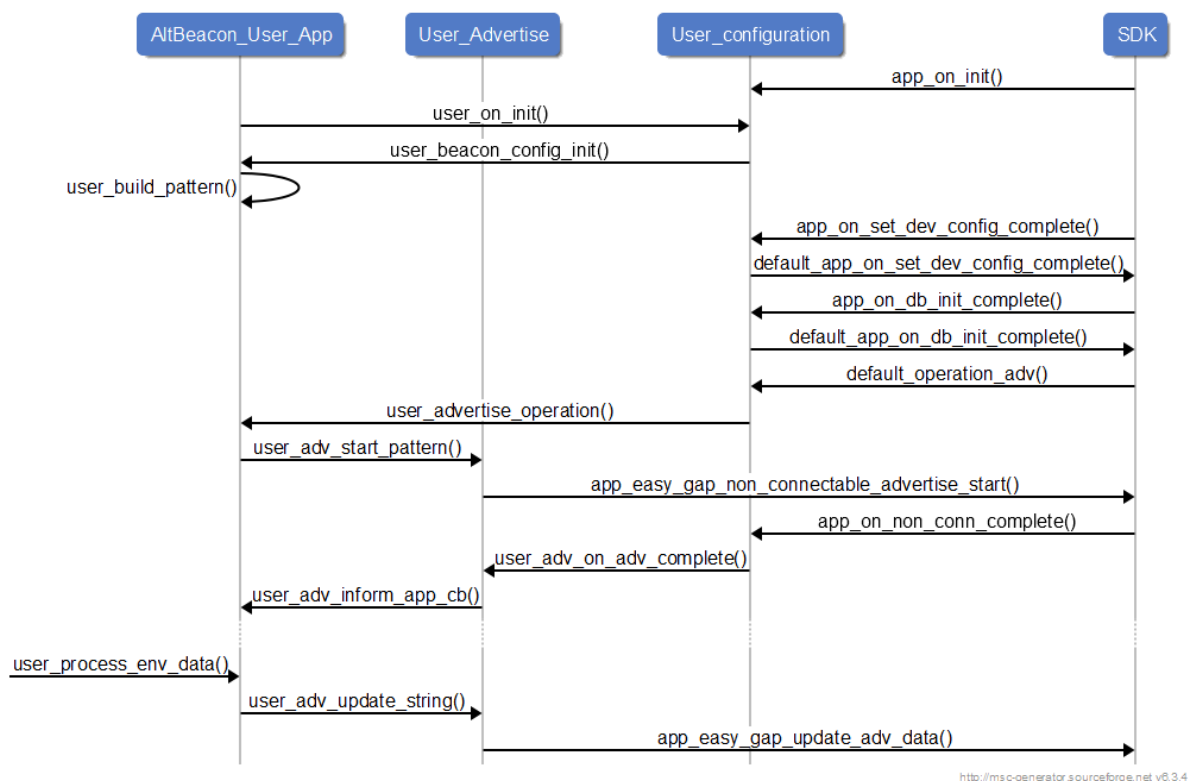


Figure 54: Altbeacon Sequence Diagram

7.14.2 Eddystone

In this example, a connectable Eddystone-UID or Eddystone URL string, depending on the corresponding build flags `USE_EDDYSTONE_UID` or `USE_EDDYSTONE_URL`, is advertised. The advertising string contains a 16-byte UID, if the flag `USE_EDDYSTONE_UID` is used. Otherwise, for the flag `USE_EDDYSTONE_URL`, an encoded URL with a length ranging from 1 to 17 bytes is contained in the advertising string. Every time when `user_adv_on_adv_complete` is called, the application advertises an EDDYSTONE-TLM advertising string and then returns to advertising Eddystone-UID or Eddystone-URL strings. As explained in 7.4.3.3, the Eddystone-TLM packet contains information about the battery voltage of the device. The application makes use of the battery averaging mechanism described in 7.13, providing more accurate voltage values to the Eddystone-TLM advertising string.

When connected to a peripheral, the device provides four different GATT services: DISS and BASS which are official BLE GATT services and two Dialog proprietary GATT services, the `env_data_ntf` and `device_config`. The `device_config` and `env_data_ntf` services are described in 7.10 and 7.11.

Figure 55 presents the advertising transition diagram of the Eddystone example:

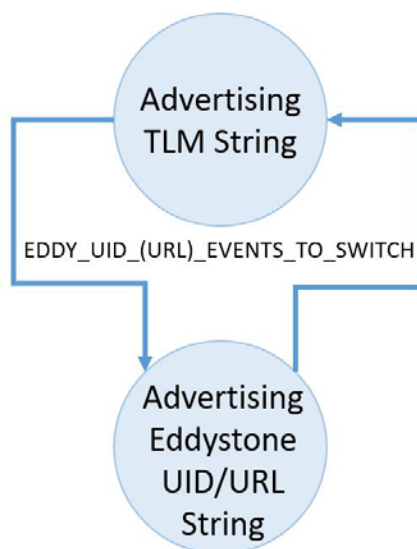


Figure 55: Eddystone UID/URL - TLM Example Transition Diagram

7.14.2.1 Eddystone Example Sequence Diagram

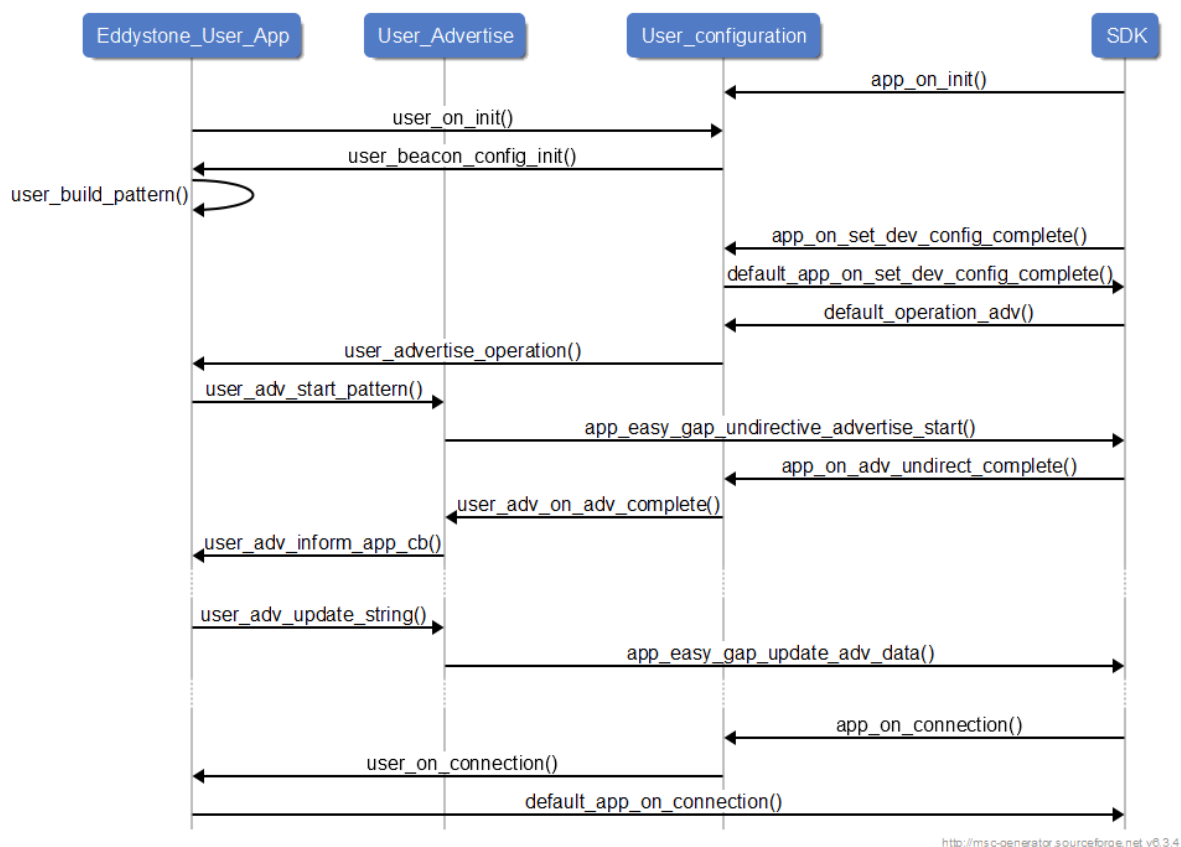


Figure 56: Eddystone Sequence Diagram

7.14.3 iBeacon

In this example, a non-connectable iBeacon string is advertised. The Major and Minor values are populated by the values set in the configuration struct in flash (if CONFIG_STORAGE is enabled) or by the default values hardcoded in the default configuration struct. On button press the device starts advertising a connectable SUOTA string for a duration of ADV_SUOTA_TIMEOUT, set in `ibeacon_suota_button.h`. However, if during this time period there is another button press, then the application returns to advertising the non-connectable iBeacon string.

Figure 57 presents the advertising transition diagram of the iBeacon example:

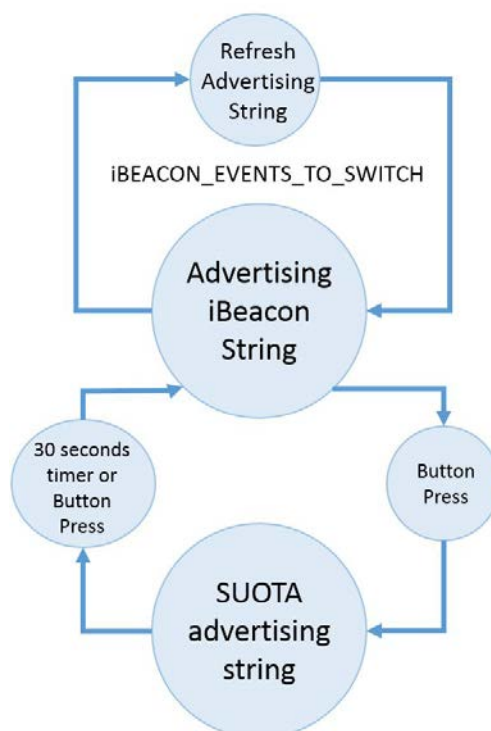


Figure 57: iBeacon Example Transition Diagram

7.14.3.1 iBeacon Example Sequence Diagram

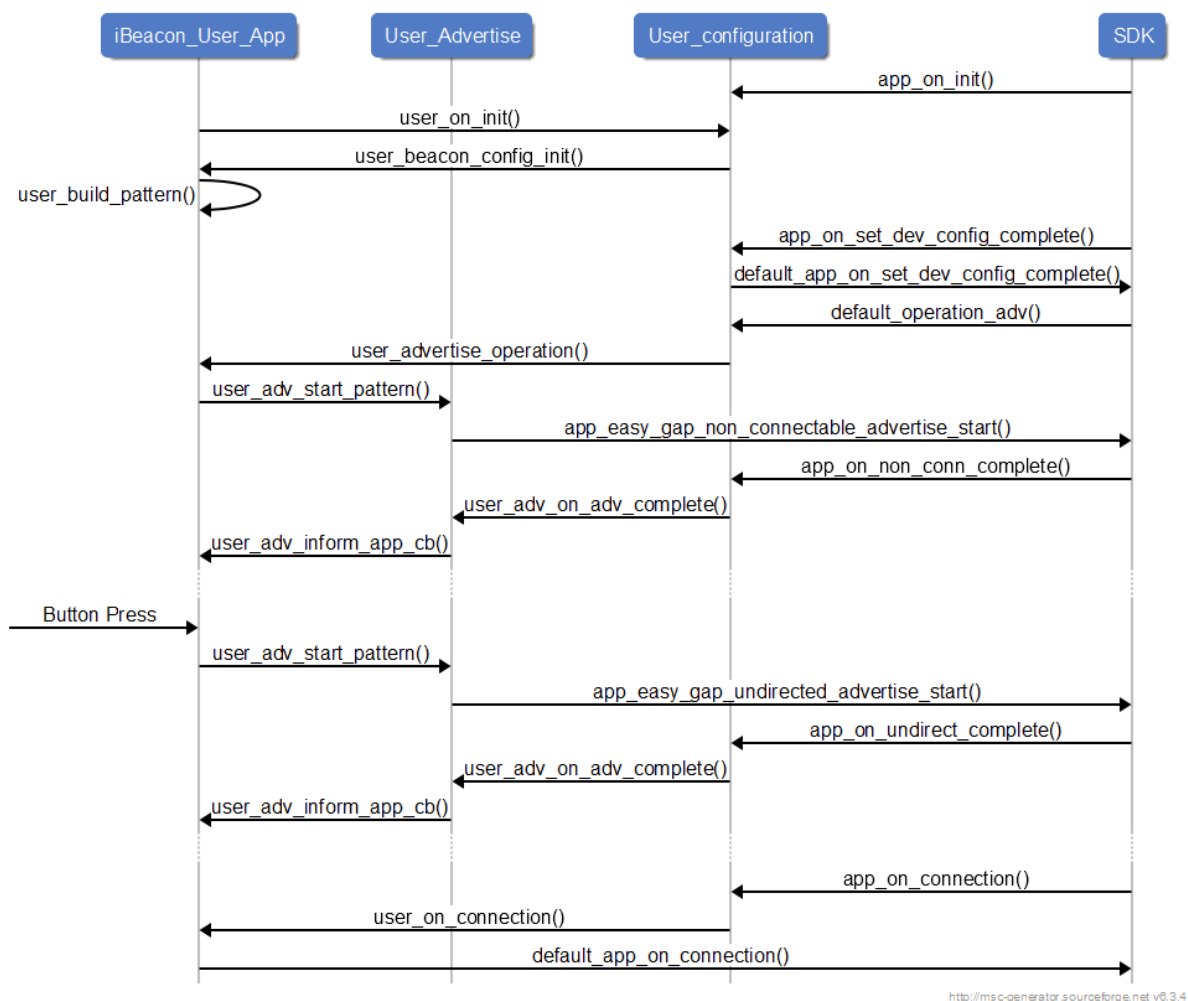
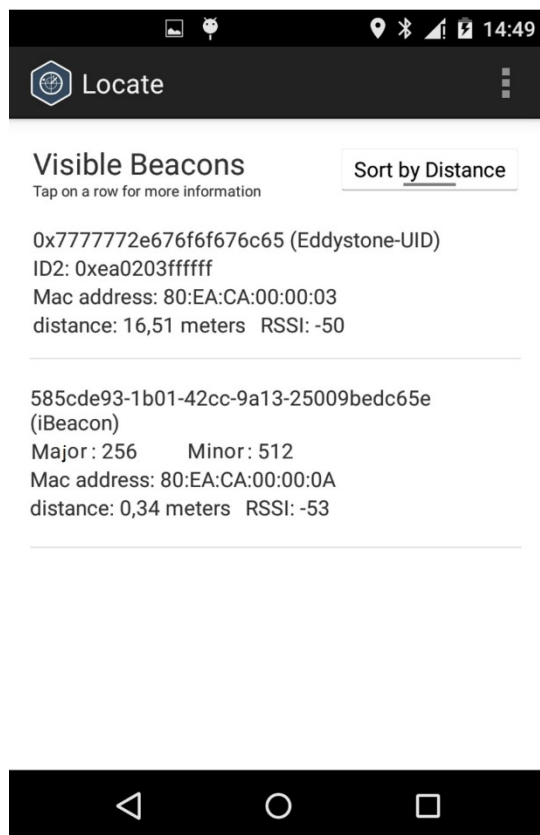


Figure 58: iBeacon Sequence Diagram

7.15 Finding Advertising Devices

A smart device running beacon scanning software will pick up the advertising data and present the received values on the screen. Several beacon monitoring applications for iOS and Android operating systems can be used for the evaluation of the beacon examples of the DA14585 IoT MSK reference application. In this document, Locate app, an Android application, was used for the test purpose (Figure 59).


Figure 59: Locate App Screenshot

In [Figure 59](#), two different beacons are advertising. The first is an Eddystone-UID and the second is an iBeacon. For the iBeacon, the application also parses the Major and Minor values.

8 Memory Footprint and Power Measurements

8.1 Memory Footprint

The SYSRAM footprint of the reference applications in the DA14585 IoT MSK reference design are depicted in [Table 82](#).

Table 82: Memory Footprint

Application	Code	RO Data	RW Data	ZI Initialized
IoT Sensors	66416	6420	416	14764
IoT Sensors without Air Quality library	46124	4556	400	13608
Smart Tag	28332	2952	8	7960
iBeacon	27056	4004	344	15756
Eddystone	30140	3936	748	15524
Alt beacon dynamic	21096	2656	60	15416

Besides the application memory footprint, there is a memory area of ~9.5 Kbytes, depending on the application, reserved for the exchange memory between BLE controller layer and BLE Core.

8.2 Power consumption

Table 83 presents the average power consumption of the most commonly used operation modes of the reference applications of DA14585 IoT MSK.

Table 83: Power Consumption

Mode	Current (avg)
IoT Sensors	
Advertising: Interval at 100 mSec & LED blink	313.596 uA
Idle (waiting for motion to advertise)	20.679 uA
All sensors: Accelerometer & Gyro at 100Hz Sensor Fusion & Magneto 10Hz Temperature, Humidity, Pressure, and Air quality at 0.3Hz Ambient Light & proximity at 0.5Hz	2.008 uA
Motion sensors: Accelerometer & Gyro at 100Hz Sensor Fusion & Magneto 10Hz	1371.032 uA
Environmental sensor Temperature, Humidity, Pressure, and Air quality at 0.3Hz	1054.313 uA
Optical sensor Ambient Light & proximity at 0.5Hz	488.423 uA
Smart Tag	
Advertising	295.977 uA
Connected	158.757 uA
Beacons	
iBeacon Advertising	37.17 uA
iBeacon Connected	124.148 uA
Eddystone Advertising: Environmental sampling at 0.25Hz	49.198 uA
Eddystone Connected Environmental sampling at 0.25Hz	138.226 uA
Altbeacon Advertising	41.146 uA

9 IoT Cloud Applications

9.1 Introduction

This section contains all essential information for end users to make full use of the mobile-side and cloud-side functionalities provided by the five target IoT applications offered together with DA14585 IoT MSK. This section describes the five available IoT cloud applications (*historical data*, *alerting*, *control*, *asset tracking* and *Alexa voice service*) and step-by-step procedures for their access and use.

The applications and the mobile apps' cloud integration make use of the Yodiwo Cloud Platform, though it is presented through a new Dialog-branded domain called *.dialog-cloud.com.

To enable the cloud component, visit the new Cloud section in DA14585 IoT MSK's native Android and iOS mobile apps. There, a link is automatically established between the mobile apps', the new cloud layer, and the Yodiwo Cloud Platform. This link is unique and securely tied to each user.

Multiple apps (and by extension, phones) can be paired to the same user account, using a simple mechanism.

Upon account creation, users also gain access to their own web application in dialog-cloud.com. From there they can:

- View their collected data
- Setup and manage rules that will automatically be relayed to their EK via their paired mobile apps
- Administer their accounts
- Manage their EK devices

All data and actions performed in this web application are completely personal to users and their EK(s) through the issue of a universally unique API Key. For convenience, users may choose to have this API Key and the web application's URI sent to an email of their choice.

9.2 User and Device Management

9.2.1 Users

Multiple mobile apps can be paired to the same account following the procedure below:

1. Open the cloud application and go to the '**Users**' page under the '**Management**' menu. Now, users can generate a token by pressing "**GENERATE TOKEN**" and then "**APPLY**" to update the Yodiwo service.

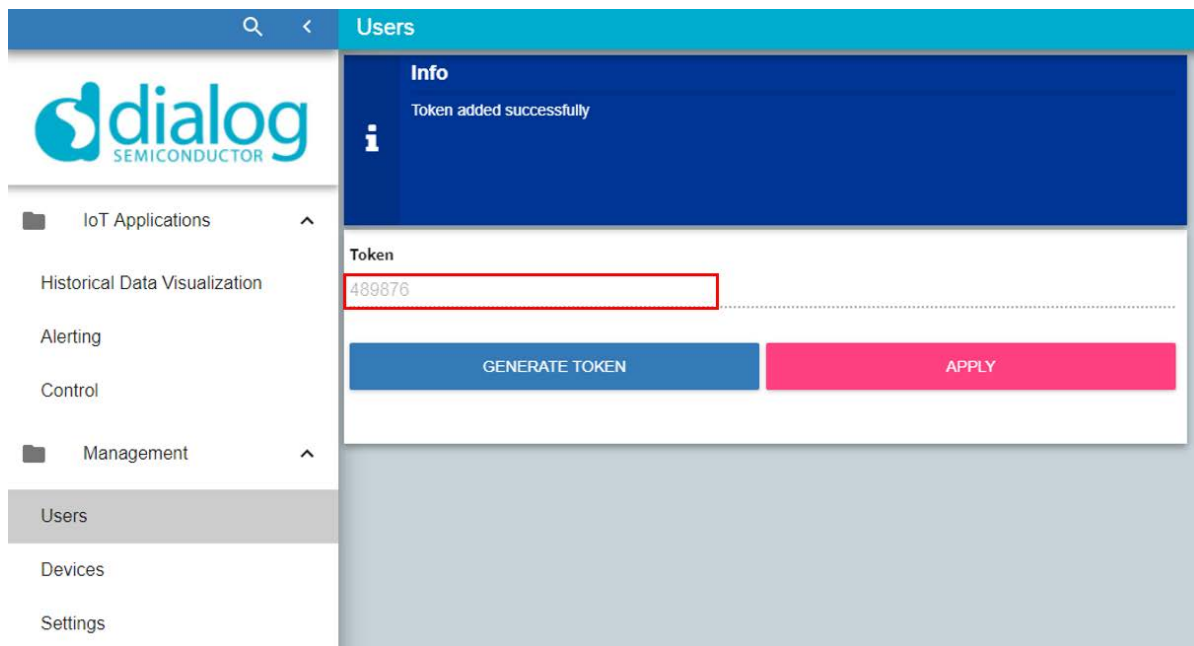


Figure 60: User Management

- Open the IoT Sensors mobile app in Android/iOS and go to "**Cloud Settings**" page under the **Settings** menu. By pressing "**Enable Cloud**", a pop-up window appears, asking users to create a new account or to add an existing account. If users want to pair this mobile app with an existing account, type in the token generated at the cloud application and press "**ADD TO AN EXISTING ACCOUNT**".

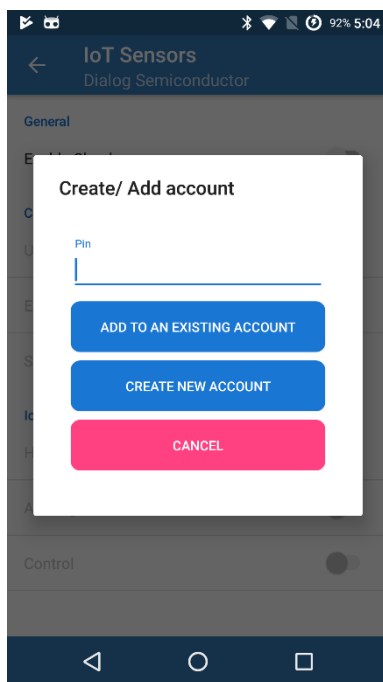


Figure 61: Create Account or Pairing with an Existing Account

9.2.2 Devices

Using the cloud application, users are able to manage their devices. Open the **"Devices"** tab under the **"Management"** menu, and all the devices are shown at the Devices table. Users can edit/delete any of their devices. Figure 62 shows that users can edit the name and description fields and update them by clicking on them.

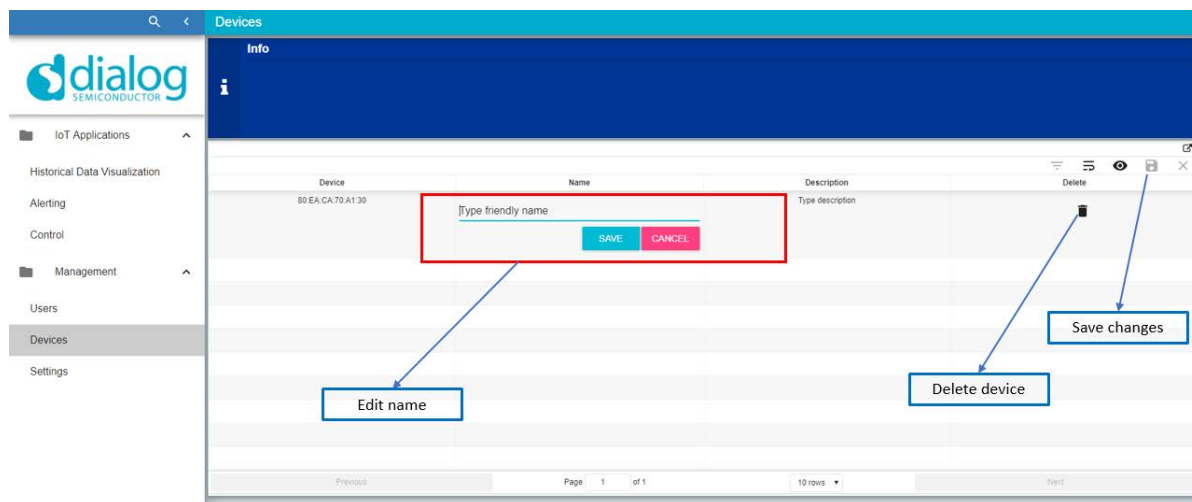


Figure 62: Device Management

9.3 Historical Data

Through the 'Historical Data' application, users can view the available data received by the devices, filtered by date range, sensor type (for example, temperature, humidity, and pressure), and device.

9.3.1 Cloud Application

To view the received data of your devices (Figure 63), open the cloud application and go to the **'Historical Data Visualization'** page under the **'IoT Applications'** folder. Now, you can set your filtering criteria and the available data will be displayed by pressing the **"APPLY"** button.

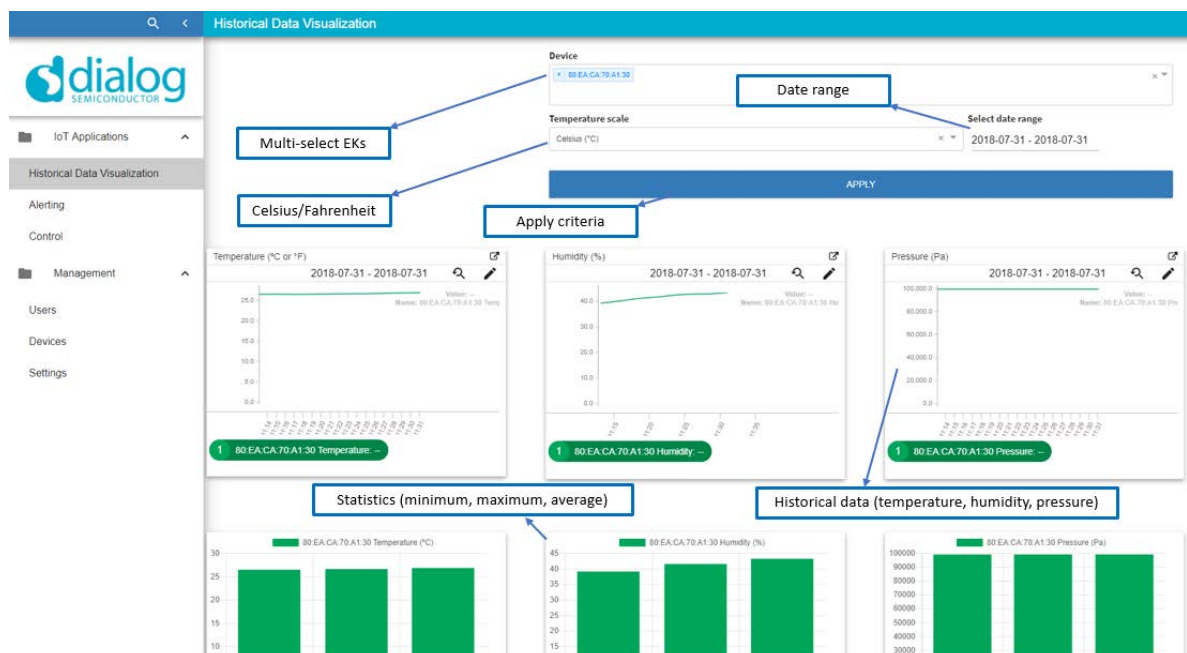


Figure 63: Historical Data: Cloud application

9.3.2 Mobile Application

Users can also view the historical data using the mobile application (Figure 64). To access your historical data, select the **'Historical Data'** tab from the navigation bar, first from the left. Then, set your filtering criteria and press the **'APPLY'** button to display the available data.

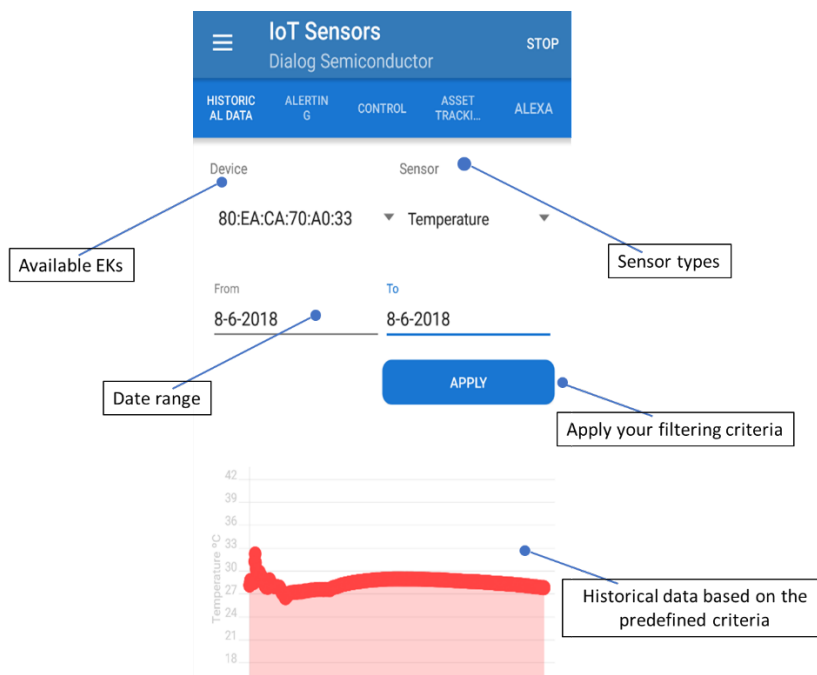


Figure 64: Historical Data: Mobile Application

9.4 Alerting

Using the IoT Alerting Application, users can configure a set of rules, which are triggered when the specified condition based on the values from the EK's sensors is met. Once a rule is triggered, an e-mail will be sent to the address specified by the user in the respective rule. The IoT Alerting application supports the following sensors in DA14585 IoT MSK:

- Temperature
- Humidity
- Pressure
- Air Quality
- Ambient Light

9.4.1 Cloud Application

To add/update an alerting rule (Figure 65), open the cloud application and go to the '**Alerting**' page under the '**IoT Applications**' folder. There a form for adding/updating rules is presented. To add a rule, first fill in the fields of the form and then press the '**ADD/UPDATE**' button to save the rule. Users can also enable/disable, delete, and/or edit an already added rule using the table at the bottom of the page.

A maximum of 10 alerting rules per EK can be added.

The screenshot shows the 'Alerting' page in the cloud application. The interface includes a sidebar with navigation options: IoT Applications, Historical Data Visualization, Alerting (selected), Control, Management, Users, Devices, and Settings. The main content area has a header 'Alerting' and a sub-header 'Add Alerting Rule for the target device'. Below this is a form with fields for 'Name' (Type name), 'Target Device' (Select device), 'IF sensor' (Select sensor), 'Becomes' (Select operator), 'Value' (Type value), and 'THEN send email to' (Type email). There are 'CLEAR FORM' and 'ADD/UPDATE RULE' buttons. Below the form is an 'Info' section and a table titled 'RETRIEVE ALL RULES'. The table has columns: Name, Target Device, Rule description, Status, and Delete. A row is shown with Name 'testandroid', Target Device '80 EA-CA-70 A1 30', Rule description 'If Temperature is > 15.0 then send email to', Status checked, and a Delete button. Annotations with arrows point to various parts of the interface: 'Form for adding/updating rules' points to the form fields; 'Define a name for your rule' points to the 'Name' field; 'Select target EK' points to the 'Target Device' dropdown; 'Apply' points to the 'ADD/UPDATE RULE' button; 'Define parameters' points to the 'IF sensor', 'Becomes', and 'Value' fields; 'Rules list' points to the table; 'Log of performed actions' points to the 'Info' section; 'Enable/Disable' points to the 'Status' column; 'Delete' points to the 'Delete' column; and 'Edit' points to the 'Edit' icon in the table row.

Figure 65: Alerting: Cloud Application

9.4.2 Mobile Application

An alerting rule can also be added from the mobile application (Figure 66). To add a new rule, select the '**Alerting**' tab from the navigation bar (second from the left), fill in the appropriate fields of the form, and press the '**APPLY RULE**' button. Users can also use the '**SYNC**' button to see the number of active rules.

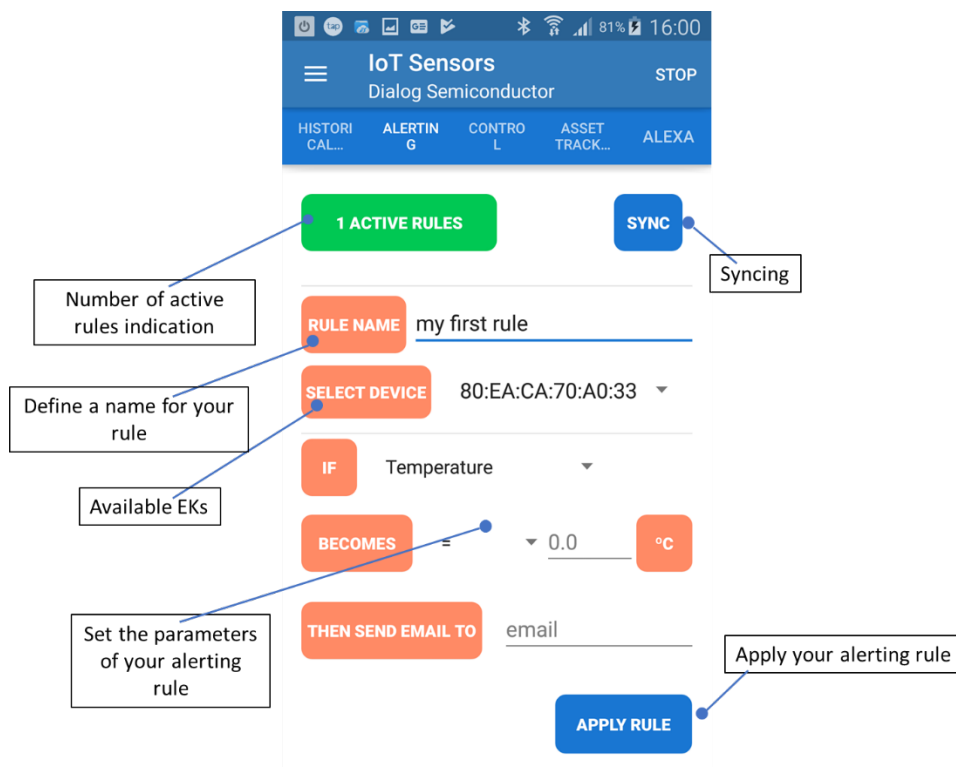


Figure 66: Alerting: Mobile application

9.5 Control

Using the IoT Control Application, users can configure a set of cloud-triggered rules. Two types of cloud triggers are supported:

- Weather
- Foreign Exchange Rates

Once a rule is triggered, the target EK will be driven with the specified actuation action. The supported actuation types are:

- Switch LED on
- Switch LED off

9.5.1 Cloud Application

To use the "control" in Cloud application (Figure 67), open the web app and go to the '**Control**' page under the '**IoT Applications**' folder. To add a new rule; first fill in the fields of the form and then press the '**ADD/UPDATE**' button to save the rule. Users can also enable/disable, delete, and/or edit an already added rule using the table at the bottom of the page.

- For the cloud trigger "Weather", only the value "Temperature" is considered valid in the 'IF' field.
- For the cloud trigger Forex, the 'IF' field should be filled in using one of the major forex symbols:
 - EURUSD, USDJPY, GBPUSD, USDCHE, EURGBP,
 - EURJPY, EURCHF, AUDUSD, USDCAD, NZDUSD
 – The 'In' field will be ignored for the case of FOREX rules.

A maximum of 10 control rules per DA14585 IoT MSK can be added.

Form for adding/updating rules

Define a name for your rule

Select target EK

Control rule type

Apply

dialog SEMICONDUCTOR

IoT Applications

Historical Data Visualization

Alerting

Control

Management

Users

Devices

Settings

Name

Type name ...

Target Device

Select device ...

Cloud Trigger

Select ...

IF

Type parameter ...

In

Type city ...

Becomes

Select operator ...

Value

Type value ...

THEN

Select ...

Off/On

CLEAR FORM

ADD/UPDATE RULE

Info

RETRIEVE ALL RULES

Name	Target Device	Rule Description	Status	Delete	Edit
katsopoulos1	80 EA CA:70 AD:32	If Weather:Temperature in athens:greece becomes > 30 then switch led Off	☒		
test	80 EA CA:70 A1:30	If Weather:Temperature in patras becomes > 20 then switch led On	☐		

Log of performed actions

Enable/Disable

Delete

Edit

Figure 67: Control: Cloud Application

9.5.2 Mobile application

A control rule can also be added from the mobile application (Figure 68). To add a new rule, select the '**Control**' tab from the navigation bar (third from the left), fill in the appropriate fields of the form, and press the '**APPLY RULE**' button. Users can also use the '**SYNC**' button to see the number of active rules.

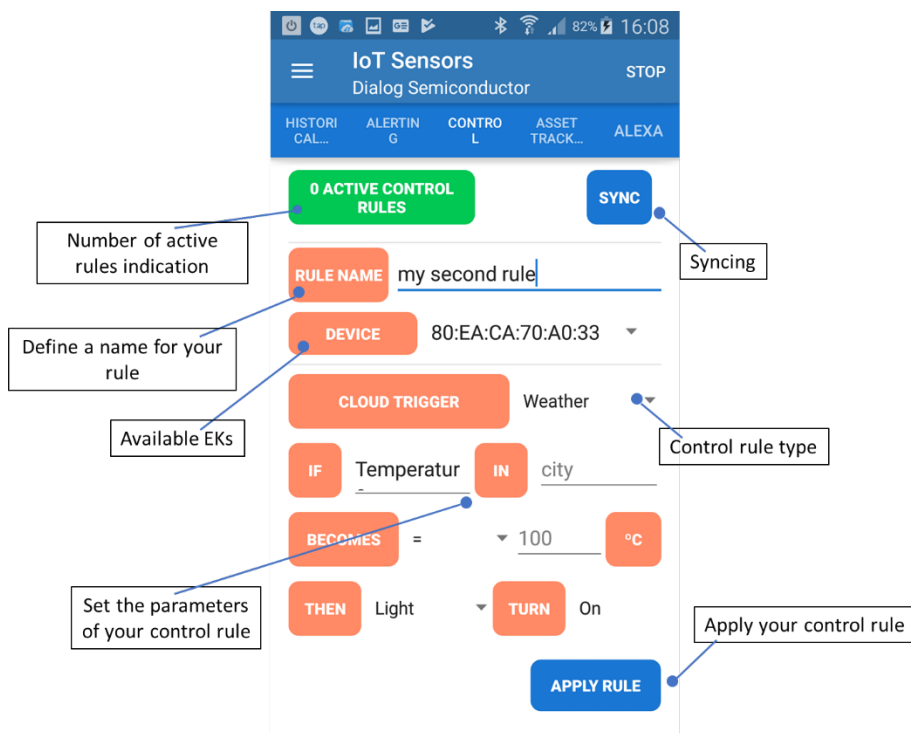


Figure 68: Control: Mobile Application

9.6 Amazon Alexa

Alexa is Amazon's voice control system. She listens to users' order and carries them out, at least simple ones, like dimming lights or playing music tracks.

9.6.1 Skill Activation

To activate the Alexa voice service, enable the **"Dialog Evaluation Kit Skill"** public skill using one of the two available options:

- via a browser
- via Amazon's mobile application **"Amazon Alexa"**.

Please follow the instructions below to activate the public skill.

- Via browser
 1. Go to this link: <https://www.amazon.com/b?ie=UTF8&node=13727921011>
 2. Login with your amazon account
 3. Search for "Dialog Evaluation Kit Skill"
 4. Select and enable the skill (Figure 69)
 5. Give your consent for the application to have access to your profile email

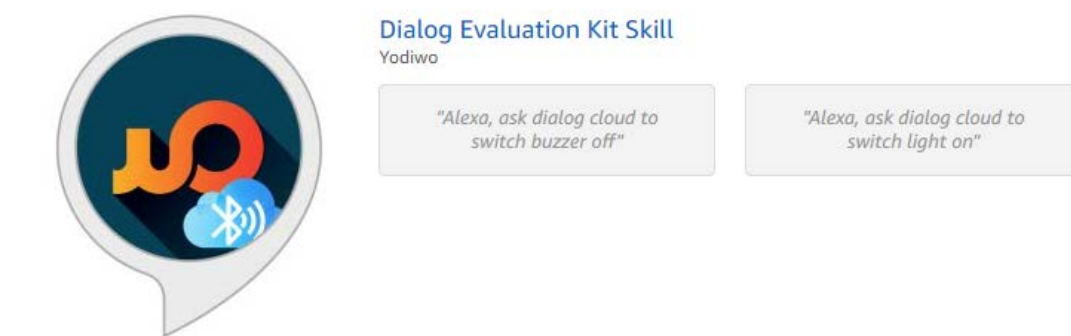


Figure 69: Dialog Evaluation Kit Skill for Alexa

- Via mobile application
 1. Follow the instructions provided by the "Amazon Alexa" application
 2. Login with your amazon account
 3. Go to the "Skills" menu
 4. Search for "Dialog Evaluation Kit Skill"
 5. Select and enable the skill
 6. Give your consent for the application to have access to your profile emailOnce the public skill is successfully activated, [Figure 70](#) is displayed.

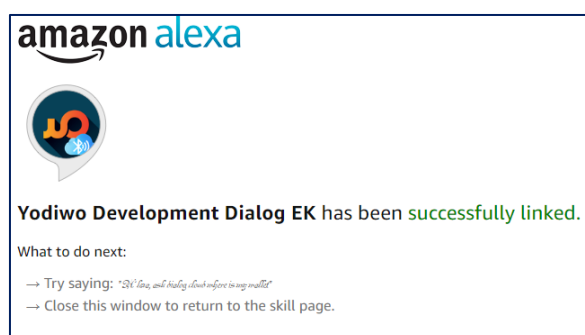


Figure 70: Alexa: Skill Activation Succeeded Screen

9.6.2 How to Use

On Amazon's mobile application, go to the 'Cloud' screen and navigate to the 'Alexa' tab. For the first time using Alexa ([Figure 71](#)), users will be redirected to the Amazon portal to login with their Amazon accounts. Then, users will be asked to give consent for the application to have access to their profile emails. This is mandatory for using Yodiwo cloud and mobile applications.

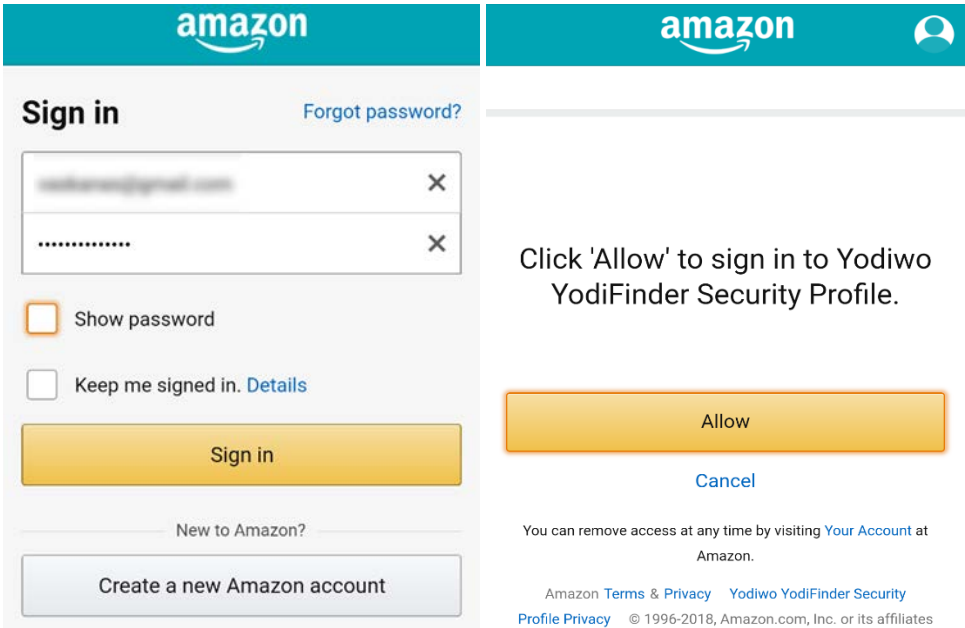


Figure 71: Alexa: Sign in Procedure

After signing in to Amazon's mobile application, [Figure 72](#) is displayed. Press the '**ASK ALEXA**' button to start recording your request (for example, "Alexa, what time is it?") and press it again to send it.



Figure 72: Alexa Mobile Application Main Screen

The currently available functional requests are the ones supported by the IoT Control Application:

- Alexa, ask dialog cloud to switch light on
- Alexa, ask dialog cloud to switch light off

9.7 IFTTT

9.7.1 Create an IFTTT Applet

1. Go to <https://ifttt.com/> and sign up for account (Figure 73).

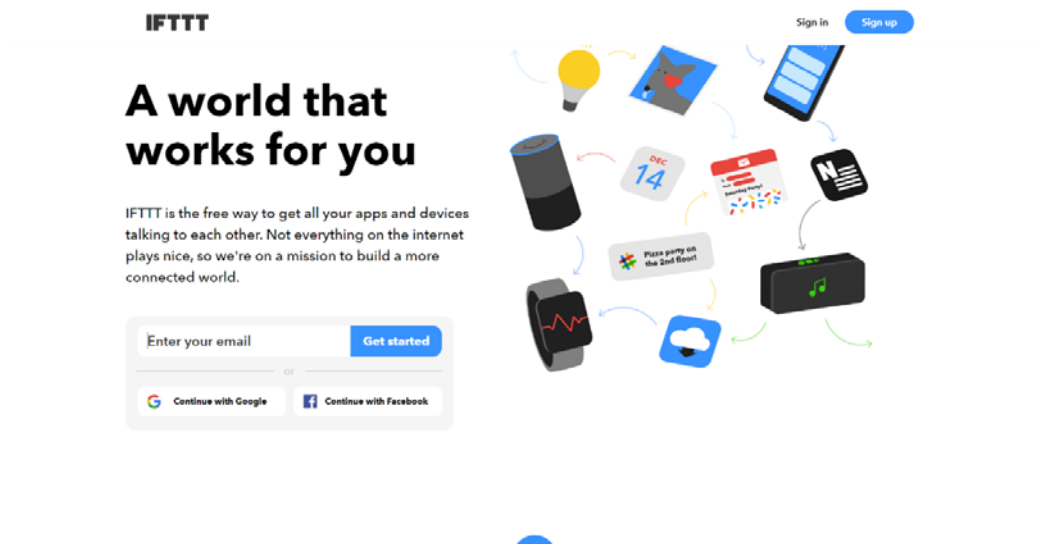


Figure 73: IFTTT Signup Page

2. After logging in to IFTTT, click on 'My Applets' at the top menu and then click on **New Applet** button (Figure 74).

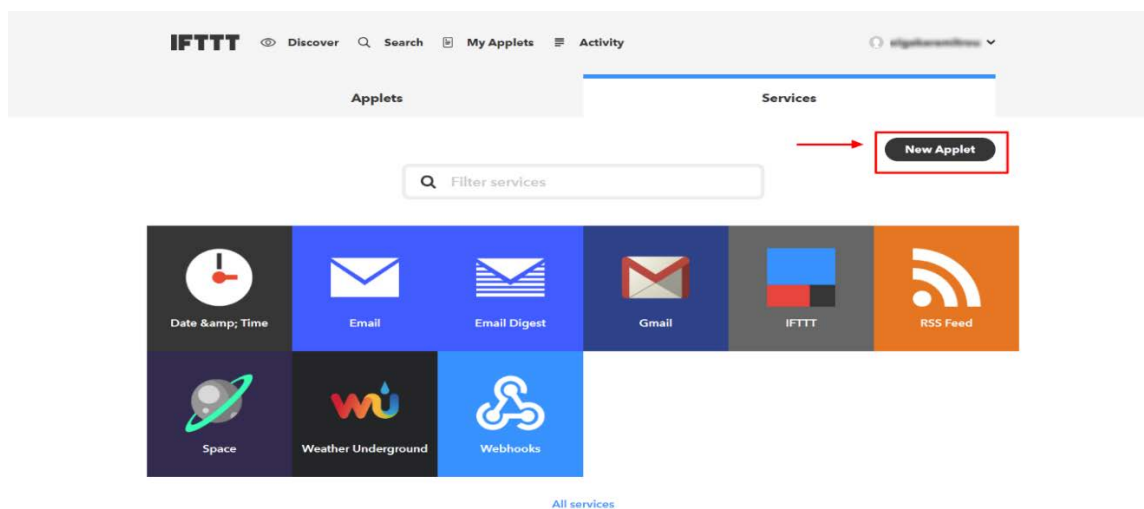
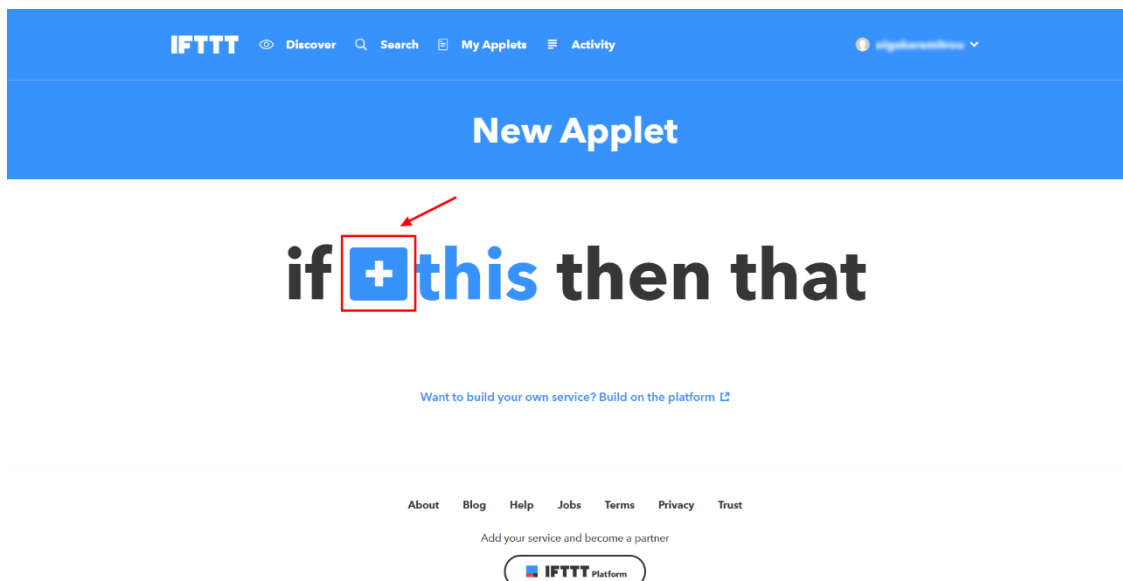
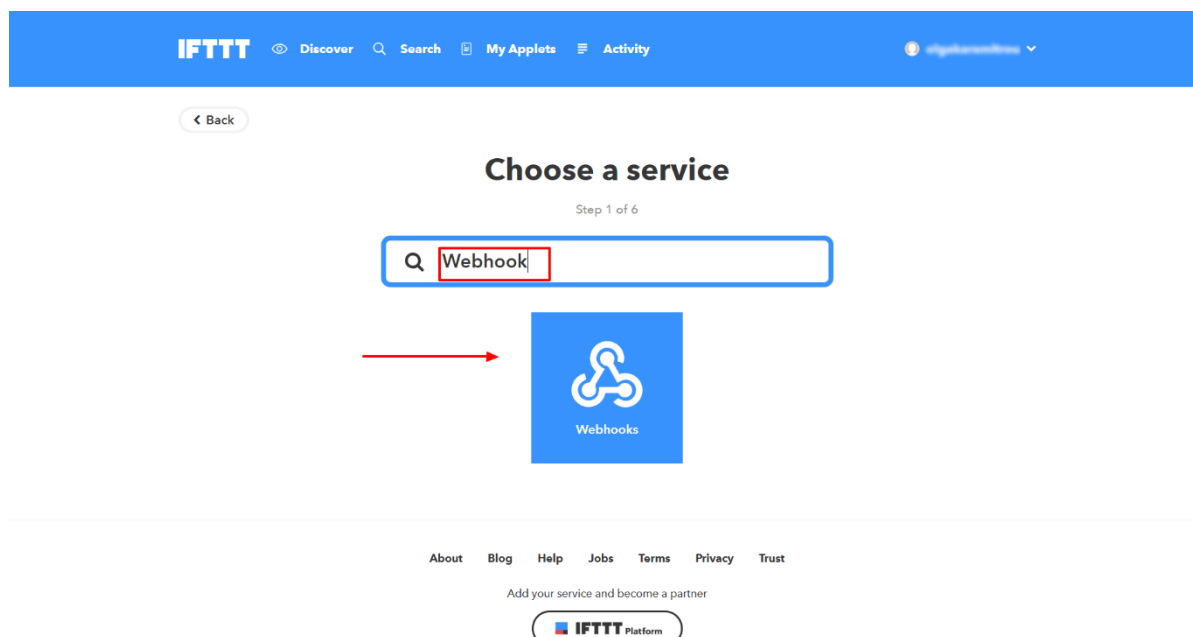


Figure 74: New Applet

3. On the New Applet screen, make an applet as shown in Figure 75.

**Figure 75: Make an IFTTT Applet**

4. Type in "**Webhooks**" in the search field and choose the Webhooks service (Figure 76).

**Figure 76: Make an IFTTT Applet Step 1**

5. On the 'Choose Trigger' screen, click on "Receive a web request" (Figure 77).

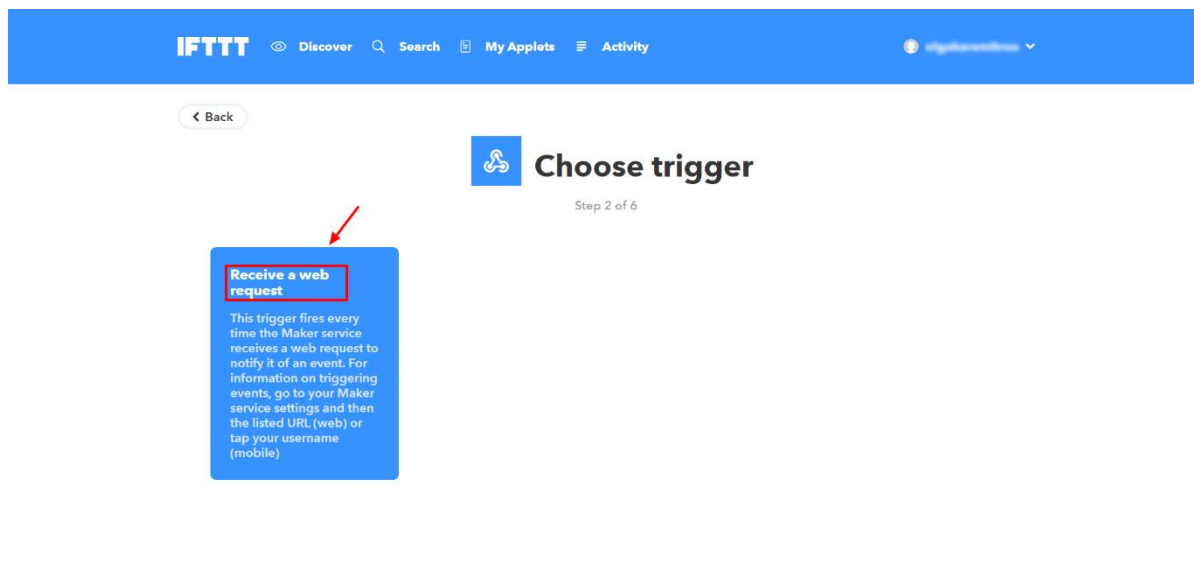


Figure 77: Make an IFTTT Applet Step 2

- On the 'Complete trigger fields' screen, type in one of the supported event names and click the 'Create trigger' button (Figure 78). Dialog IoT Sensors app sends events at four different routes as it supports the following event names: temperature, humidity, pressure, and button.

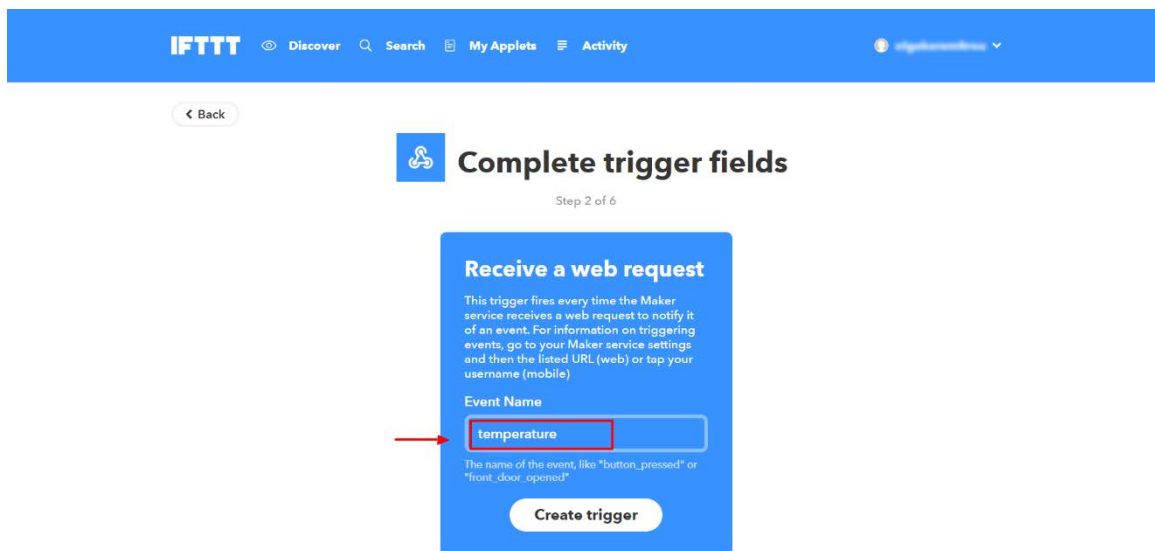


Figure 78: Make an IFTTT Applet, Complete Step 2

- Follow the instruction on Figure 79.

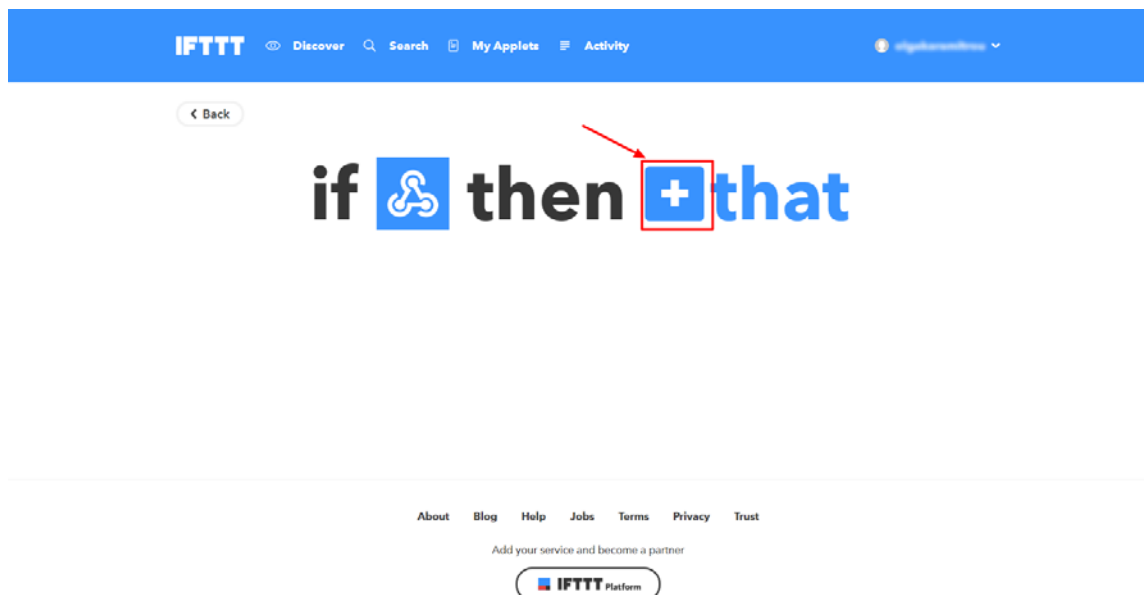


Figure 79: Make an IFTTT Applet, Create That

- On the '**Choose action service**' screen, type in "email" in the search field and click the **Email icon**, as shown below in Figure 80.

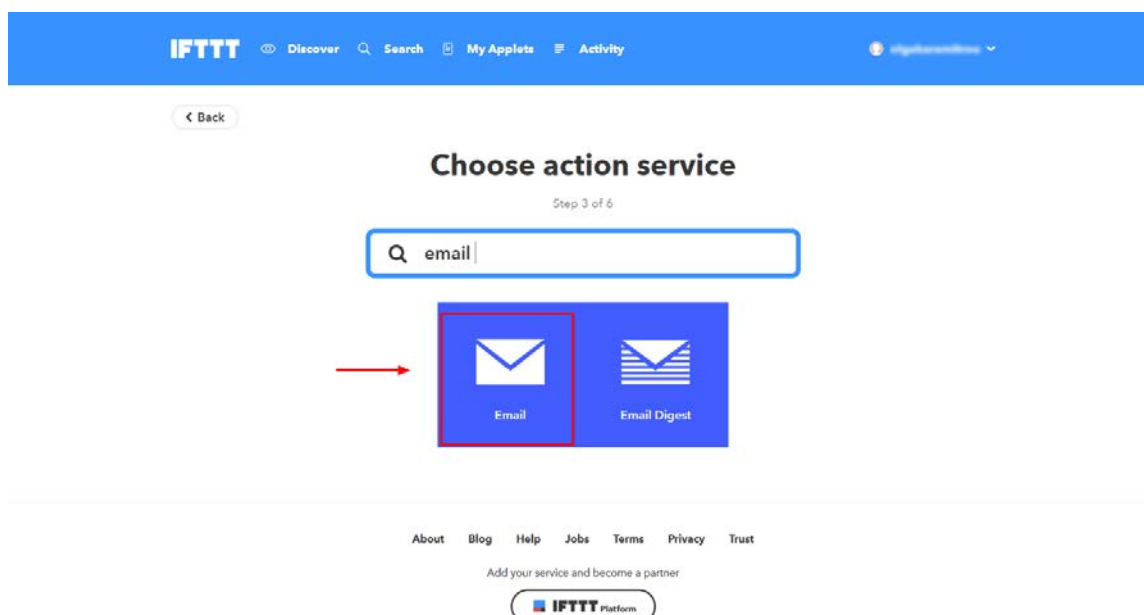


Figure 80: Make an IFTTT Applet, Step 3

- On the '**Choose Action**' screen, click on "**Send me an email**" action (Figure 81).

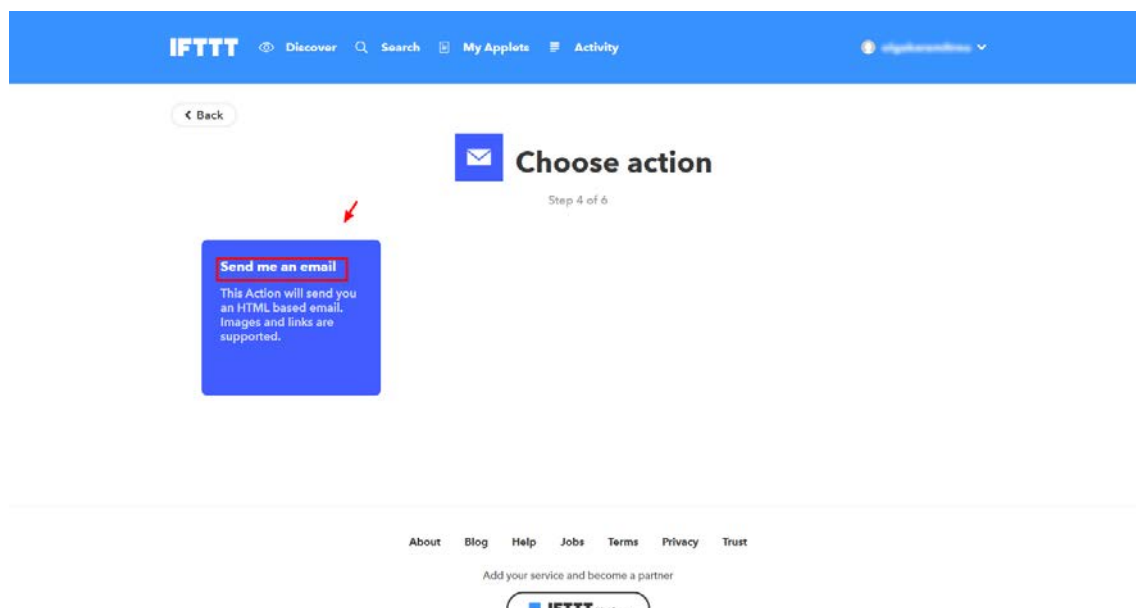


Figure 81: Make an IFTTT Applet, Step 4

10. On the 'Complete action fields' screen, click the "Create action" button (Figure 82).

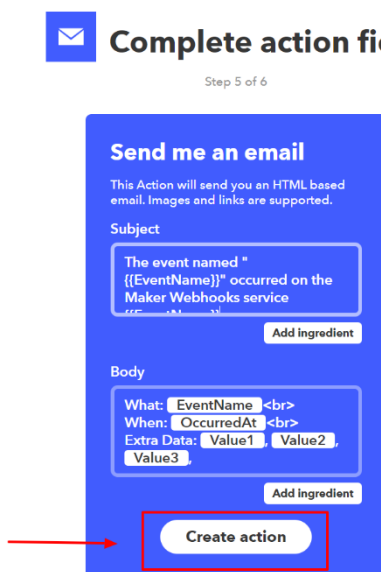


Figure 82: Make an IFTTT Applet Step 5

11. The IFTTT applet has been created. Enable the switch to receive notifications and click 'Finish' button on the 'Review and finish' screen (Figure 83). To receive events for all the supported event names (temperature, humidity, pressure, and button), please repeat the above procedures and add as many applets as the supported events names.

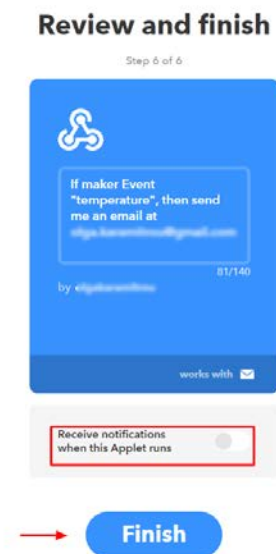


Figure 83: Make an IFTTT applet, Finish

9.7.2 IFTTT Web Hook Key

1. To find your key, go to **'My Applets'**, click **'Services'** tab and then click **Webhooks** (Figure 84).

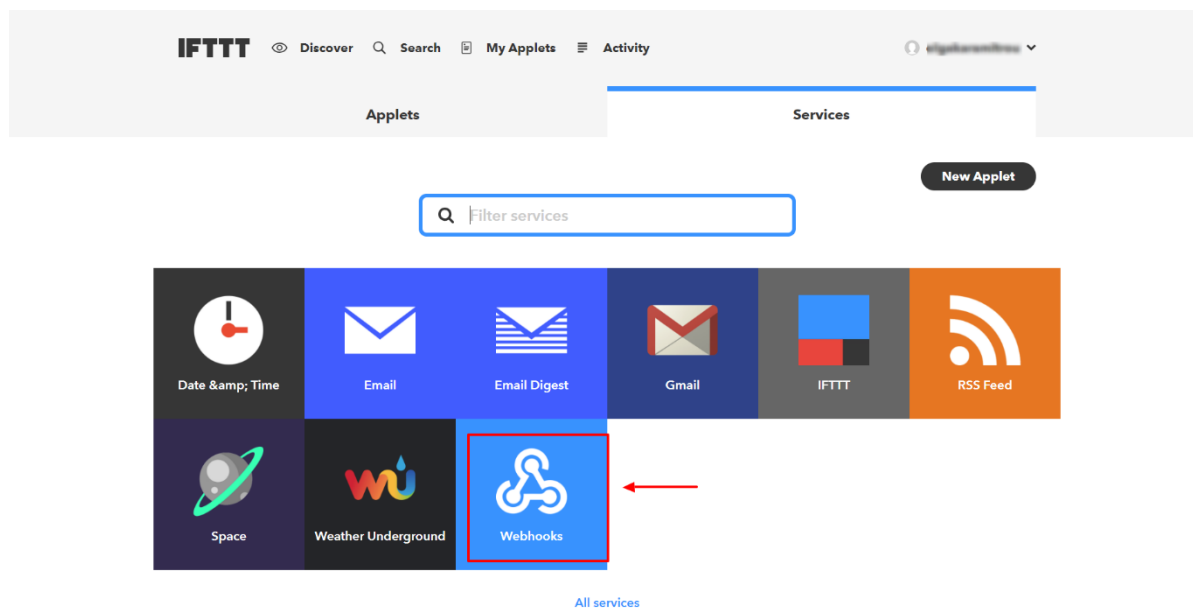


Figure 84: IFTTT Web Hook Key, step 1

2. In the **'Webhooks'** screen click **Documentation** (Figure 85).

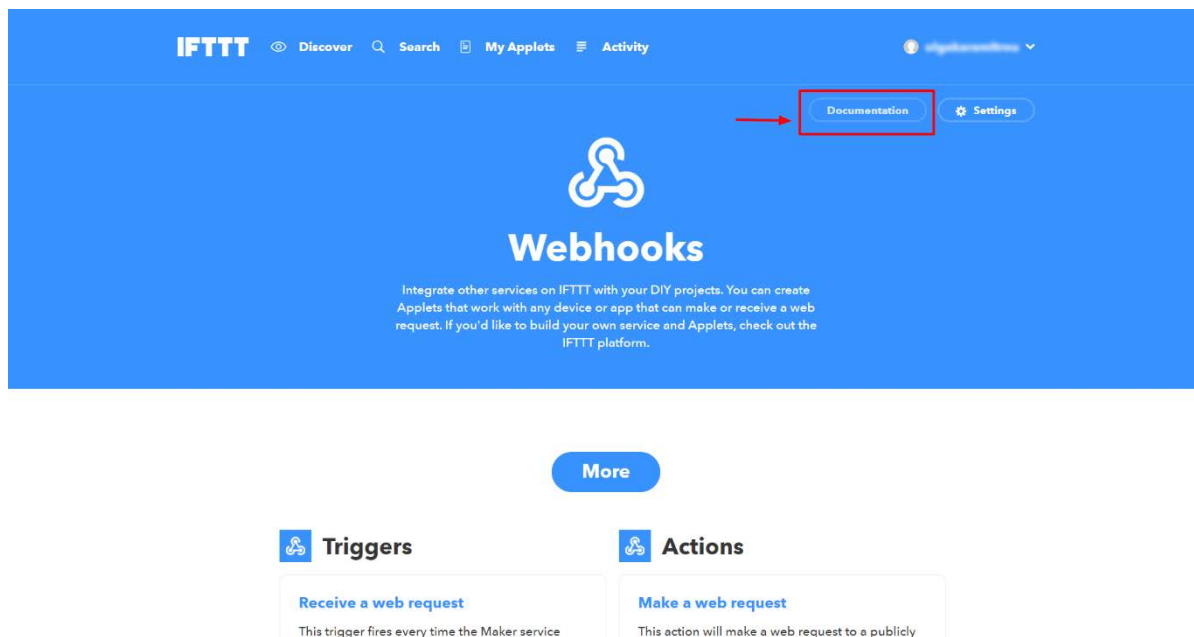


Figure 85: IFTTT Web Hook Key Step 2

- The key will open in a new screen, as shown in Figure 86.



Figure 86: IFTTT Web Hook Key Step 3

9.7.3 Web App - IFTTT Settings

Once users have created a new webhook applet and retrieve the corresponding web hook key, they can update the IFTTT settings in the following step (Figure 87):

- Open the web app
- Go to settings
- Add the key in the '**IFTTT Web Hook Key**' field and click on '**ADD/UPDATE**' button.

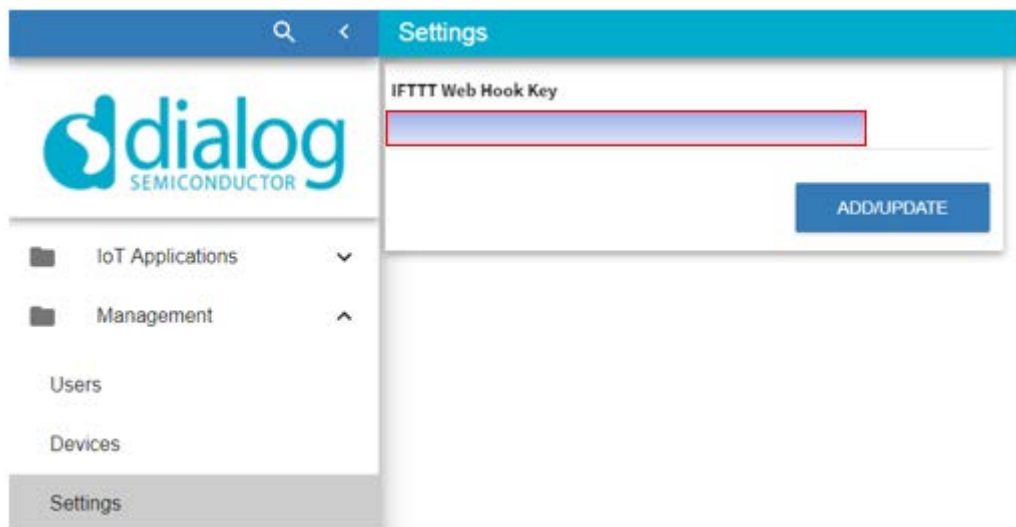


Figure 87: Web App - IFTTT Settings

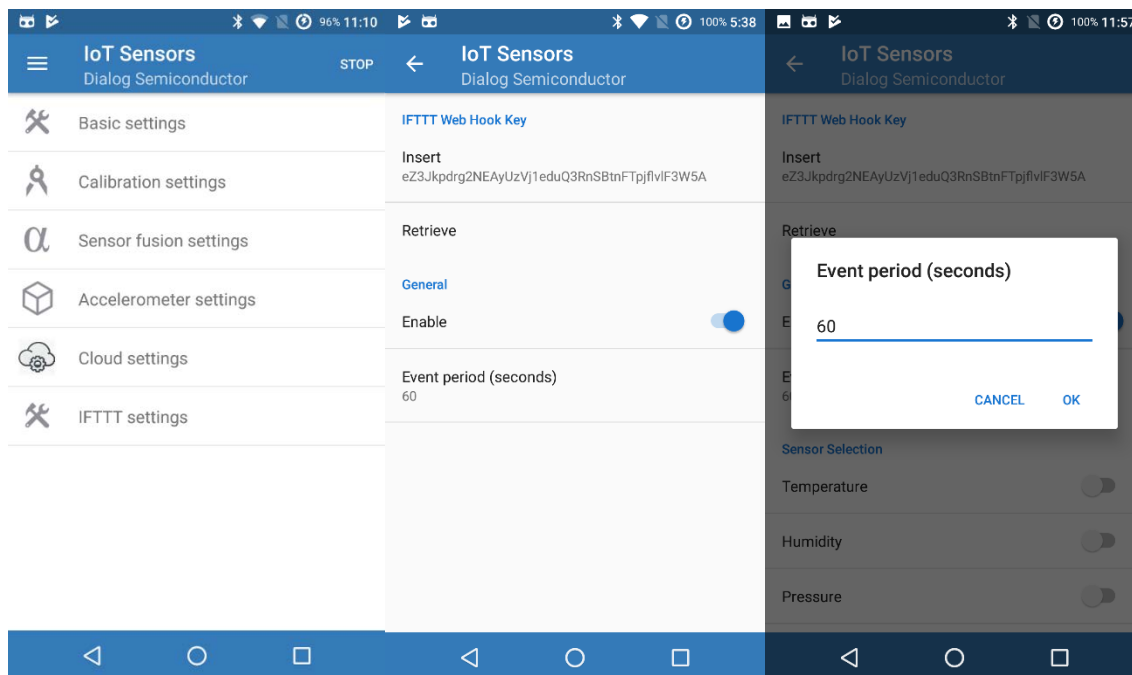
9.7.4 Mobile App: IFTTT Settings

To enable IFTTT and start sending events using the created applet, follow the following steps (Figure 88):

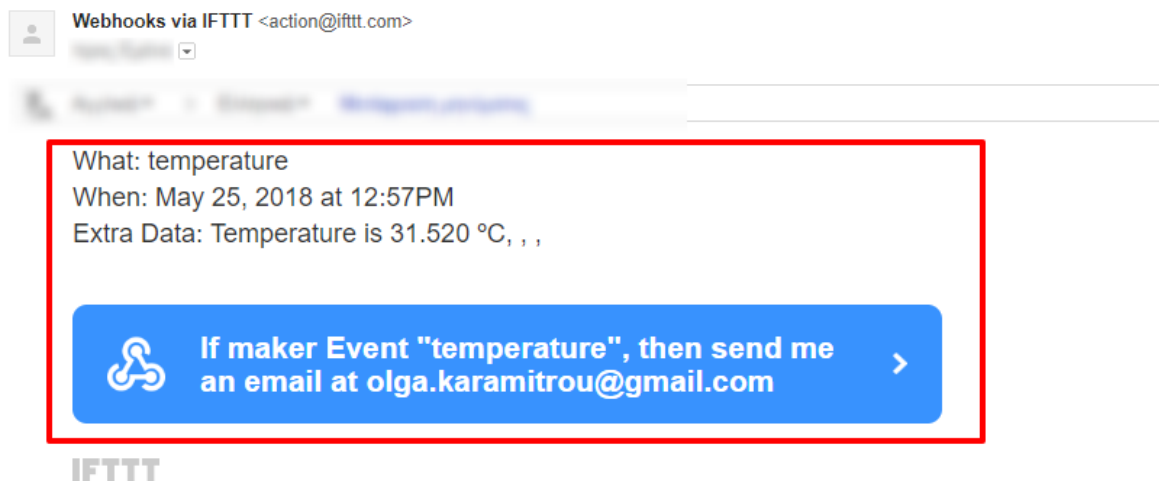
1. Go to '**Settings**' screen in the Dialog IoT Sensors via the side menu
2. Click on '**IFTTT settings**'
3. Insert the web hook API key or retrieve it from cloud. To add a web hook API key from the web application, please refer to section 9.7.2.
4. Enable IFTTT switch in "**General**" and sensors from which you would like to receive events in "**Sensor Selection**".

By default, events will be received every 60 seconds for environmental sensors (temperature, humidity, and pressure). To change the event period, please update the event period from IFTTT settings as shown in Figure 88.

Figure 89 shows the template of the email that users will receive once they have enabled the IFTTT and have created the "temperature" web hook event.


Figure 88: IFTTT Settings - Mobile app

The event named "temperature" occurred on the Maker Webhooks service


Figure 89: Email When "Temperature" Occurred

Appendix A Multi Sensor Kit Boot Sequence

The Multi Sensor Kit (MSK) boot sequence consists of the following stages:

- BootROM sequence (*sections 4.4, 4.4.3, [5]*).
- Secondary Bootloader (*Appendix H, [2]*) is part of the released code inside folder `utilities\secondary bootloader`.
- Application Code.

In order for the application to be initialized correctly, an architecture that uses Secondary Bootloader with mirrored images is used. The secondary bootloader serves four purposes:

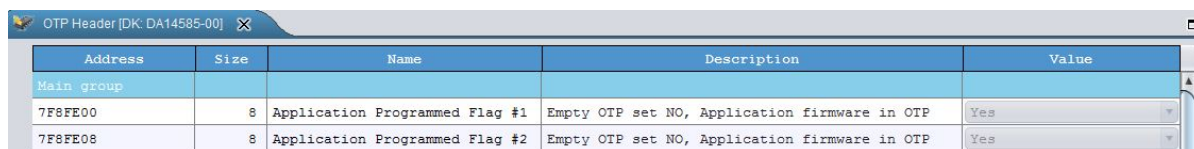
- Initializes the IMU (ICM42605) device to operate in SPI mode.
 - In case offset 0xFE20 in OTP (user area) is not programmed as 0x0, the IMU initialization is skipped.
 - Please be aware that if this bit is set, it cannot be undone and the device may be unable to boot.
- Scans UART during UART boot.
- Selects and copies the most recent application image from the flash memory to SRAM.
- Passes control to application

The Secondary Bootloader is copied from OTP offset 0x0000 to the end of SRAM, so it is crucial to keep the footprint of the Secondary Bootloader as small as possible.

In non-MSK boards, in order to use the Secondary Bootloader, users should follow the following steps:

1. Burn the generated `secondary_bootloader_585.bin` into offset 0x0000 of OTP.
2. Program the two application flags located in offsets 0xFE00 and 0xFE08 of OTP memory using [SmartSnippets](#) as in [Figure 90](#).

Note 22 The MSK boards have the Secondary Bootloader and the OTP application flags written into OTP when shipped.



Address	Size	Name	Description	Value
Main group				
7F8FE00	8	Application Programmed Flag #1	Empty OTP set NO, Application firmware in OTP	Yes
7F8FE08	8	Application Programmed Flag #2	Empty OTP set NO, Application firmware in OTP	Yes

Figure 90: Application Programmed in OTP Flags

Appendix B Memory Map

Figure 91 shows the default memory locations of the different parts of the various images for all projects. The figure also shows that the product header, among other information, contains the offsets of the images and the configuration struct. The offsets of the two images and the configuration struct (in beacon projects) can be modified in the product header thus enabling users to write the images and the struct in those offsets. (The secondary bootloader will “look” for those offset when booting the device. See Appendix D.1). Please pay attention to the distance of the memory locations so as to fit the size of the corresponding parts (product header, application image, and others).

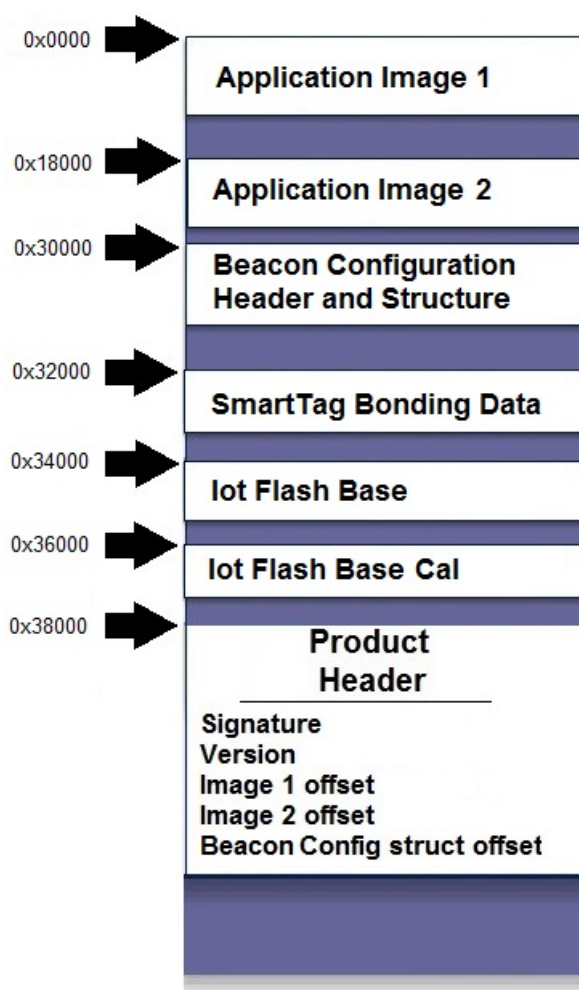


Figure 91: Analyzing a Flash Memory Image

At this point, it is important to note that all these different parts **do not exist all at once**. Table 84 shows which part exists and at which location depending on the application.

Table 84: Parts of the Image Depending on the Application

	All Beacons	Smart Tag	IoT Sensors
Application Image 1	X	X	X
Application Image 2	X	X	X

Beacon Configuration Header and Struct	X		
SmarTag Bonding Data		X	
IoT Flash Base			X
IoT Flash Base Cal			X
Product Header	X	X	X

Appendix C Using the mkimage Application

C.1 mkimage Scripts

To help users, we created mkimage scripts that run the mkimage modes and create the desired multi-image for the "tag" and "IoT sensors" applications or whole images for the "Beacon" applications. The mkimage modes are analyzed in Appendix C.2. The multi-images or whole images are in .bin format and are written in Flash. The multi images contain two alternative images of the application as well as a product header at the end. The whole images are essentially multi images (two images with a product header) but also include a configuration struct for the Beacon.

The available mkimage scripts as well as the various files needed for the scripts to work are located (or should be placed) in ".\utilities\mkimage_utils_scripts" and are shown in Table 85.

Please note that the .hex extension for all images is added by the script automatically and should be:

- C:>iot_image_folder>make_image_beacon.bat altbeacon_dynamic
- C:>iot_image_folder>make_image_iot.bat io585_585
- C:>iot_image_folder>make_image_tag.bat smart_tag_585
- C:>iot_image_folder>make_all_images.bat (no parameters, builds everything if all the needed .hex files exist in the build folder).

Table 85 Available mkimage scripts

Script	Purpose	Needed Files
make_image_beacon.bat	Creates a whole image for the beacon applications	<beacon_project>.hex, dev_conf_struct_default.cfg, user_config_sw_ver.h (struct version), beacon_sw_version.h (Note 23)
make_image_iot.bat	Created a multi-image for the IoT sensors application	<iot_project>.hex, iot_sw_version.h (Note 24)
make_image_tag.bat	Creates a multi-image for the tag application	<tag_project>.hex, tag_sw_version.h (Note 25)
make_all_images.bat	Creates whole images for all beacon applications and multi images for the IoT sensors and tag application by calling all aforementioned bats.	This script only works when the .hex files are named as following: ibeacon_suota_button.hex altbeacon_dynamic.hex, eddy_uid_url_tlm.hex. smart_tag_585.hex, iot585_585.hex and the corresponding sw_version files and beacon configuration files are also present.

clean.bat	Removes all .img and .bin files from the folder	N/A
-----------	---	-----

Note 23 If an alternative .hex file exists, it should go by the name <beacon_project>_1.hex and a file by the name "beacon_sw_version1.h" should be provided.

Note 24 If an alternative .hex file exists, it should go by the name <iot_project>_1.hex.

Note 25 If an alternative .hex file exists it should go by the name <tag_project>_1.hex.

C.2 mkimage modes

The mkimage application has different modes to create desired images.

- **Single:** creates an .img file from the .bin file of the Keil project.
- **Multi:** creates a .bin file from the .bin file of the Keil project that contains two alternative .img files that are needed when using the SUOTA functionality and the product header.
- **Whole_img:** creates an .img file containing two alternative .img files that are needed when using the SUOTA functionality, the config_struct.cfg file, the product header, and optionally the boot loader .bin file.
- **Multi_no_suota:** creates an .img file containing the config_struct.cfg file, the product header, and the .bin file of the Keil project, which is preceded by an AN-B-001 header [16].
- **cfg:** creates a .cfg file containing a device configuration struct preceded by its header.

IMPORTANT NOTE

In order for the mkimage app to work, all needed files should be brought in the mkimage folder where the mkimage.exe will be located after building the mkimage project.

Typing "mkimage" in the command console will show users instructions on the syntax needed to create an .img file for all modes of the application.

C.2.1 mkimage Single

The "mkimage single" mode is used to create an .img file from the .bin file of the Keil project. This image contains the software version and the software version date. The .img files created in this mode are used for manually burning images one-by-one at specific addresses in Flash memory using the [SmartSnippets Studio](#) (see Appendix D.2).

Example: `mkimage single my_project_1.bin sw_version.h img_1.img`

Users should also make a second (different) image file so that during the SUOTA procedure the SUOTA application can find another image to load. Users should either include a different .bin file, a different sw_version.h file, or both.

Example: `mkimage single my_project_2.bin sw_version.h img_2.img`

C.2.2 mkimage Multi

The "mkimage multi" mode is used to create a .bin file from the .bin file of the Keil project. This bin file contains two images created by the "mkimage single" mode and a product header at the end of the file. Optionally, the image can be created for an SPI Flash memory or an EEPROM Flash memory. The .bin files created by this mode are used for burning in Flash memory using the [SmartSnippets Studio](#) (see Appendix D.2).

Example:

```
mkimage multi spi img_1.img 0x0 img2_1.img 0x18000 0x38000 multi_myproject.bin
```

C.2.3 mkimage Whole_img

The "mkimage whole_img" mode is used to create a complete .img file, containing two alternative .img files created by "mkimage single" mode that are needed when using the SUOTA functionality, the config_struct.cfg file and the product header.

Example:

```
mkimage whole_img img_1.img 0x0 img_2.img 0x18000 config.cfg 0x30000 0x38000  
whole_%1.bin
```

The offsets 0x0, 0x18000 and 0x3000 correspond to the file that precedes them: 0x0 is the offset where img_1.img is written and so on. The final offset (in this example 0x38000) is the offset where the product header will be written.

C.2.4 mkimage Multi_no_suota

The "mkimage multi_no_suota" mode is used to create a whole .img file containing the .bin file of the Keil project preceded by an AN-B-001 header and the config_struct.cfg file. Optionally, the image can be created for an SPI Flash memory or an EEPROM Flash memory. The generated image will not include a SUOTA functionality.

In "mkimage multi_no_suota" mode, no ".img" file generated by the "mkimage single" mode is needed.

Example:

```
mkimage multi_no_suota spi out585.bin dev_conf_with_header.cfg 0x30000 0x38000 out.img
```

In this example the out585.bin file (preceded by an AN-B-001 header) is written at address 0x00. 0x38000 refers to the offset where the product header will be written, whereas 0x30000 refers to the offset of the config_struct.cfg file.

C.2.5 mkimage cfg

The "mkimage cfg" mode is used to create a .cfg file containing a device configuration struct preceded by its header. The device configuration struct header also contains a 4-byte CRC which is calculated from the fields of the configuration struct. The application also checks a software version file and includes the version in the header of the corresponding field.

Example: `mkimage cfg dev_conf.bin sw_ver.h dev_conf_with_header.cfg`

Appendix D Flash Programming in MSK Applications

D.1 Basic Information About the MSK Applications

The programmed devices come with the secondary bootloader already burned in the OTP memory. Upon booting, the secondary bootloader is expected to find the product header at address 0x38000, so that information on the signature, version, and as the image offsets can be retrieved. [Appendix B](#) presents the different locations of MSK applications memory map. The following subsections present the formats of the product header, the image header, and the device configuration struct (for beacon projects).

D.1.1 Product Header

Table 86: Product Header Format

Byte #	Field	Description
0, 1	Signature	Product Header signature (0x70, 0x52)
2, 3	Version	Version of the product header
4 to 7	Offset1	Start address of the first image
8 to 11	Offset2	Start address of the second image
12 to 17	RFU	Reserved for future use
18 to 21	Config_offset	Start address of the device config struct (for beacon projects)

D.1.2 Image Header

The Image Header format, which is common for any image created with `mkimage.exe`, is shown in [Table 87](#). The application checks the image header when attempting to write in Flash memory.

Table 87: Image Header Format

Byte	Field	Description
0, 1	Signature	Image Header signature (0x70, 0x51)
2	Valid flag	To be set to 0xAA (STATUS_VALID_IMAGE) at the end of the image burning
3	ImageID	Used to determine which image is newer
4 to 7	Code size	Image size
8 to 11	CRC	Image CRC (Not checked in current version)
12 to 27	Version	Version of the image
28 to 31	Timestamp	Time stamp
32	Encryption	Encryption flag
33 to 63	Reserved	For future use

D.1.3 Beacon Configuration Struct and Configuration Struct Header

The device configuration struct is preceded by the device configuration header in the Flash memory. [Table 88](#) and [Table 89](#) show the Device Configuration Header and the Device Configuration struct, respectively.

Table 88: Beacon Configuration Header

Byte #	Field	Description
0, 1	Signature	Device Config Signature (0x70, 0x53)
2	Valid flag	A value of 0xAA denotes a valid image
3	Number of items	The number of configuration parameters in the configuration data
4 to 7	CRC	CRC of the device configuration struct
8 to 23	Version	Determines the configuration structure type and version
24 to 25	Data size	Size of the configuration data
26	Encryption flag	(Not supported in current version)
27 to 63	Reserved	-

The Beacon Configuration struct appears in Flash memory in the same way as in the code (see section 7.5.1.1).

Table 89: Beacon Configuration Struct Format

Byte #	Field	Description
0 to 15	UUID	Beacon Universally Unique ID
16, 17	Major	Major Value (MSB first)
18, 19	Minor	Minor Value (MSB first)
20, 21	Company_id	Beacon Company id (MSB first)
22, 23	Adv_int	Advertising interval (MSB first)
24	Power	Beacon output power at 1 m
25	beacon type	iBeacon/AltBeacon/Eddystone (not used in provided examples)
27	url prefix	URL prefix for Eddystone - URL
28 to 46	url[19]	The URL preceded by the length of service data and succeeded by the extension (.com, .net and others)
47	TLM_version	TLM version
48	TLM_used	Flag to indicate whether Eddystone-TLM is used or not

D.1.4 Smart Tag Bonding Data, IoT Flash Base, IoT Flash Base Cal

Smart Tag bonding data, IoT Flash base, and IoT Flash base cal are spaces used by the Smart Tag and IoT sensors applications. They are initially blank but are populated by the application for their needs. In that sense, they cannot be configured by users.

D.2 Flash Programming

D.2.1 Burning the Whole Image in Flash Memory

The procedure to make use of the provided mkimage scripts and files utilized by the scripts has been thoroughly explained in Appendix C.1.

With **SmartSnippets Studio**, users can follow the steps below to burn the generated .bin file (multi-image for tag and IoT sensors applications and whole image for beacon applications) (see [Appendix C](#)):

1. Open SmartSnippets Studio > SmartSnippets Toolbox ([Figure 92](#)). Select **JTAG** (above), JTAG adapter (middle panel), a project name (left panel), and chip version (DA14585, right panel). If **SmartSnippets** Toolbox is being run for the first time, first define a new project by the **New** button.

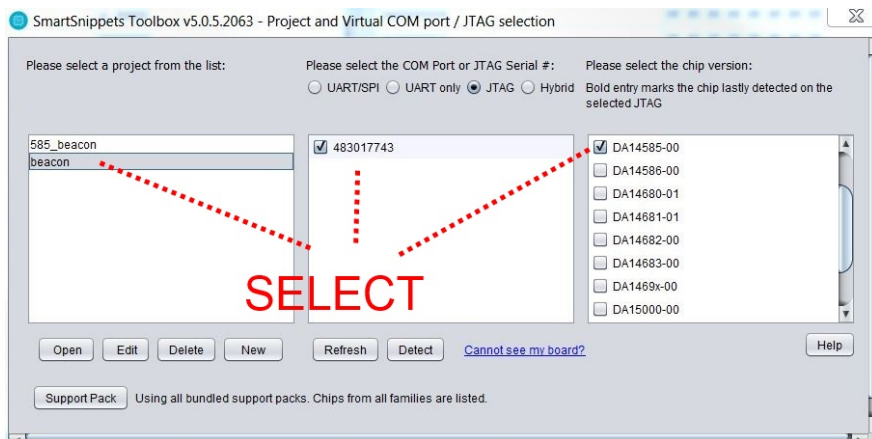


Figure 92: Initial Window to Choose Device and Connection Type

2. Click the **Open** button.
3. From the **Tools > Board Setup** ([Figure 93](#)), make sure that the correct board settings ([Figure 94](#)) are set.



Figure 93: Opening SmartSnippets Board Setup

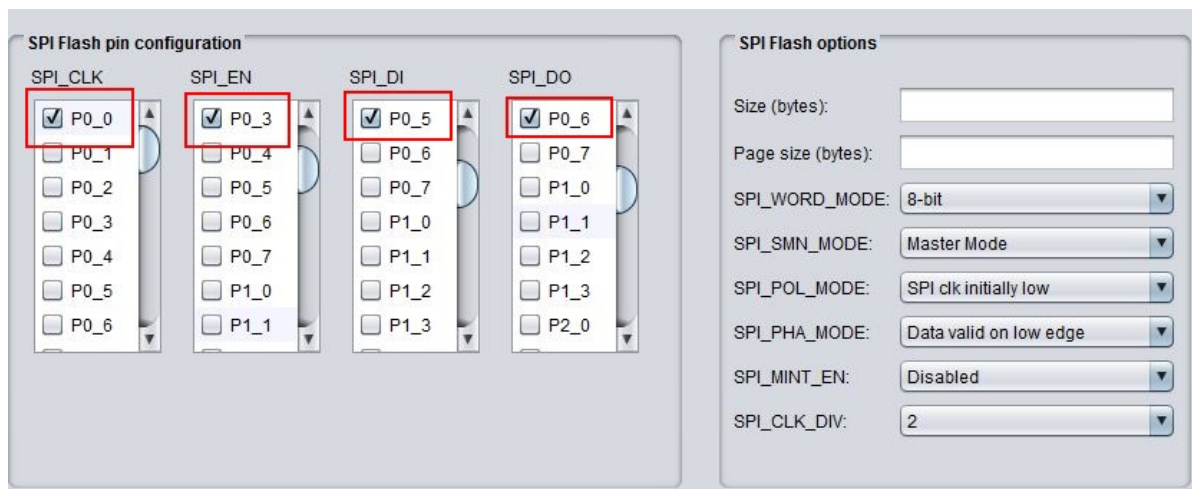


Figure 94: Smart Snippets Board Setup Window

4. Click the button on the left to open the **SPI Flash Programmer** (see Figure 95):

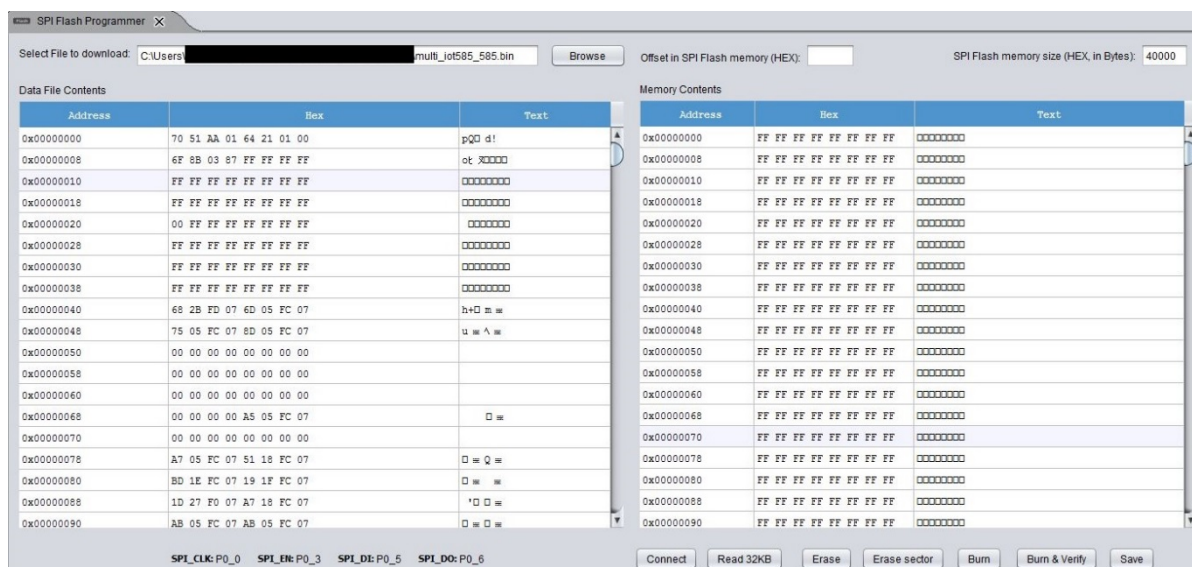


Figure 95: SPI Flash Programmer

5. From **Select File to download**, browse for the <multi/whole_app>.bin file.
6. Insert address 0x00 in the Offset in SPI Flash memory (HEX) field and follow the steps:
 - a. Press **Connect** and wait for confirmation from the Log window;
 - b. Press **Erase** and wait for confirmation from the Log window;
 - c. Press **Burn & Verify** and wait for confirmation from the Log window.

IMPORTANT NOTES

- When burning Flash memory with **SmartSnippets Studio**, click **NO** when the "Do you want SPI Flash memory to be bootable?" window appears.
- In **SmartSnippets Studio**, type (for example) "30000" instead of "0x30000".

- The image is now burnt in flash and by pressing the Reset button on the CIB board, it will start working with the programmed application.

D.2.2 Preparing the Various .img and .bin Files Manually

The following files (Table 90) are needed to program the Flash memory so that a MSK application starts on a button or hard reset.

Table 90: Files Needed or Created During Flash Programming

File Name	Location	Description
<application>.hex	..\..\Keil_5\out_585	.hex file generated by Keil project
<application>.bin	mkimage folder	.bin file generated from .hex file
<sw_version>.h	..\..\sdk\platform\include	.h software version file needed to create .img
<file>.img	mkimage folder	.img file created from the .bin file and the software version file
device_config_all_beacons.txt	mkimage_utils_scripts folder	Contains the format of the device config struct
<dev_conf_struct_default>.cfg	Any (not important)	.cfg file containing a default configuration struct preceded by its header
user_config_sw_ver.h	sw_version folder	.h beacon sw version file used to create the device configuration field (struct+header)

In order to prepare the necessary files, follow to steps below:

- Open the mkimage app folder.
- In the folder, open a cmd window (Shift + Left click > Open command window here).
- If there already is a .bin file of the Keil project, go directly to step 8.**
- Copy and paste the .hex file generated by your Keil project in the mkimage folder.
- Create a .bin file from the .hex file (using the hex2bin.exe utility).
- Create an .img file from the .bin file. Typing "mkimage" in the command console will show instructions on the syntax needed to create an .img file. Example: `mkimage single <generated_bin_file>.bin <version_file>.h <desired_name>.img`.
- When using dev_conf_struct_default.bin to create the <dev_conf_struct_default>.cfg, skip to step 8.g.
- Open [SmartSnippets Studio](#) > SmartSnippets Toolbox. The following dialog ([Figure 96](#)) appears:

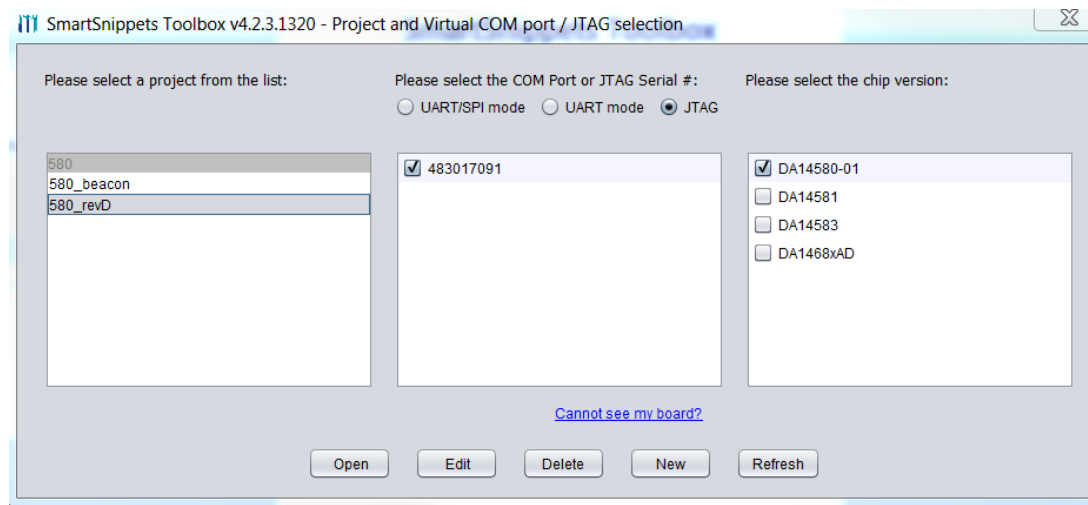


Figure 96: Initial Window to Choose Device and Connection Type

- Choose option **JTAG** and click button **Open**.
- In the next screen, press button  on the left to open the Proprietary Header Programmer. The mkimage folder should contain a device_config_all_beacons.txt file (see Figure 97).

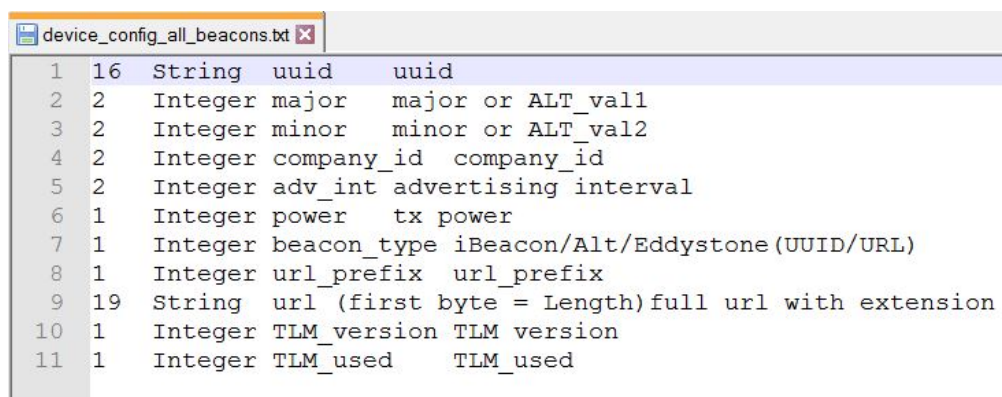


Figure 97: Device Config Struct Format in .txt File

- Browse to the file device_config_all_beacons.txt and open it for editing.
- Enter the desired values at the corresponding blanks (see Figure 98 and Figure 99):

Memory offset (HEX): 1f000

File: ties\tools_binaries\device_config_all_beacons.txt

Memory Type: ☐ EEPROM ☒ SPI Flash

Address	Size (bytes)	Type	Parameter	Description	Value
0x1F000	16	String	uuid	uuid	
0x1F010	2	Integer	major	major or ALT_val1	
0x1F012	2	Integer	minor	minor or ALT_val2	
0x1F014	2	Integer	company_id	company_id	
0x1F016	2	Integer	adv_int	advertising interval	
0x1F018	1	Integer	power	tx power	
0x1F019	1	Integer	beacon_type	iBeacon/Alt/Eddystone (UUID/URL)	
0x1F01A	1	Integer	url_prefix	url_prefix	
0x1F01B	19	String	url	(first byte = Length)full url with extension	
0x1F02E	1	Integer	TLM_version	TLM version	
0x1F02F	1	Integer	TLM_used	TLM_used	

Figure 98: Programming the Various Fields of the Device Configuration Struct

Note 26 Addresses and values are random.

Memory offset (HEX): 1f000

File: ties\tools_binaries\device_config_all_beacons.txt

Memory Type: ☐ EEPROM ☒ SPI Flash

Address	Size (bytes)	Type	Parameter	Description	Value
0x1F000	16	String	uuid	uuid	2380C52068D211E498030800200C9A66
0x1F010	2	Integer	major	major or ALT_val1	0003
0x1F012	2	Integer	minor	minor or ALT_val2	0004
0x1F014	2	Integer	company_id	company_id	4C00
0x1F016	2	Integer	adv_int	advertising interval	E803
0x1F018	1	Integer	power	tx power	C5
0x1F019	1	Integer	beacon_type	iBeacon/Alt/Eddystone (UUID/URL)	00
0x1F01A	1	Integer	url_prefix	url_prefix	00
0x1F01B	19	String	url	(first byte = Length)full url with extension	00000000000000000000E064696173656D6907
0x1F02E	1	Integer	TLM_version	TLM version	00
0x1F02F	1	Integer	TLM_used	TLM_used	00

Figure 99: Creating a Custom Dev_Conf_Struct .bin File

Note 27 Addresses and values are random.

- e. Click the **Export** button at the bottom of the screen to save the .bin file.
- f. To edit the generated .bin files:
 - i. Press button to open the Proprietary Header Programmer.
 - ii. Click the **Import** button at the bottom of the screen and browse for <filename>.bin. The **Value** column will now be populated with the programmed values.
 - iii. Modify the values as required.

- iv. Click button **Export** to save the .bin file.
- g. To create the device configuration file to be burned into Flash memory, use the mkimage application in cfg mode (see Appendix C.2.5). This file includes the device configuration struct preceded by the device configuration struct header.

Appendix E Using the SUOTA Application for Android

This appendix describes how to use Dialog's SUOTA application to update the software programmed in the DA14585 MSK HW reference design. There are two variants of the SUOTA application: one for the Android operating system and one for the iOS. Users can find the SUOTA application by searching for 'Dialog SUOTA' in Google Play Store or Apple App Store. Since these applications have a similar user interface, only the application for Android operating system is described here.

STEP 1: Prepare the MSK Device

If your device is already programmed, skip this step and proceed with STEP 2.

As described in section 4.3, a dual image secondary boot loader needs to be programmed into the OTP memory of the DA14585. Also, the initial software image with the right header needs to be programmed into the DA14585 IoT MSK SPI Flash memory using the Flash Programmer (see [Appendix D](#)).

To verify that this step has been completed successfully, reset the device and check the expected behavior for the desired application:

- Green led blinking for Smart Tag,
- Yellow led blinking for IoT sensors,
- Beacon advertising for beacon examples.

STEP 2: Install and Start the Application on the Android Device

After successful installation, the SUOTA icon appears under the installed applications menu. Click on the icon ([Figure 100](#)) to start the application.

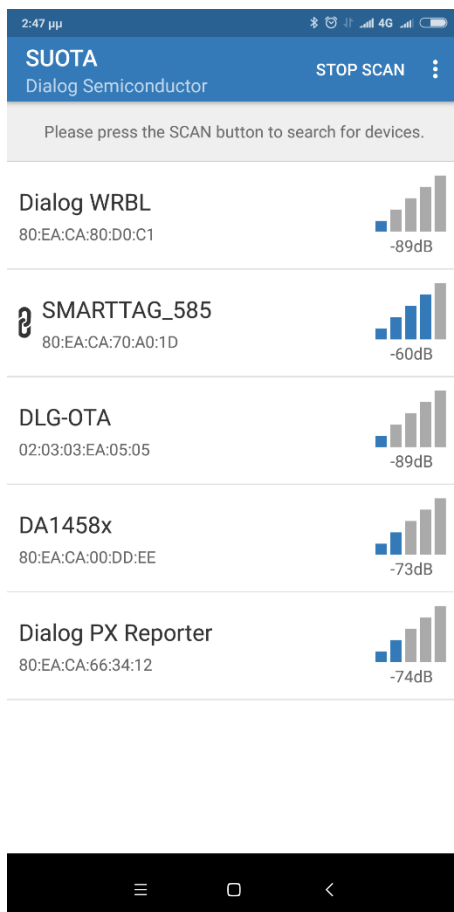


Figure 100: SUOTA App Icon

STEP 4: Initial Menu, Scan for Advertising Devices

Assuming the MSK device is advertising a SUOTA advertising string upon opening the app, the Bluetooth Device address and device name of the device will be displayed as shown in [Figure 101](#).

To re-initiate scanning, just press the “SCAN” button in the top right corner.

**Figure 101: Device Selection Menu****STEP 5: Connect to the Smart Tag Device**

Click on the desired device to connect. Upon successful connection to the device, the DIS information will be displayed on the screen as shown in [Figure 102](#).

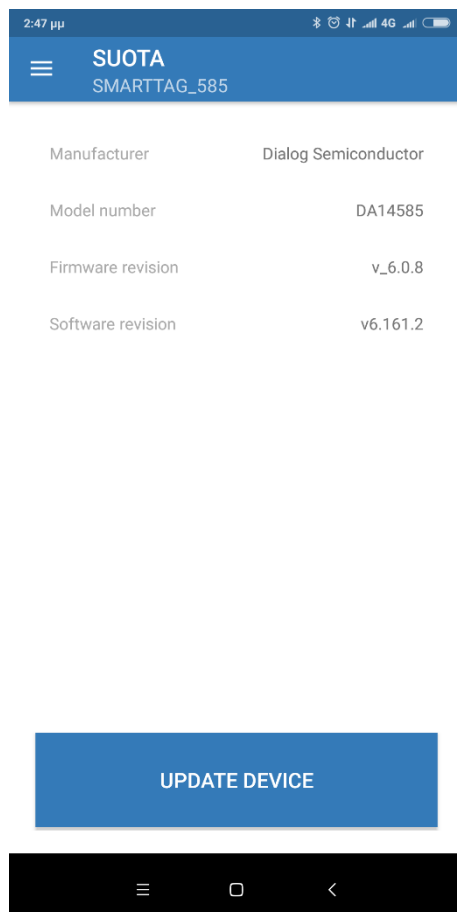


Figure 102: DIS Screen

STEP 6: Update SmartTag Software Image

Click on the **Update device** button and a list of files will appear on the screen. In order for the file to appear in this **File selection** screen, it has to be copied to the SUOTA directory of the Android device. Connect the Android device via USB to the PC where the images have been created and copy the images under the SUOTA directory. An example of the File selection screen is shown in [Figure 103](#).

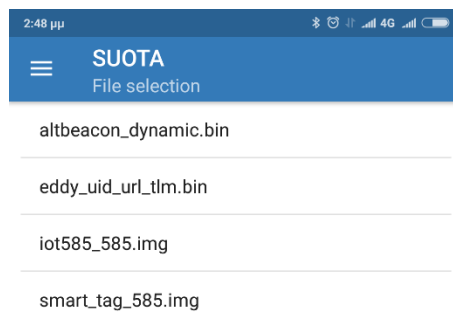
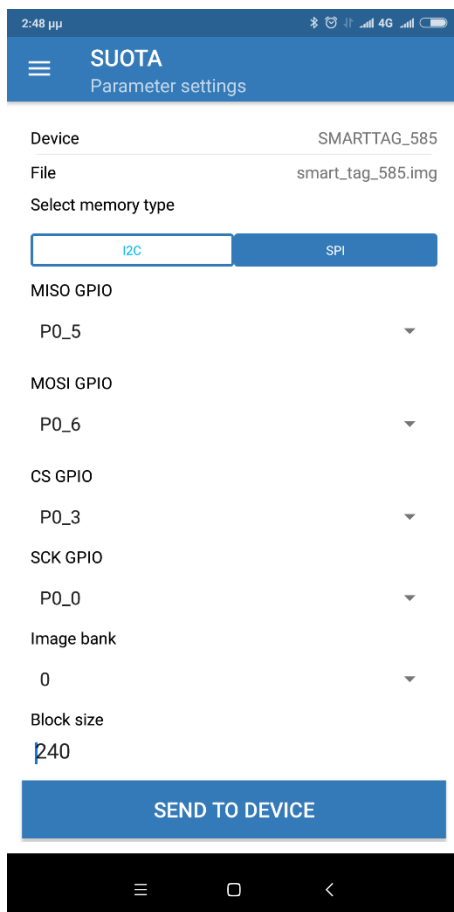


Figure 103: File Selection Screen

STEP 7: Set SUOTA Parameters

Once an image is selected, the **Parameter settings** ([Figure 104](#)) screen appears.


Figure 104: SUOTA Parameter Settings

First set the memory type. The image update procedure is only supported for non-volatile memory types of SPI (Flash memory) and I2C (EEPROM). In this example SPI (Flash) has been selected.

Then select the **Image (memory) bank** (Figure 104):

- 1: Use the first bank with start address as indicated in the Product Header.
- 2: Use the second bank with start address as indicated in the Product Header.
- 0: Burn the image into the bank that holds the oldest image.

Next, define the GPIO pins of the memory device. In the SmartTag device, the SPI Flash GPIO configuration is as follows:

- MISO => P0_5
- MOSI => P0_6
- CS => P0_3
- SCK => P0_0

Finally, scroll down to choose the block size. A few points should be considered when this size is being set.

- The block size must be larger than 64 bytes, which is the size of the image header.
- The block size must be a multiple of 20 bytes, which is the maximum amount of data that can be written at once in the SPOTA_PATCH_DATA characteristic.

- The block size must not be larger than the SRAM buffer in the SUOTA Receiver implementation, which holds the image data received over the BLE link before burning it into the non-volatile memory.

This example uses a block size of 240 bytes.

After all the parameters have been set, users can click on the **Send to device** button at the bottom of the screen.

STEP 8: Reboot the Device

Once the **Send to device** button is clicked, a load screen appears that shows the image data blocks sent over the BLE link (Figure 105). In case an error occurs, a pop up indication will inform the user.

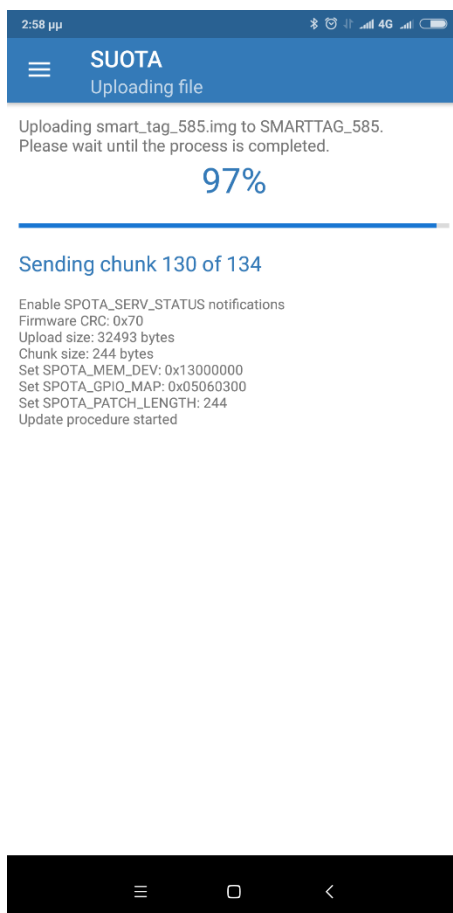


Figure 105: SUOTA Uploading Screen

When no error occurs and the SmartTag device has received and programmed the image successfully, the screen in Figure 106 will appear, asking the user to reboot the SmartTag device.

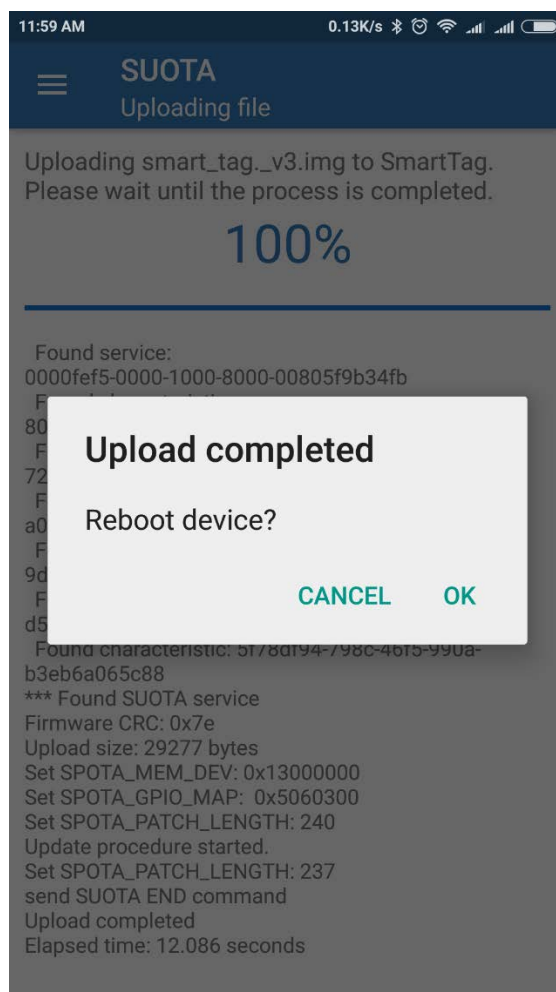


Figure 106: Successful Update Screen

STEP 9: Verify that the New Software is Running on the SmartTag Device

Repeat [STEP 4](#) and [STEP 5](#) to verify that the DIS screen shows the firmware and software version of the new software.

To re-initiate scanning just press the "**SCAN**" button in the top right corner.

Appendix F Miscellaneous**F.1 Calculating the RSSI Value**

Follow these steps to convert the RSSI level (-59 dBm) into signed format:

1. Take positive value: $(59)_{DECIMAL} = (0011\ 1011)_{BINARY}$
2. Reverse all bits: $(0011\ 1011)_{BINARY} \Rightarrow (1100\ 0100)_{BINARY\ REVERSE}$
3. Take 1's complement: $(1100\ 0100)_{BINARY\ REVERSE} \Rightarrow (1100\ 0101)_{1'S\ COMPL}$
4. Convert to hexadecimal: $(1100\ 0101)_{1'S\ COMPL} = (C5)_{HEX}$

Appendix G Troubleshooting

G.1 Magnetometer or Sensor Fusion Have Inconsistent Behaviors

Inconsistent behaviors of magnetometer or Sensor Fusion usually means that the magnetometer is not calibrated or the magnetometer is calibrated but the environmental magnetic field has changed.

Please refer to sections 4.4, 5.11.5, and 5.11.6 for how to calibrate the magnetometer.

G.2 Device Does Not Connect after Switching to a Different Firmware

When switching between different reference applications supported by DA14585 IoT MSK by directly burning one in Flash using the CIB or via SUOTA, users may experience connectivity or functionality issues with the respective Dialog central application. For example, when customers switch to IoT Sensors after using Smart Tags, the IoT Sensors central application might not be able to establish connection.

This issue is caused by the Android/iOS BLE stack and a possible solution is to un-pair the device and/or restart Bluetooth in the Android/iOS system Settings.

G.3 Proximity Sensor is always On

The proximity sensor can detect infrared returns from its IR emitter at angles up to 75° from its perpendicular axis, meaning that the "field of view" is approximately 150°. The plastic enclosure reflects some of the IR light back to the detector and the default thresholds of the proximity detector (section 5.11.5) have been carefully chosen to avoid this problem.

However, due to PCB and enclosure tolerances, the default values might not work as expected. Users can solve this issue by increasing the thresholds of the proximity detector (section 5.11.5).

Revision History

Revision	Date	Description
1.0	3-Aug-2018	Initial version.
1.1	25-Jan-2022	Updated logo, disclaimer, copyright.

Status Definitions

Status	Definition
DRAFT	The content of this document is under review and subject to formal approval, which may result in modifications or additions.
APPROVED or unmarked	The content of this document has been approved for publication.

RoHS Compliance

Dialog Semiconductor's suppliers certify that its products are in compliance with the requirements of Directive 2011/65/EU of the European Parliament on the restriction of the use of certain hazardous substances in electrical and electronic equipment. RoHS certificates from our suppliers are available on request.