DWM1001 FIRMWARE APPLICATION PROGRAMMING INTERFACE (API) GUIDE

Version 2.2

This document is subject to change without notice

DOCUMENT INFORMATION

Disclaimer

Decawave reserves the right to change product specifications without notice. As far as possible changes to functionality and specifications will be issued in product specific errata sheets or in new versions of this document. Customers are advised to check the Decawave website for the most recent updates on this product

LIFE SUPPORT POLICY

Decawave products are not authorized for use in safety-critical applications (such as life support) where a failure of the Decawave product would reasonably be expected to cause severe personal injury or death. Decawave customers using or selling Decawave products in such a manner do so entirely at their own risk and agree to fully indemnify Decawave and its representatives against any damages arising out of the use of Decawave products in such safety-critical applications.

Caution! ESD sensitive device.

Precaution should be used when handling the device in order to prevent permanent damage

DISCLAIMER

(1) This Disclaimer applies to the software provided by Decawave Ltd. ("Decawave") in support of its DWM1001 module product ("Module") all as set out at clause 3 herein ("Decawave Software").

(2) Decawave Software is provided in two ways as follows: -

(a) pre-loaded onto the Module at time of manufacture by Decawave ("Firmware");

(b) supplied separately by Decawave ("Software Bundle").

(3) Decawave Software consists of the following components (a) to (d) inclusive:

(a) The **Decawave Positioning and Networking Stack** ("PANS"), available as a library accompanied by source code that allows a level of user customisation. The PANS software is pre-installed and runs on the Module as supplied, and enables mobile "tags", fixed "anchors" and "gateways" that together deliver the DWM1001 Two-Way-Ranging Real Time Location System ("DRTLS") Network.

(b) The **Decawave DRTLS Manager** which is an Android™ application for configuration of DRTLS nodes (nodes based on the Module) over Bluetooth™.

(c) The **Decawave DRTLS Gateway Application** which supplies a gateway function (on a Raspberry Pi ®) routing DRTLS location and sensor data traffic onto an IP based network (e.g. LAN), and consists of the following components:

- DRTLS Gateway Linux Kernel Module
- DRTLS Gateway Daemon
- DRTLS Gateway Proxy
- DRTLS Gateway MQTT Broker
- DRTLS Gateway Web Manager

(d) **Example Host API functions**, also designed to run on a Raspberry Pi, which show how to drive the Module from an external host microprocessor.

(4) The following third party components are used by Decawave Software and are incorporated in the Firmware or included in the Software Bundle as the case may be: -

(a) The PANS software incorporates the Nordic SoftDevice S132-SD-v3 version 3.0.0 (production) which is included in the Firmware and is also included in the Software Bundle;

(b) The PANS software uses the eCos RTOS which is included in the Software Bundle. The eCos RTOS is provided under the terms of an open source licence which may be found at: http://ecos.sourceware.org/license-overview.html;

(c) The PANS software uses an open source CRC-32 function from FreeBSD which is included in the Software Bundle. This CRC-32 function is provided under the terms of the BSD licence which may be found at:

https://github.com/freebsd/freebsd/blob/386ddae58459341ec56760470780581 4a2128a57/COPYRIGHT;

(d) The Decawave DRTLS Manager application uses open source software which is provided as source code in the Software Bundle. This open source software is provided under the terms of the Apache Licence v2.0 which may be found at http://www.apache.org/licenses/LICENSE-2.0;

(e) The Decawave DRTLS Gateway Application uses the following third party components: -

(i) The Linux Kernel which is provided as source code in the Software Bundle. The Linux Kernel is provided under the terms of the GPLv2 licence which may be found at: https://www.gnu.org/licenses/old-licenses/gpl-2.0.en.html and as such the DWM1001 driver component of the DRTLS Gateway Application is provided under the same license terms;

(ii) The three.js JavaScript library, the downloadable version of which is available here https://threejs.org/, is provided under the terms of the MIT Licence which may be found at https://opensource.org/licenses/MIT.

Items (a), (b), (c), (d) and (e) in this section 4 are collectively referred to as the "Third Party Software"

(5) Decawave Software incorporates source code licensed to Decawave by Leaps s.r.o., a supplier to Decawave, which is included in the Firmware and the Software Bundle in binary and/or source code forms as the case may be, under the terms of a license agreement entered into between Decawave and Leaps s.r.o.

(6) Decawave hereby grants you a free, non-exclusive, non-transferable, worldwide license without the right to sub-license to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating Decawave Software, to modify Decawave Software or incorporate Decawave Software in other software and to design, make, have made, market, sell, have sold or otherwise dispose of products incorporating such modified or incorporated software PROVIDED ALWAYS that the use by you of Third Party Software as supplied by Decawave is subject to the terms and conditions of the respective license agreements as set out at clause 4 herein AND PROVIDED ALWAYS that Decawave Software is used only in systems and products based on Decawave semiconductor products. NO OTHER LICENSE, EXPRESS OR IMPLIED, BY ESTOPPEL OR OTHERWISE TO ANY OTHER DECAWAVE INTELLECTUAL PROPERTY RIGHT, AND NO LICENSE TO ANY THIRD PARTY TECHNOLOGY OR INTELLECTUAL PROPERTY RIGHT, IS GRANTED HEREIN, including but not limited to any patent right, copyright, mask work right, or other intellectual property right relating to any combination, machine, or process in which Decawave semiconductor products or Decawave Software are used.

(7) Downloading, accepting delivery of or using Decawave Software indicates your agreement to the terms of (i) the license grated at clause 6 herein, (ii) the terms of this Disclaimer and (iii) the terms attaching to the Third Party Software. If you do not agree with all of these terms do not download, accept delivery of or use Decawave Software.

(8) Decawave Software is solely intended to assist you in developing systems that incorporate Decawave semiconductor products. You understand and agree that you remain responsible for using your independent analysis, evaluation and judgment in designing your systems and products. THE DECISION TO USE DECAWAVE SOFTWARE IN WHOLE OR IN PART IN YOUR SYSTEMS AND PRODUCTS RESTS ENTIRELY WITH YOU AND DECAWAVE ACCEPTS NO LIABILTY WHATSOEVER FOR SUCH DECISION.

(9) DECAWAVE SOFTWARE IS PROVIDED "AS IS". DECAWAVE MAKES NO WARRANTIES OR REPRESENTATIONS WITH REGARD TO DECAWAVE SOFTWARE OR USE OF DECAWAVE SOFTWARE, EXPRESS, IMPLIED OR STATUTORY, INCLUDING ACCURACY OR COMPLETENESS. DECAWAVE DISCLAIMS ANY WARRANTY OF TITLE AND ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NON-INFRINGEMENT OF ANY THIRD PARTY INTELLECTUAL PROPERTY RIGHTS WITH REGARD TO DECAWAVE SOFTWARE OR THE USE THEREOF.

(10) DECAWAVE SHALL NOT BE LIABLE FOR AND SHALL NOT DEFEND OR INDEMNIFY YOU AGAINST ANY THIRD PARTY INFRINGEMENT CLAIM THAT RELATES TO OR IS BASED ON DECAWAVE SOFTWARE OR THE USE OF DECAWAVE SOFTWARE. IN NO EVENT SHALL DECAWAVE BE LIABLE FOR ANY ACTUAL, SPECIAL, INCIDENTAL, CONSEQUENTIAL OR INDIRECT DAMAGES, HOWEVER CAUSED, INCLUDING WITHOUT LIMITATION TO THE GENERALITY OF THE FOREGOING, LOSS OF ANTICIPATED PROFITS, GOODWILL, REPUTATION, BUSINESS RECEIPTS OR CONTRACTS, COSTS OF PROCUREMENT OF SUBSTITUTE GOODS OR SERVICES; LOSS OF USE, DATA, OR PROFITS; OR BUSINESS INTERRUPTION), LOSSES OR EXPENSES RESULTING FROM THIRD PARTY CLAIMS. THESE LIMITATIONS WILL APPLY REGARDLESS OF THE FORM OF ACTION, WHETHER UNDER STATUTE, IN CONTRACT OR TORT INCLUDING NEGLIGENCE OR ANY OTHER FORM OF ACTION AND WHETHER OR NOT DECAWAVE HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, ARISING IN ANY WAY OUT OF DECAWAVE SOFTWARE OR THE USE OF DECAWAVE SOFTWARE.

(11) You acknowledge and agree that you are solely responsible for compliance with all legal, regulatory and safety-related requirements concerning your products, and any use of Decawave Software in your applications, notwithstanding any applications-related information or support that may be provided by Decawave.

(12) Decawave reserves the right to make corrections, enhancements, improvements and other changes to its software, including Decawave Software, at any time.

Mailing address: Decawave Ltd.,
Adelaide Chambers,
Peter Street,
Dublin D08 T6YA
IRELAND.

are the property of their respective owners.

TABLE OF CONTENTS

DISCLAIMER 3
1 INTRODUCTION AND OVERVIEW 12
1.1 DWM1001 MODULE AND THE FIRMWARE 12
1.2 API AND ITS GUIDE 12
2 GENERAL API INTRODUCTION 13
2.1 EXTERNAL INTERFACE USAGE 13
2.2 LOW POWER MODE WAKE-UP MECHANISM 13
2.3 TLV FORMAT 13
2.4 DWM1001 THREADS 13
2.5 API LIMITATIONS 14
2.6 SPI/UART WAKEUP 14
2.7 POSITION REPRESENTATION 14
3 API OPERATION FLOW 15
3.1 API VIA BLE INTERFACE 15
3.2 API VIA SPI INTERFACE 15
3.2.1 DWM1001 SPI overview 15
3.2.2 SPI Scheme: normal TLV communication 18
3.2.3 SPI Example: normal TLV communication - dwm_gpio_cfg_output 19
3.2.4 SPI Scheme: TLV communication using data ready pin 20
3.2.5 SPI Example: interrupt TLV communication dwm_backhaul_xfer 21
3.2.6 SPI error recovery mechanism 22
3.3 API VIA UART INTERFACE 23
3.3.1 DWM1001 UART overview 23
3.3.2 UART TLV Mode 23
3.3.3 UART scheme: TLV mode communication 24
3.3.4 UART example: TLV mode communication 25
3.3.5 UART scheme: Shell mode communication 25
3.3.6 UART example: Shell Mode communication 26
3.4 GPIO SCHEME: DWM1001 NOTIFIES FOR STATUS CHANGE 26
3.5 API FOR ON-BOARD C CODE DEVELOPERS 27
3.5.1 On-board C code user application 27
3.5.2 On-board User application specifications 28
4 GENERIC API INFORMATION 33
4.1 USED TERMINOLOGY 33
4.2 LITTLE ENDIAN 33
4.3 FIRMWARE UPDATE 33
4.3.1 Firmware update via Bluetooth 33
4.3.2 Firmware update via UWB 33
4.4 FREQUENTLY USED TLV VALUES 34
4.4.1 err_code 34
4.4.2 position 34
4.4.3 gpio_idx 34
4.4.4 gpio_value 35
4.4.5 gpio_pull 35

4.4.6 fw_version 35
4.4.7 cfg_tag 35
4.4.8 cfg_anchor 36
4.4.9 int_cfg 36
4.4.10 stnry_sensitivity 36
4.4.11 evt_id_map 37

5 API FUNCTION DESCRIPTIONS 38

5.1 LIST OF API FUNCTIONS 38
5.2 USAGE OF THE APIS 39
5.3 DETAILS OF THE API FUNCTIONS 39
5.3.1 dwm_pos_set 40
5.3.2 dwm_pos_get 41
5.3.3 dwm_upd_rate_set 42
5.3.4 dwm_upd_rate_get 43
5.3.5 dwm_cfg_tag_set 44
5.3.6 dwm_cfg_anchor_set 46
5.3.7 dwm_cfg_get 48
5.3.8 dwm_sleep 50
5.3.9 dwm_anchor_list_get 51
5.3.10 dwm_loc_get 53
5.3.11 dwm_baddr_set 55
5.3.12 dwm_baddr_get 56
5.3.13 dwm_stnry_cfg_set 57
5.3.14 dwm_stnry_cfg_get 58
5.3.15 dwm_factory_reset 59
5.3.16 dwm_reset 60
5.3.17 dwm_ver_get 61
5.3.18 dwm_uwb_cfg_set 62
5.3.19 dwm_uwb_cfg_get 63
5.3.20 dwm_usr_data_read 64
5.3.21 dwm_usr_data_write 65
5.3.22 dwm_label_read 66
5.3.23 dwm_label_write 67
5.3.24 dwm_gpio_cfg_output 68
5.3.25 dwm_gpio_cfg_input 69
5.3.26 dwm_gpio_value_set 70
5.3.27 dwm_gpio_value_get 71
5.3.28 dwm_gpio_value_toggle 72
5.3.29 dwm_panid_set 73
5.3.30 dwm_panid_get 74
5.3.31 dwm_nodeid_get 75
5.3.32 dwm_status_get 76
5.3.33 dwm_int_cfg_set 78
5.3.34 dwm_int_cfg_get 79
5.3.35 dwm_enc_key_set 80
5.3.36 dwm_enc_key_clear 81
5.3.37 dwm_nvm_usr_data_set 82
5.3.38 dwm_nvm_usr_data_get 83
5.3.39 dwm_gpio_irq_cfg 84
5.3.40 dwm_gpio_irq_dis 85

5.3.41 dwm_i2c_read 86
5.3.42 dwm_i2c_write 87
5.3.43 dwm_evt_listener_register 88
5.3.44 dwm_evt_wait 89
5.3.45 dwm_wake_up 91
5.4 BACKHAUL API FUNCTIONS 92
5.4.1 dwm_bh_status_get 92
5.4.2 dwm_backhaul_xfer 93
6 SHELL COMMANDS 95
6.1 USAGE OF UART SHELL MODE 95
6.2 ? 95
6.3 HELP 96
6.4 QUIT 96
6.5 GC 96
6.6 GG 97
6.7 GS 97
6.8 GT 97
6.9 F 97
6.10 RESET 97
6.11 UT 97
6.12 FRST 98
6.13 TWI 98
6.14 AID 98
6.15 AV 98
6.16 SCS 98
6.17 SCG 99
6.18 LES 99
6.19 LEC 99
6.20 LEP 99
6.21 UTPG 99
6.22 UTPS 99
6.23 SI 100
6.24 NMG 100
6.25 NMO 100
6.26 NMA 100
6.27 NMI 101
6.28 NMT 101
6.29 NMTL 102
6.30 NMB 102
6.31 LA 102
6.32 LB 102
6.33 NIS 103
6.34 NLS 103
6.35 STG 103
6.36 STC 106
6.37 UDI 106
6.38 UUI 106
6.39 TLV 107
6.40 AURS 107
6.41 AURG 107

6.42 APG 107
6.43 APS 107
6.44 ACAS 108
6.45 ACTS 108
6.46 AKS 108
6.47 AKC 108
6.48 ANS 108
6.49 ANG 109

7 BLE API 110

7.1 BLE GATT MODEL 110
7.1.1 Network Node Characteristics 110
7.1.2 Operation mode characteristic 111
7.1.3 Location data characteristic 111
7.1.4 Proxy Positions Characteristic 112
7.1.5 Anchor-specific Characteristics 112
7.1.6 Tag-specific Characteristics 113
7.2 AUTO-POSITIONING 113
7.3 BLE ADVERTISEMENTS 114
7.3.1 Presence Broadcast 114
7.4 FIRMWARE UPDATE 115
7.4.1 Initiating FW Update 115
7.4.2 Transmitting the FW binary 115
7.4.3 Finishing the transmission 116
7.4.4 FW update push/poll format 116

8 APPENDIX A – TLV TYPE LIST 117

9 APPENDIX B – BIBLIOGRAPHY 119

10 DOCUMENT HISTORY 120

11 FURTHER INFORMATION 121

LIST OF TABLES

TABLE 1 TLV FORMAT DATA EXAMPLE 13
TABLE 2 AVAILABLE OF PERIPHERALS IN DWM1001 FIRMWARE SYSTEM 28
TABLE 3 API REQUEST FUNCTION LIST 38
TABLE 4 API EXAMPLES LOCATION 39
TABLE 5 DWM1001 TLV TYPE LIST 117
TABLE 6: DOCUMENT HISTORY 120

LIST OF FIGURES

FIGURE 1 DWM1001 BLE SCHEME 15
FIGURE 2 DWM1001 SPI WORK FLOW 16
FIGURE 3 SPI SCHEME: NORMAL TLV COMMUNICATION 18
FIGURE 4 SPI EXAMPLE: NORMAL TLV COMMUNICATION 19
FIGURE 5 SPI SCHEME: TLV COMMUNICATION USING DATA READY PIN 20
FIGURE 6 SPI EXAMPLE: INTERRUPT TLV COMMUNICATION 22
FIGURE 7 DWM1001 UART WORK FLOW 23

FIGURE 8 UART SCHEME: TLV MODE COMMUNICATION 24
FIGURE 9 UART EXAMPLE: TLV MODE COMMUNICATION 25
FIGURE 10 UART SCHEME: SHELL MODE COMMUNICATION 26
FIGURE 11 UART EXAMPLE: SHELL MODE COMMUNICATION 26
FIGURE 12 GPIO SCHEME: DWM1001 NOTIFIES HOST DEVICE OF STATUS CHANGE, USING GPIO 27

1 INTRODUCTION AND OVERVIEW

1.1 DWM1001 module and the firmware

The DWM1001 module is a radio transceiver module integrating the Nordic Semiconductor nRF52 MCU and Decawave's DW1000 IC. The nRF52 MCU, which has Bluetooth v4.2 protocol stack implemented [1], is acting as the main processor of the DWM1001 module. The DW1000 IC part, which has the UWB physical layer defined in IEEE 802.15.4-2011 standard [2], is acting as the UWB radio module controlled by the nRF52 MCU.

Decawave provides a pre-built firmware library, the "Positioning and Networking stack" (PANS) library, in the DWM1001 module which runs on the nRF52 MCU. The firmware provides the Application Programming Interface (API) for users to use their own host devices to communicate with the DWM1001 module, namely the PANS API. The PANS API essentially is a set of functions providing a means to communicate with the nRF52 MCU to drive the module through the PANS library on application level without having to deal with the details of accessing the DW1000 IC part and other peripherals directly through its SPI/I2C interface register set. The detailed information of the firmware is introduced in the DWM1001 Firmware User Guide [3].

1.2 API and its guide

The PANS APIs are a set of functions. Each individual API function may be accessed through various communication interfaces providing flexibility to developers in using the DWM1001 module and integrating it into their systems. The API accesses mainly come in as two types:

1) External access API: via UART, SPI and BLE.
2) Integrated access API: via on-board user app (C code).

Among the above API interfaces:

- The UART (Generic), the SPI and the on-board APIs are introduced in Sections 5.
- The UART (Shell) APIs are introduced Section 6.
- The BLE APIs are introduced in Section 7.

The detailed introduction to the flow with examples of how the API can be used is introduced in [3].

This document, "*DWM1001 API Guide*", specifies the PANS API functions themselves, providing descriptions of each of the API functions in detail in terms of its parameters, functionality and utility. Users can use the PANS API to configure each individual DWM1001 module. To setup a location system with multiple DWM1001 modules, users should refer to the DWM1001 System Guide [4].

This document relates to: `"DWM1001 PANS Library Version 1.3.0"`

2 GENERAL API INTRODUCTION

2.1 External Interface usage

The external interfaces, including the UART, the SPI and the BLE, are used by the external APIs in the PANS library for Host connection. The on-board user application through C code API cannot make use of the external interfaces due to compatibility reasons.

2.2 Low power mode wake-up mechanism

As provided in the PANS library, the DWM1001 module can work in a low power mode. In the low power mode, the module puts the API related threads into “sleep” state when the API is not currently being communicated by host devices. In this case, the host device needs to wake up the module through external interfaces before the real communication can start.

For UART interface, the host device needs to send a single byte first to wake up the module.

For SPI interface, putting the chip select pin to low level wakes up the module, i.e. no extra transmission is needed.

After the API transmission has finished, the host device needs to put the module back to “sleep” state to save power, as introduced in section 5.3.8 and section 6.4.

2.3 TLV format

TLV format, the Type-Length-Value encoding, is used in the DWM1001 module API. Data in TLV format always begins with the type byte, followed by the length byte, and then followed by a variable number of value bytes [0 to 253] as specified by the length byte. Table 1 shows an example of TLV format data, in which the type byte is 0x28, the length byte is 0x02, and as declared by the length byte, the value field is of two bytes: 0x0D and 0x01.

In DWM1001 firmware, both SPI and UART APIs use TLV format for data transmission. Users should refer to the type list for the meaning of type bytes (see Section 7). And for each specific command or response, the value field is of different length to provide the corresponding information.

Table 1 TLV format data example

TLV request			
Type	Length	Value	
		gpio_idx	gpio_value
0x28	0x02	0x0d	0x01

2.4 DWM1001 threads

In the DWM1001 firmware system, there are many threads, including SPI, BLE, UART, Generic API, User App and other threads. Each thread deals with specific tasks.

The SPI, BLE and UART threads control the data transmission with external devices. They don’t parse the requests they’ve received. All received requests are sent to the Generic API thread.

The Generic API thread is a parser of the received requests. It judges whether the incoming request is valid. If valid, the firmware goes on to prepare the corresponding data as response; if invalid, the firmware uses error message as response. Then the Generic API thread runs the call_back() function which sends the prepared response message back to the thread where the request comes from.

The on-board user application thread is an independent thread for the users to add their own functionalities. The entrance is provided in the dwm\examples\ folder in the DWM1001 on-board package. An example project is given in dwm\examples\dwm-simple\ folder.

2.5 API limitations

API on SPI/UART interface might not work properly when node configuration is being changed via BLE interface while API commands being executed on SPI/UART interface at the same time. Either BLE or SPI/UART should be used at the time for now to avoid potential errors. Certain shell commands also write data into FLASH memory and therefore can cause problems on UART/SPI and BLE if these interfaces being used at the same time. It is caused by the fact that CPU is halted while FLASH is being erased or written.

2.6 SPI/UART wakeup

If DWM sleeps (in low power mode), the wakeup procedure has to be executed before SPI/UART starts accepting commands. At least 35 microseconds wide pulse has to be generated on CS pin of the SPI or at least one byte has to be sent on UART interface in order to wake up from the sleep (only in low power mode).

2.7 Position Representation

In presenting locations and distances in a Real-Time Positioning System, there are two things to consider:

- Accuracy
- Precision

Accuracy is the error between the position reported by the nodes and the real position. Currently the DW1000 used in this design provides approximately 10 cm accuracy.

Precision is the value a least-significant bit (LSB) represents. In the on-board firmware of this system, the precision is 1 mm, i.e. 0.001 meter. The positions are presented in 3-dimension coordinates (X, Y, Z), where X, Y, and Z each is a 32-bit integer (4 bytes). Each LSB represents 1 mm. This is for easier interpretation of the value as well as easier mathematics over the reported values.

When deciding the precision, it is important to choose it with respect to accuracy to get a meaningful result. It is useless to show the user precise values if accuracy is low. The 1mm precision is too fine-grained with respect to the current 10 cm accuracy. Therefore, in presenting the location, precision of 1 cm, i.e. 0.01 meter, is used. Only when the coordinate/distance has changed over 1 cm will the updated value be presented. It is similar as rounding float/double values to meaningful number of decimal places.

3 API OPERATION FLOW

This chapter details the flow of how the provided API interfaces should be used.

3.1 API via BLE interface

In the DWM1001 BLE API design, the DWM1001 module is acting as the BLE peripheral and can be communicated by BLE central devices through the APIs. This document introduces the APIs that the BLE central devices can use for the communication. An android application, the Decawave DRTLS Manager, is provided to exercise the BLE APIs.

In the DWM1001 BLE scheme, normal GATT operations including read, write and notification are provided.

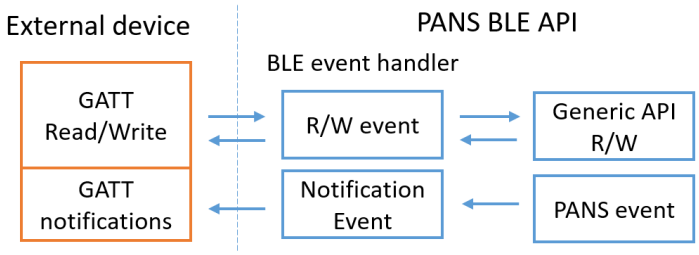

Figure 1 DWM1001 BLE scheme

As shown in Figure 1, the DWM1001 BLE event handler translates the GATT operations into generic API commands. In the meantime, when there are BLE related events, the BLE event handler will send the corresponding notification to the BLE client.

Detailed BLE APIs are introduced in Section 7.

3.2 API via SPI interface

3.2.1 DWM1001 SPI overview

DWM1001 SPI interface uses TLV format data. Users can use an external host device in SPI master mode to connect to the DWM1001 module SPI interface which operates in slave mode. The maximum SPI clock frequency is 8 MHz. (This is maximum supported by the nRF52 MCU)

In the DWM1001 SPI schemes, host device communicates with the DWM1001 through TLV requests. A full TLV request communication flow includes the following steps:

1) Host device sends TLV request;
2) DWM1001 prepares response;
3) Host device reads the SIZE (length of each response) and NUM (number of transfers);
4) Host device reads data response;

Because SPI uses full duplex communication, when the host device (as the SPI master) writes x bytes, it actually sends x bytes to the DWM1001 module (as the slave), and receives x bytes dummy in the same time. Similar for reading, the host device sends x bytes of dummy, and receives x bytes data back as response. In the DWM1001 SPI scheme, the dummy bytes are octets of value 0xFF.

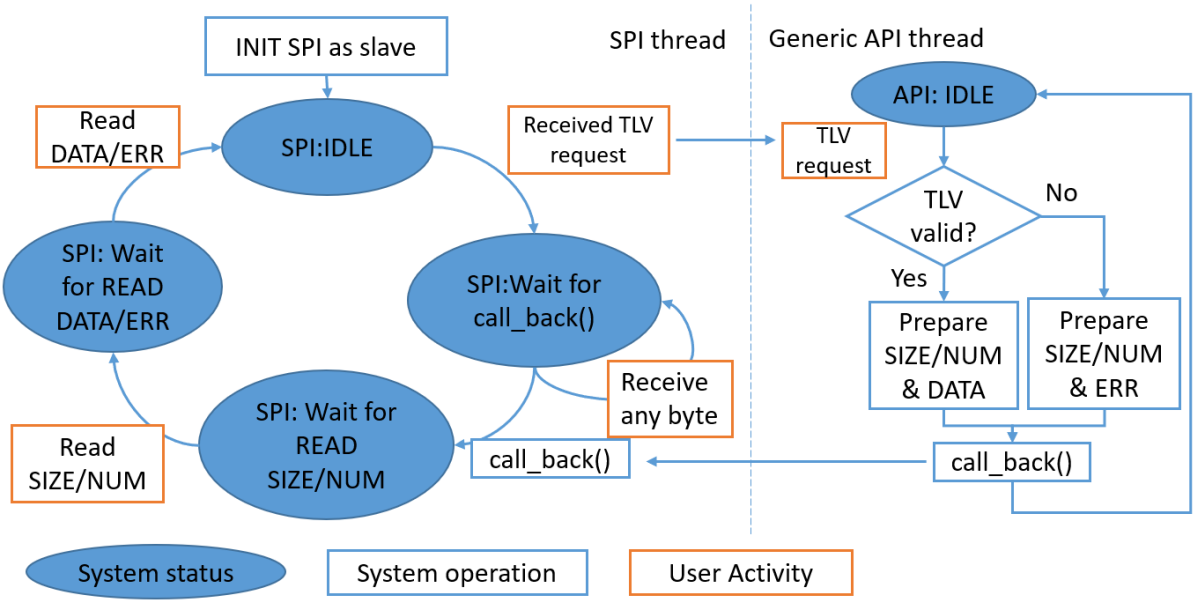

Figure 2 DWM1001 SPI work flow

As shown in Figure 2, the DWM1001 SPI thread transfers between four states in serial. Each state will transfer to its next corresponding state on certain events.

- **SPI: Idle**: the state after initialization and after each successful data transmission/response. In this state, any received data is assumed to be a TLV request. Thus, on receiving any data, the SPI thread in the firmware passes the received data to the Generic API. The Generic API parses the TLV requests and prepares the return data.
    - Waiting for event: receiving TLV requests.
    - Action on event – if Type==0xFF:
        - No action, stay in SPI: Idle. See Section 3.2.6 for more details.
    - Action on event – if Type≠0xFF :
        - Send received TLV request to Generic API thread.
          Transfer to “SPI: Wait for call_back”.

- **SPI: Wait for call_back**: the DWM1001 SPI is waiting for the Generic API to parse the TLV request and prepare the response. Any data from the host side will be ignored and returned with 0x00 in this state.
    - Waiting for event: call_back() function being called by the Generic API.
    - Action on event:
        - Toggle data ready pin HIGH – detailed in Section 3.2.4.
        - Transfer to “SPI: Wait for READ SIZE/NUM”.

- **SPI: Wait for Read SIZE/NUM**: the DWM1001 SPI has prepared the SIZE byte as response, and in total there will be NUM of transfers. SIZE and NUM are totally 2 bytes non-zero data, namely SIZE/NUM. Together indicating the total number (i.e. SIZE*NUM bytes) of data or error message to be responded. The DWM1001 SPI as slave is waiting for the host device to read the SIZE/NUM bytes.

Note: NUM is always of value 1 except for API function dwm_backhaul_xfer.

- Waiting for event: host device reads two bytes.
- Action on event:
  - Respond with the SIZE/NUM bytes.
  - Transfer to “SPI: Wait for READ DATA/ERR”.

- **SPI: Wait for Read DATA/ERR**: the DWM1001 SPI has prepared SIZE bytes of data or error message for each of the NUM transfers as response to the TLV request, and is waiting for the host device to read it.
  - Waiting for event: host device does NUM transfers. Each transfer should be SIZE bytes, otherwise there may be data loss.
  - Action on event:
    - Respond with DATA/ERR.
    - Toggle data ready pin LOW – detailed in Section 3.2.4.
      Transfer to “SPI: Idle”.

In DWM1001, starting from “SPI: Idle”, traversing all four states listed above and returning to “SPI: Idle” indicates a full TLV request communication flow. The user should have received the response data or error message by the end of the communication flow.

A few different usages and examples are illustrated in the following sub-sections.

3.2.2 SPI Scheme: normal TLV communication

Figure 3 shows the normal communication flow of a host device writing/reading information to/from the DWM1001 module:

1) Host device sends the request in TLV format.
2) Host device reads the SIZE/NUM bytes, indicating the number of transfers to be done and the number bytes ready to be read in each transfer.
    a. On receiving SIZE/NUM = 0/0, meaning the response is not ready yet, repeat step 2).
    b. On receiving SIZE/NUM != 0/0, meaning the response is ready, go to step 3)
3) Host device reads SIZE bytes data response in TLV format.

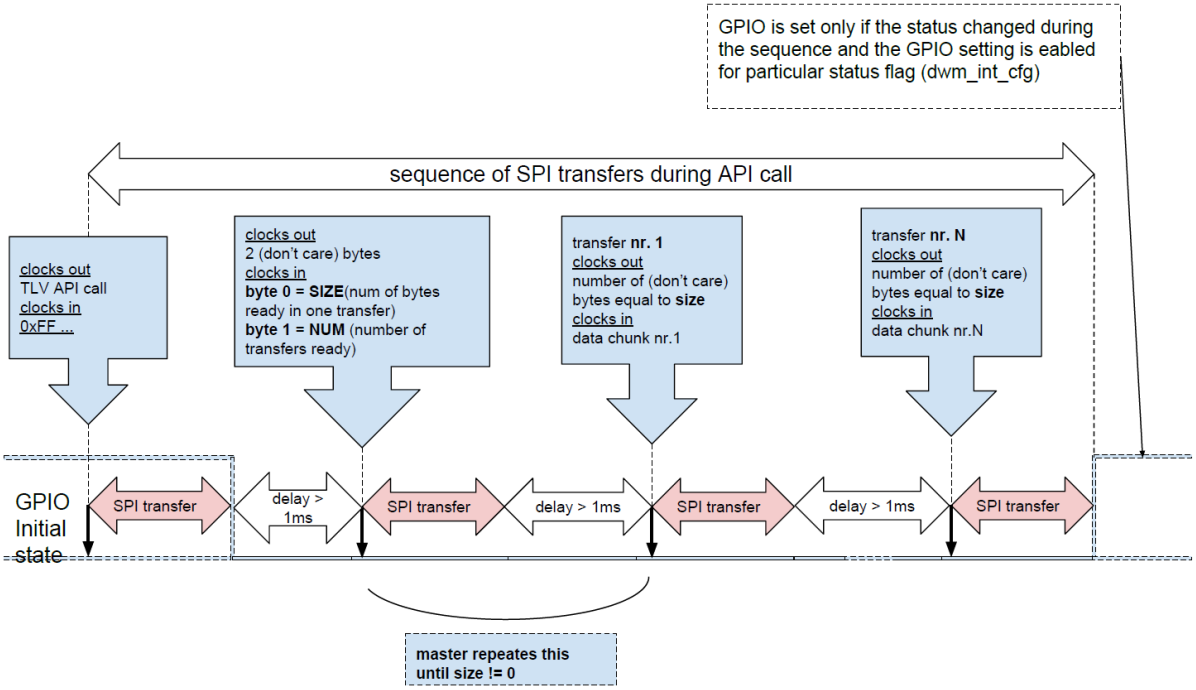

Figure 3 SPI scheme: normal TLV communication

3.2.3 SPI Example: normal TLV communication - dwm_gpio_cfg_output

Figure 4 illustrates an example of host device writing the dwm_gpio_cfg_output TLV request to set pin 13 level HIGH ([0x28, 0x02, 0x0D, 0x01] in byte array, detailed in Section 5.3.18) and reading the response from the DWM1001 module.

The communication flow of the transmission and response includes:

1) Host device writes the dwm_gpio_cfg_output command to set pin 13 level HIGH in TLV format, 4 bytes in total:
   Type = 0x28, length = 0x02, value = 0x0D 0x01.
2) Host device reads the SIZE/NUM bytes, and receives SIZE/NUM = 0/0: the response is not ready yet. Repeat to read the SIZE/NUM bytes.
3) Host device reads the SIZE/NUM bytes again, and receives SIZE/NUM = 3/1: the response is ready. Proceed to read 3 bytes response data, in 1 transfer.
4) Host device reads the 3-byte TLV format data:
   Type = 0x40, Length = 0x01, value = 0x00,
   where Type = 0x40 indicates this is a return message (see Section 7), Length = 0x01 indicates that there is one byte following as data, value = 0x00 indicates the TLV request is parsed correctly.

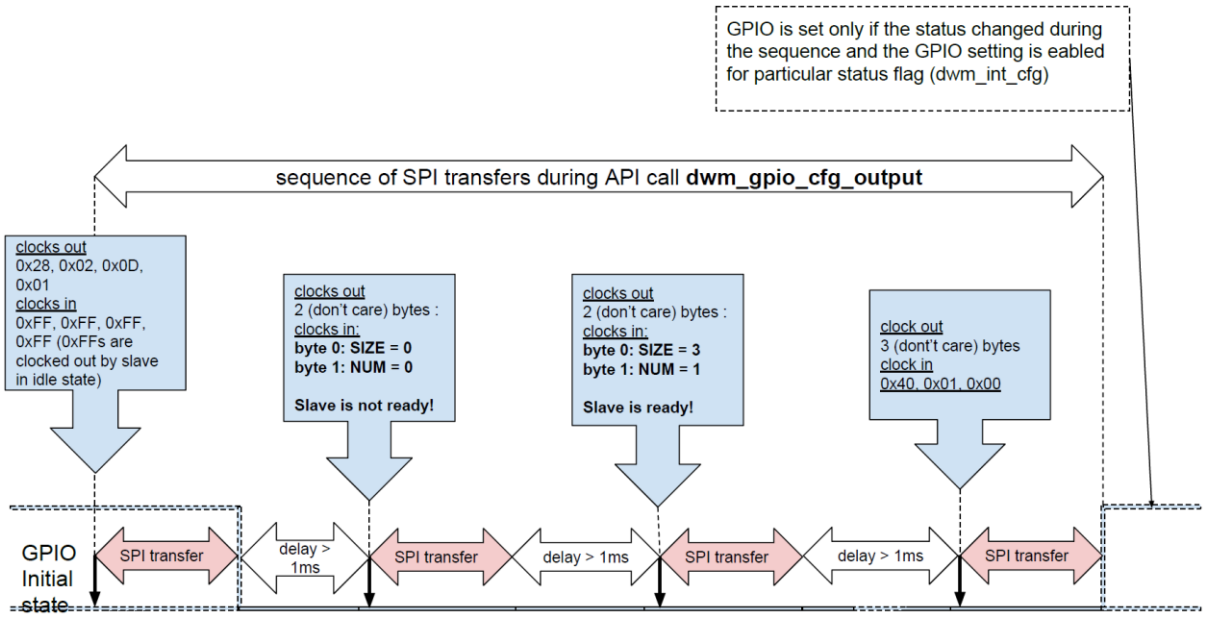

Figure 4 SPI example: normal TLV communication

3.2.4 SPI Scheme: TLV communication using data ready pin

Similar to the scheme described in Section 3.2.2, users can setup the data ready pin (GPIO P0.26) from the DWM1001 to indicate when data is ready, instead of the master polling multiple times to check the response status. When the data ready function is setup, the data ready pin will be set to LOW level when there's no data to be read; when the response SIZE/NUM and data is ready to be read, the data ready pin will be set to HIGH level during states "SPI: Wait for Read SIZE/NUM" and "SPI: Wait for Read DATA/ERR". Thus, the users can use the data ready pin either as an interrupt or as a status pin.

To setup data ready pin for SPI scheme, users need to use dwm_int_cfg TLV request through SPI Scheme: normal TLV communication introduced in Section 3.2.2. The detail of dwm_int_cfg request is introduced in Section 5.3.18.

The communication flow of this scheme is illustrated in Figure 5.

1) Setup the SPI interrupt.
2) Host device writes the request in TLV format.
3) Host device waits until the data ready pin on DWM1001 to go HIGH.
4) Host device reads the SIZE/NUM bytes.
5) Host device reads SIZE bytes data response in TLV format in each transfer for NUM times.

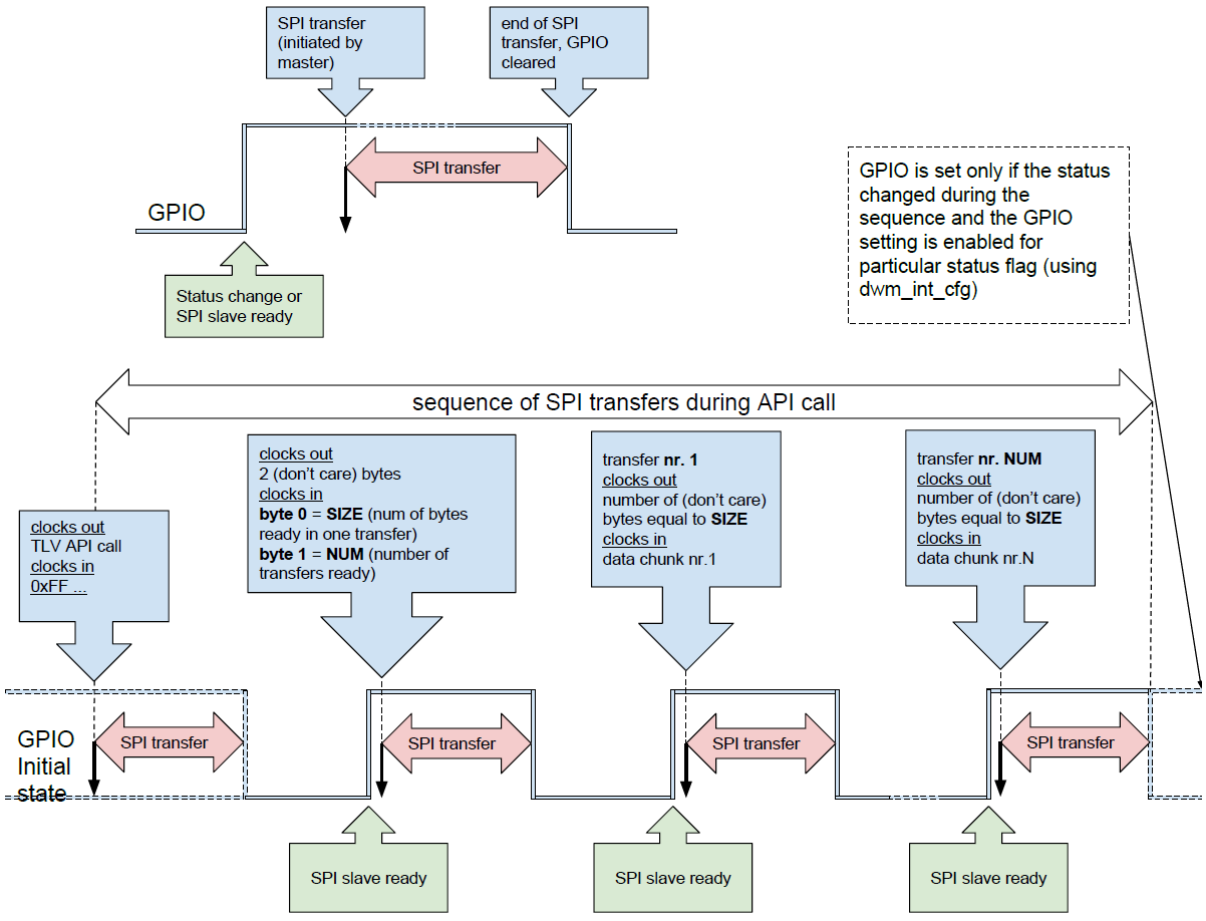

Figure 5 SPI scheme: TLV communication using data ready pin

As can be seen from the steps, this scheme is using the data ready pin on the DWM1001 to indicate the host device when the response data is ready. This makes the host device less busy in reading SIZE byte.

3.2.5 SPI Example: interrupt TLV communication dwm_backhaul_xfer

In the DWM1001 firmware system, function dwm_backhaul_xfer is the only API that makes use of more than one transfer in SPI response.

Maximum payload size of single TLV frame is 253 bytes (Max TLV frame supported by the slave - TLV header = 255 - 2 = 253). Number of bytes which master wants to transfer to the slave (downlink) using command dwm_backhaul_xfer is encoded in the argument of the command. Slave takes the number of downlink data and the number of uplink data that is ready to be transferred to the master and decides the size and the number of transactions in order to transfer both the downlink and the uplink in current command.

Assuming master wants to transfer 299 bytes of downlink data and slave has 1124 bytes of uplink data ready.

downlink bytes: 299 (at least 2 TLV frames)
uplink bytes: 1124 (at least 5 TLV frames)

The master executes command dwm_backhaul_xfer with argument == 299. The slave responds with size == 255 and number of transactions == 5 (5 * 253 = 1265). Both downlink and uplink TLV data chunks are encoded using reserved TLV types that can be used to serialize the data that were encoded in multiple TLV frames. At most 5 TLV frames are supported for now. TLV types 100-104 (0x64-0x68) are reserved for uplink data chunks. TLV types 110-114 (0x6E-0x72) are reserved for downlink data chunks.

The communication flow of this scheme is illustrated in Figure 6.

1) Host device writes the dwm_backhaul_xfer command to indicate it will transfer 299 bytes downlink data in TLV format, 4 bytes in total:
   Type = 0x37, length = 0x02, value = 0x2b 0x01.
2) Host device waits for the data ready pin on DWM1001 to go HIGH.
3) Host device reads the SIZE/NUM bytes, and receives SIZE/NUM = 255/5: the response is ready, should be read through 5 transfers, each of a 255-byte chunk. Proceed to read 1st data chunk in response, expecting 5 transfers.
4) Host device writes its 1st downlink data chunk of 253 bytes. In the meantime, reads the first uplink chunk 253 bytes of the uplink data:
   Type = 0x6E, Length = 0xFD, value = [downlink_data bytes 0-252],
5) Host device waits for the data ready pin on DWM1001 to go HIGH.
6) Host device writes its 2nd downlink data chunk of 46 bytes, together with 207 bytes of dummy bytes 0xFF. In the meantime, it reads the 2nd chunk of 253 bytes of the uplink data:
   Type = 0x6F, Length = 0x2E, value = [downlink_data bytes 253-298], dummy = 0xFF*207
7) For the 3rd to the 5th data transfer, the host device waits until the data ready pin on DWM1001 to go HIGH. And then transfers 255 dummy bytes of 0xFF to receive the 3rd to the 5th uplink data chunks. In the 5th uplink data chunk, the received data is 112 bytes of uplink data plus 141 bytes of dummy data 0xFF because all uplink data have been transferred.

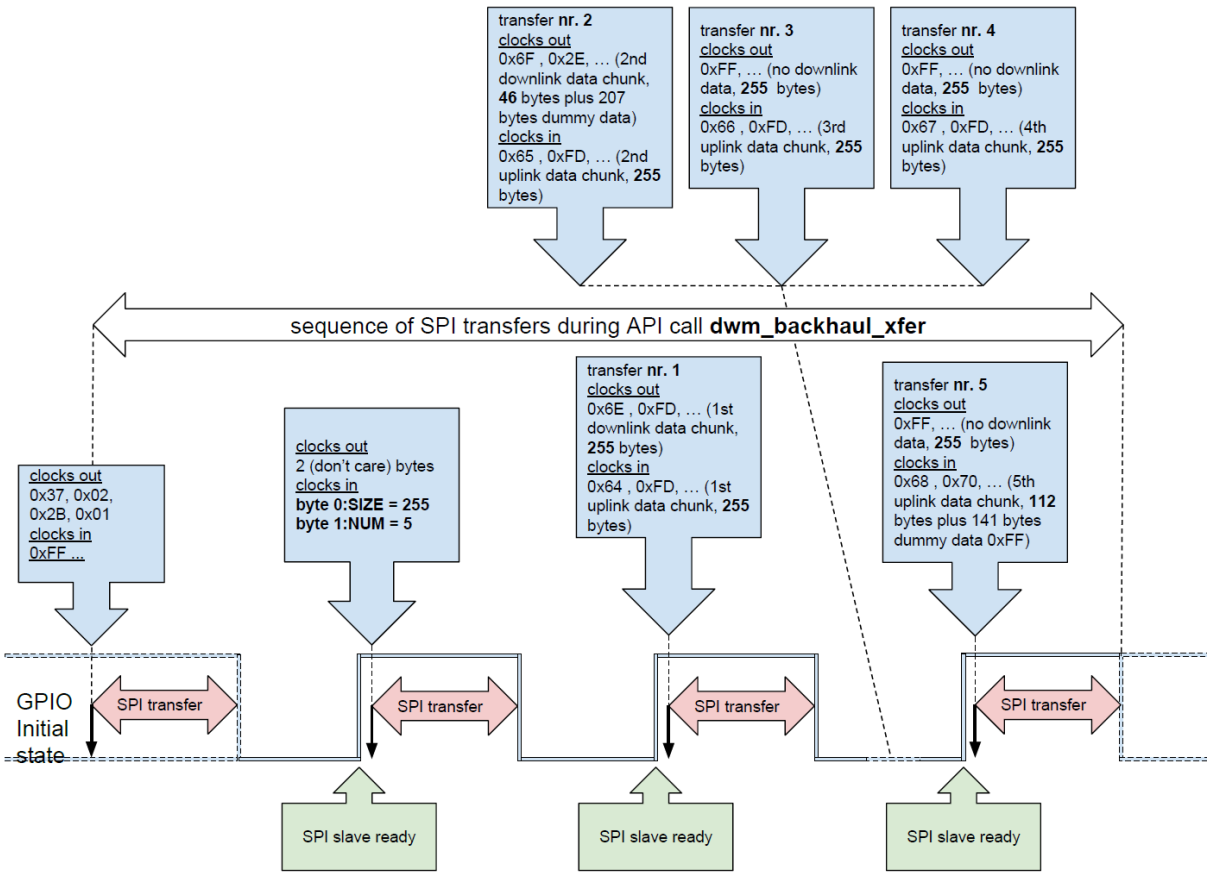

Figure 6 SPI example: interrupt TLV communication

3.2.6 SPI error recovery mechanism

3.2.6.1 SPI data doesn't allow partial transmission

When reading data from the DWM1001 module, if the host device doesn't read all bytes of data in one transmission, the reading operation will still be considered as done. The rest of the response will be abandoned. For example, in "SPI: Wait for Read DATA/ERR" state, the DWM1001 module has prepared SIZE bytes of response data and expects the host device to read all SIZE bytes of the response. However, if the host device only reads part of the data, the DWM1001 module will drop the rest of the data, and transfers to the next state: "SPI: IDLE".

3.2.6.2 SPI state recovery: type_nop message

The DWM1001 SPI has a special Type value 0xFF, called type_nop. A TLV data message with type_nop means no operation. In "SPI: IDLE" state, when the DWM1001 SPI receives a message and finds the type byte is 0xFF, it will not perform any operation, including sending the TLV data message to the Generic API thread.

The type_nop is designed for error recovery. If the host device is not sure what state the DWM1001 SPI is in, it can make use of the SPI response and the non-partial transmission mechanism, and reset the DWM1001 SPI to "SPI: IDLE" state by sending three 0xFF dummy bytes, each in a single transmission. After the three transmissions, the response data from the DWM1001 SPI will become all dummy bytes of value 0xFF, indicating that the DWM1001 SPI is in "SPI: IDLE" state.

3.3 *API via UART interface*

3.3.1 DWM1001 UART overview

Users can use an external host device to connect to the DWM1001 module through UART interface at baud rate 115200. Figure 7 shows the work flow of the DWM1001 UART interface. In the UART Generic mode communication, the host device is acting as the initiator, while the DWM1001 module is the responder.

DWM1001 UART provides two modes: the UART Generic mode, introduced in Section 5, and the UART Shell mode, introduced in Section 6. The default mode of the DWM1001 UART is Generic mode. However, the two modes are transferrable:

Generic mode to Shell mode: press “Enter” twice or input two bytes [0x0D, 0x0D] within one second. If the module is in “sleep” state in low power mode as introduced in Section 2.2, an extra byte will be needed before the double “Enter”. E.g. pressing “Enter” three times will wake up the module and transfer it to Shell mode.

Shell mode to Generic mode: users need to input “quit” command.

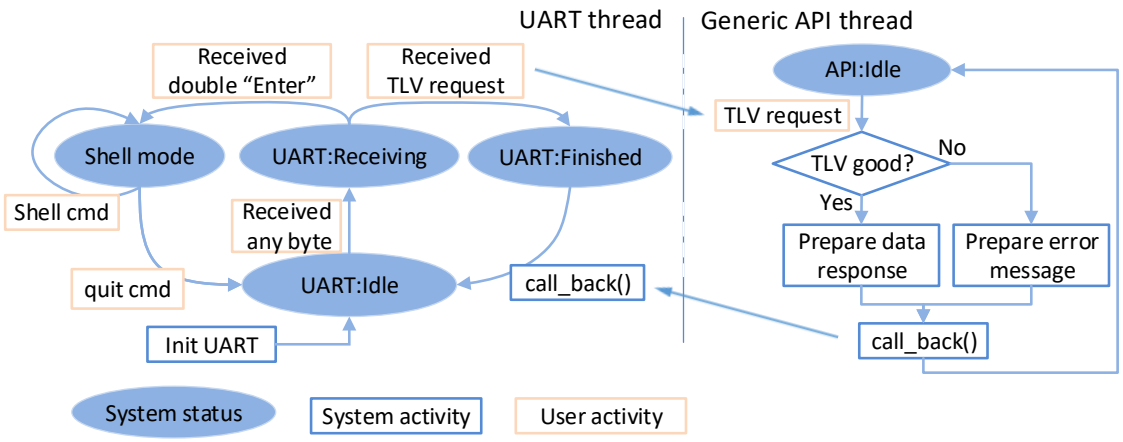

Figure 7 DWM1001 UART work flow

3.3.2 UART TLV Mode

UART TLV mode uses TLV format data. In this mode, host device and the DWM1001 module communicate using TLV requests/responses. A full TLV request communication flow includes the following steps:

1) Host device sends TLV request;
2) DWM1001 responds with TLV data.

On receiving any data, the UART starts a delay timer for the following data. If there is new data coming in within a delay period, specifically 25 clock cycles (25/32768 second ≈ 763 µs), the UART starts the delay timer and waits for new data. If there’s no new data coming in within the delay period, the delay timer will expire. The UART then sends the received data to the Generic API thread and waits for it to return the response data or error message.

As shown in Figure 7, the DWM1001 UART TLV mode thread transfers between three states in serial:

"UART: Idle", "UART: Receiving" and "UART: Finished". Each state will transfer to its next corresponding state on certain events.

- **UART: Idle**: is the state after initialization and after each successful TLV response. In this state, the UART is only expecting one byte as the start of the TLV request or the double "Enter" command.
    - Waiting for event: receiving TLV requests.
    - Action on event:
        - Start the delay timer.
        - Transfer to UART: Receiving.

- **UART: Receiving**: is the state waiting for end of the incoming request. On receiving any data in this state, the UART will refresh the delay timer. If the host device has finished sending bytes, the delay timer will expire.
    - Waiting for event: delay period timed out.
    - Action on event - if received request is double "Enter":
        - Transfer to UART Shell mode.
    - Action on event - if received request is not double "Enter":
        - Send received request to Generic API thread.
        - Transfer to UART: Finished.

- **UART: Finished**: is the state waiting for the Generic API thread to parse the incoming request and send the response data or error message back to UART thread.
    - Waiting for event: call_back() function called by the Generic API thread.
    - Action on event:
        - Send the response data or error message to host device.
        - Transfer to UART: Idle.

3.3.3 UART scheme: TLV mode communication

The UART communication in TLV mode is illustrated in Figure 8, steps described as below:

1) The host device sends a request in TLV format;
2) The DWM1001 module responds with a message in TLV format.

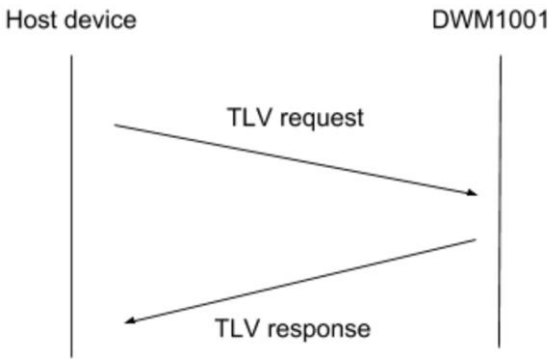

Figure 8 UART scheme: TLV mode communication

3.3.4 UART example: TLV mode communication

Figure 9 illustrates an example of host device sending the dwm_gpio_cfg_output command to set pin 13 level HIGH ([0x28, 0x02, 0x0D, 0x01] in byte array, detailed in Section 5.3.18) and receiving the response from the DWM1001 module using the UART API in TLV mode. The steps of the communication flow are:

1) The host device sends the dwm_gpio_cfg_output command in TLV format:
   Type = 0x28, Length = 0x02, Value = 0x0D 0x01.

2) The DWM1001 module responds in TLV format:
   Type = 0x40, Length = 0x01, value = 0x00,
   where Type = 0x40 indicates this is a return message (see Section 7), Length = 0x01 indicates that there is one byte following as data, value = 0x00 indicates the TLV request is parsed correctly.

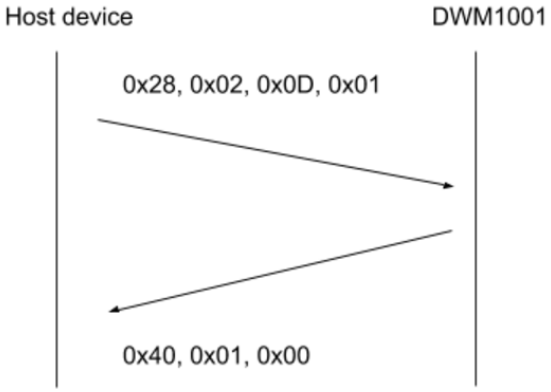

Figure 9 UART example: TLV mode communication

3.3.5 UART scheme: Shell mode communication

UART Shell mode provides prompt and uses Shell commands. The UART Shell mode intends to provide users with human readable access to the APIs, thus, all Shell commands are strings of letters followed by a character return, i.e. “Enter”. Users can input the string directly through keyboard and press “Enter” to send the Shell commands. DWM1001 UART stays in the Shell mode after each Shell commands except for the “quit” command.

As illustrated in Figure 10, a full Shell command communication flow includes the following steps:

1) Host device sends the Shell command + “Enter” to the DWM1001.
2) If there’s any message to respond, the DWM1001 sends the message to the host device.
3) If there’s nothing to respond, the DWM1001 doesn’t send anything and keeps quiet.

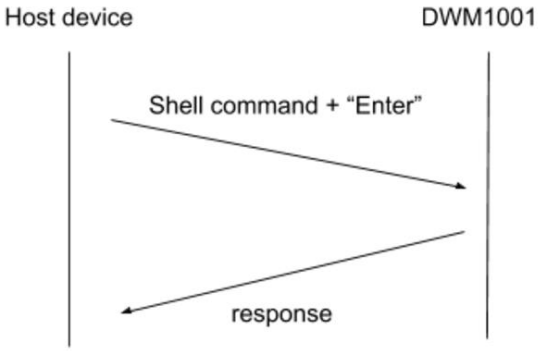

Figure 10 UART scheme: Shell mode communication

3.3.6 UART example: Shell Mode communication

Figure 11 illustrates an example of host device sending the “GPIO set” command using UART Shell to set pin 13 level HIGH (“gs 13” in byte array, followed by “Enter” key, detailed in Section 6.7). The steps of the communication flow are:

1) The host device sends the “GPIO set” command in Shell mode: “gs 13” + “Enter”.
2) The DWM1001 responds the host with string “gpio13: 1”.

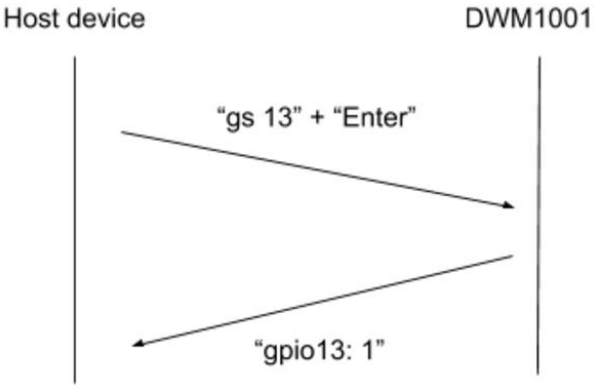

Figure 11 UART example: Shell mode communication

3.4 GPIO Scheme: DWM1001 notifies for status change

Rather than the host device initiating the SPI/UART communication, the DWM1001 module can send notifications of status change by toggling the dedicated GPIO pin (P0.26) to HIGH level, as illustrated in Figure 12. To enable this feature, host device needs to use the dwm_int_cfg command, detailed in Section 5.3.18. On detecting the HIGH level, host device can initiate a dwm_status_get command, detailed in Section 0, to get the status from the DWM1001 device. Both dwm_int_cfg and dwm_status_get commands can be sent through SPI or UART schemes introduced in the previous sections.

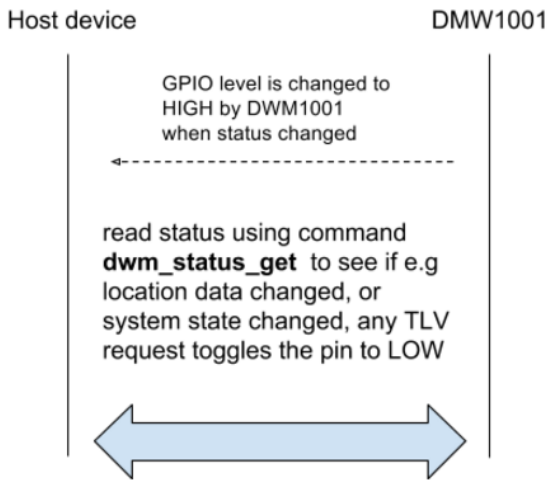

Figure 12 GPIO scheme: DWM1001 notifies host device of status change, using GPIO

This GPIO pin level change will be postponed if the status change happens during the SPI TLV request/response procedure described above to avoid conflict. In detail, when the SPI is in status: "SPI: Wait for call_back", "SPI: Wait for Read SIZE" and "SPI: Wait for Read DATA/ERR", the GPIO scheme will surrender the control of the GPIO pin. After the SPI communication when the SPI is in status "SPI: Idle", the GPIO scheme will regain the control of the GPIO pin.

3.5 API for on-board C code developers

3.5.1 On-board C code user application

Decawave provides the DWM1001 on-board pack [5] of the pre-built firmware. Users can add their own code and make use of the C code API functions in certain entry files provided in the source pack. In this way, users are able to add their own functions inside the module firmware and perhaps not need to add an external host controller device. The detail of adding such user code is provided in DWM1001 Firmware User Guide [3].

A few points the C code users should note when using the on-board firmware:

1) User application is based on eCos RTOS and DWM libraries.
2) Files used for linking with the user applications:
    a. dwm.h - header file - wrapper for all header files necessary for building of the user application
    b. libdwm.a - static library
    c. extras.o, vectors.o, libtarget.a - eCos static library
    d. target_s132_fw2.ld - linker script for firmware section 2
3) The API provides functions and defines to the user application
    a. Common functions on operating systems like thread creation, memory allocation, access to interfaces (e.g. GPIO, SPI), synchronization (e.g. mutex, signal)
    b. Initialization, configuration and maintenance of the DWM communication stack
    c. Register of callbacks for incoming data and measurements

3.5.2 On-board User application specifications

3.5.2.1 System specifications

Maximum user threads: 5
RAM for end user: 5 kB
FLASH for end user: 40 kB

3.5.2.2 Peripheral used by PANS

Table 2 below indicates the usage of all the peripherals used by PANS. Note that the ones in red are all used (blocked or restricted) by PANS. Users can only make use of the “open” ones.

Table 2 available of peripherals in DWM1001 firmware system

Interrupt vector	Peripheral	Instance	Description	Access
0	CLOCK	CLOCK	Clock control	restricted if BLE is enabled (can be accessed by SD API)
	POWER	POWER	Power control	
	BPROT	BPROT	Block Protect	
1	RADIO	RADIO	2.4 GHz radio	Blocked if BLE is enabled
2	UARTE	UARTE0	Universal Asynchronous Receiver/Transmitter with EasyDMA	blocked if UART API is compiled
	UART	UART0	Universal Asynchronous Receiver/Transmitter	
3	SPIM	SPIM0	SPI master 0	open
	SPIS	SPIS0	SPI slave 0	
	TWIM	TWIM0	Two-wire interface master 0	
	TWI	TWI0	Two-wire interface master 0	

	SPI	SPI0	SPI master 0	
	TWIS	TWIS0	Two-wire interface slave 0	
4	SPIM	SPIM1	SPI master 1	blocked
	TWI	TWI1	Two-wire interface master 1	
	SPIS	SPIS1	SPI slave 1	
	TWIS	TWIS1	Two-wire interface slave 1	
	TWIM	TWIM1	Two-wire interface master 1	
	SPI	SPI1	SPI master 1	
5	NFCT	NFCT	Near Field Communication Tag	open
6	GPIOTE	GPIOTE	GPIO Tasks and Events	blocked
7	SAADC	SAADC	Analog to digital converter	open
8	TIMER	TIMER0	Timer 0	blocked
9	TIMER	TIMER1	Timer 1	blocked
10	TIMER	TIMER2	Timer 2	open
11	RTC	RTC0	Real-time counter 0	blocked
12	TEMP	TEMP	Temperature sensor	blocked
13	RNG	RNG	Random number generator	blocked
14	ECB	ECB	AES Electronic Code Book (ECB) mode block encryption	blocked
15	CCM	CCM	AES CCM Mode Encryption	restricted if BLE is enabled (can be accessed by SD

				API)
	AAR	AAR	Accelerated Address Resolver	restricted if BLE is enabled (can be accessed by SD API)
16	WDT	WDT	Watchdog timer	blocked (timeout is 10 seconds)
17	RTC	RTC1	Real-time counter 1	blocked
18	QDEC	QDEC	Quadrature decoder	open
19	LPCOMP	LPCOMP	Low power comparator	open
19	COMP	COMP	General purpose comparator	open
20	SWI	SWI0	Software interrupt 0	open
	EGU	EGU0	Event Generator Unit 0	
21	EGU	EGU1	Event Generator Unit 1	blocked if BLE is enabled
	SWI	SWI1	Software interrupt 1	
22	SWI	SWI2	Software interrupt 2	blocked if BLE is enabled
	EGU	EGU2	Event Generator Unit 2	
23	SWI	SWI3	Software interrupt 3	open
	EGU	EGU3	Event Generator Unit 3	
24	EGU	EGU4	Event Generator Unit 4	
	SWI	SWI4	Software interrupt 4	
25	SWI	SWI5	Software interrupt 5	

	EGU	EGU5	Event Generator Unit 5	
26	TIMER	TIMER3	Timer 3	open
27	TIMER	TIMER4	Timer 4	open
28	PWM	PWM0	Pulse Width Modulation Unit 0	open
29	PDM	PDM	Pulse Density Modulation (Digital Microphone Interface)	open
30	NVMC	NVMC	Non-Volatile Memory Controller	blocked
31	PPI	PPI	Programmable Peripheral Interconnect	blocked
32	MWU	MWU	Memory Watch Unit	restricted if BLE is enabled (can be accessed by SD API)
33	PWM	PWM1	Pulse Width Modulation Unit 1	open
34	PWM	PWM2	Pulse Width Modulation Unit 2	open
35	SPI	SPI2	SPI master 2	blocked if SPI API is compiled
	SPIS	SPIS2	SPI slave 2	
	SPIM	SPIM2	SPI master 2	
36	RTC	RTC2	Real-time counter 2	blocked
37	I2S	I2S	Inter-IC Sound Interface	open
38	FPU	FPU	FPU interrupt	open
0	GPIO	P0	General purpose input and output	blocked, user can access application pins via API

N/A	FICR	FICR	Factory Information Configuration	blocked
N/A	UICR	UICR	User Information Configuration	blocked

3.5.2.3 On-board application use cases

1) Location engine improvements - is possible to retrieve distances between TN and ANs and use user implemented location engine to compute position. Data offload is possible only via IoT data.
2) Sensor fusion - is possible to connect to other sensors via SPI/I2C and improve location accuracy via sensor fusion (e.g. Kalman and etc…).
3) Sensor stations - read some sensors (temperature, humidity …) and report their value via IoT.
4) Remote controller - sending command via IoT to the node which will respond with some activity (PWM change, LED blink, GPIO state change).
5) Data bridge - add IoT/location services for external attached MCU (e.g. robotic platform).

4 GENERIC API INFORMATION

4.1 Used terminology

Anchor: has a fixed location – as a reference point to locate Tags. The module may be configured to behave as an anchor node. An anchor initiator is an anchor with the initiator flag enabled as introduced in section 4.4.8. It is a special anchor node that initiates the whole network, see the system overview for more detail [4].

Tag: usually moving/changing location, determines its position dynamically with the help of the anchors. The module may be configured to behave as a tag node.

Gateway: knows about all nodes in the network, provides status information about network nodes (read/inspect), cache this information and provide it to gateway client, provides means to interact with network elements (a.k.a. interaction proxy).

Node: network node (anchor, tag, gateway...)

LE: location engine – position solver function (on the tag)

4.2 Little endian

The integers used in the PANS API are little endian, unless otherwise stated.

4.3 Firmware Update

When the RTLS network is forming, the initiator anchor specifies the firmware version necessary for the network. When automatic FW update is enabled, any devices wishing to participate (join) the network must have the same firmware (version number and the checksum). If a new device does not have the correct firmware it will be updated as per the sub-sections below.

4.3.1 Firmware update via Bluetooth

If one wants to update the entire network to a new firmware image while the network is operational, it is sufficient to just update the initiator via Bluetooth. The initiator will then propagate the new firmware to all of the other devices over the UWB radio link automatically.

Note, as the initiator is updated first, it will restart the network and as each device re-joins the network its firmware will be updated. Thus, during the FW update the nodes which are performing the updated will be “offline”.

4.3.2 Firmware update via UWB

As introduced in the DWM1001 System Overview [4], the nodes will compare their firmware version to the network they want to join. If the firmware version is different, the nodes will try to update their firmware before joining. This firmware update function can be enabled/disabled in the configuration. Here lists the rules of the function that the nodes will follow.

Tag:

When enabled, the tag will always check the firmware version and try to synchronise its firmware version with the network by sending the update request to the nearby anchor nodes in the network before it starts ranging.

When disabled, the tag will start ranging without checking the firmware version. This can lead to version compatibility problems and must be dealt with very carefully.

Anchor:

When enabled, before joining the network, the anchor will check the firmware version and try to synchronise its firmware version with the network by sending the update request to the nearby anchor nodes. After having joined the network, the anchor will respond to nearby nodes' requests to update their firmware.

When disabled, before joining the network, the anchor will directly send the join request and will not check the firmware version. This can lead to version compatibility problems and must be dealt with very carefully. After having joined the network, the anchor will ignore the firmware update requests from the nearby nodes.

4.4 Frequently used TLV values

Below lists the data that are frequently used in the APIs, either as input parameters or output parameters. These parameters are of fixed size and some have their own value ranges.

4.4.1 err_code

byte error code, response information to the incoming requests.

err_code : 8-bit integer, valid values:
0 : OK
1 : unknown command or broken TLV frame
2 : internal error
3 : invalid parameter
4 : busy
5 : operation not permitted

4.4.2 position

13-byte position information of the node (anchor or tag).

position = x, y, z, pqf : bytes 0-12, position coordinates and quality factor
x : bytes 0-3, 32-bit integer, in millimeters
y : bytes 4-7, 32-bit integer, in millimeters
z : bytes 8-11, 32-bit integer, in millimeters
pqf : bytes 12, 8-bit integer, position quality factor in percent

4.4.3 gpio_idx

byte index of GPIO pins available to the user.

gpio_idx : 8-bit integer, valid values: 2, 8, 9, 10, 12, 13, 14, 15, 22, 23, 27, 30, 31.

Note: some GPIO pins are occupied by the PANS library and can only be accessed/controlled by through the APIs partially:

- When configured as tag and BLE function is enabled, GPIO2 is occupied.
- When LED function is enabled, GPIO 22, 30 and 31 are occupied.
- During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus, these GPIOs should be used with care during the first 1s of a reboot operation.

4.4.4 gpio_value

byte value of the GPIO pin.

gpio_value = 8-bit integer, valid values:
0 : set the I/O port pin to a LOW voltage, logic 0 value
1 : set the I/O port pin to a HIGH voltage, logic 1 value

4.4.5 gpio_pull

1-byte status of the GPIO pin as input.

gpio_pull = 8-bit integer, valid values:
0 : DWM_GPIO_PIN_NOPULL
1 : DWM_GPIO_PIN_PULLDOWN
3 : DWM_GPIO_PIN_PULLUP

4.4.6 fw_version

4-byte value representing the version number of the firmware.

fw_version = **maj**, **min**, **patch**, **ver**: bytes 0-3, firmware version
maj : byte 0, 8-bit number, MAJOR
min : byte 1, 8-bit number, MINOR
patch : byte 2, 8-bit number, PATCH
ver : byte 3, 8-bit number, res and var
res : byte 3, bits 4-7, 4-bit number, RESERVED
var : byte 3, bits 0-3, 4-bit number, VARIANT

4.4.7 cfg_tag

2 bytes, configuration of the tag, which contains common tag configuration parameters. Each bit represents different thing.

cfg_tag = stnry_en, low_power_en, meas_mode, loc_engine_en, led_en, ble_en, uwb_mode,

fw_update_en, enc_en: tag configuration information.
stnry _en: Stationary detection enabled, if enabled, the stationary update rate is used instead of normal update rate if node is not moving.
meas_mode: measurement mode. 0 - TWR; 1, 2, 3 - reserved.
low_power_en: low-power mode enable.
loc_engine_en: internal Location Engine enable. 0 means do not use internal Location Engine, 1 means internal Location Engine.
enc_en : encryption enable
led_en : general purpose LEDs enable
ble_en: Bluetooth enable.
uwb_mode: UWB operation mode: 0 - offline, 1 – passive, 2 – active.
fw_upd_en: firmware update enable.

4.4.8 cfg_anchor

2 bytes, configuration of the anchor, which contains common anchor configuration parameters. Each bit represents different thing.

cfg_anchor = initiator, bridge, led_en, ble_en, uwb_mode, fw_update_en, enc_en,
initiator : Initiator role enable.
bridge : Bridge role enable.
enc_en : encryption enable
led_en : general purpose LEDs enable
ble_en : Bluetooth enable.
uwb_mode: UWB operation mode: 0 - offline, 1 – passive, 2 – active.
fw_upd_en : Firmware update enable.

4.4.9 int_cfg

16 bits (2 bytes), enable and disable configuration of interrupt through dedicated GPIO pin (pin 19: READY). Each bit represents different thing. For all the flags, 0=disabled, 1=enabled.

int_cfg = spi_data_ready, loc_ready, bh_status_changed, bh_data_ready, bh_initialized_changed, uwb_scan_ready, usr_data_ready, uwbmac_joined_changed, usr_data_sent
bit 0: **loc_ready:** interrupt is generated when location data are ready
bit 1: **spi_data_ready**: new SPI data generates interrupt on dedicated GPIO pin
bit 2: **bh_status_changed:** UWBMAC status changed
bit 3: **bh_data_ready:** UWBMAC backhaul data ready
bit 4: **bh_initialized_changed:** UWBMAC route configured
bit 5: **uwb_scan_ready**: UWB scan result is available
bit 6: **usr_data_ready**: user data received over UWBMAC
bit 7: **uwbmac_joined_changed:** UWBMAC joined
bit 8: **usr_data_sent**: user data TX completed over UWBMAC
bit 9-15: reserved

4.4.10 stnry_sensitivity

1-byte stationary sensitivity. The threshold configuration of the on-board accelerometer module.

stnry_sensitivity = 8-bit integer, valid values:
0: low [512 mg]
1: normal [2048 mg]
2: high [4064 mg]

4.4.11 evt_id_map

32-bit ID map of the event. Each bit represents a different event.

evt_id_map = 32-bit integer, flags representing:
bit 0: **DWM_EVT_LOC_READY**
bit 1: **DWM_EVT_UWBMAC_JOINED_CHANGED**
bit 4: **DWM_EVT_UWB_SCAN_READY**
bit 5: **DWM_EVT_USR_DATA_READY**
bit 6: **DWM_EVT_USR_DATA_SENT**
other bits: reserved

5 API FUNCTION DESCRIPTIONS

5.1 List of API functions

The API functions for on-board user app (c code) and SPI/UART TLV commands are unified. Listed below in Table 3, in TLV type ascending order.

Table 3 API request function list

API function name	On-board user app available	SPI/UART TLV		Ref (Section)
		type	Length	
dwm_pos_set	y	0x01	0x0d	5.3.1
dwm_pos_get	y	0x02	0x00	5.3.2
dwm_upd_rate_set	y	0x03	0x02	5.3.3
dwm_upd_rate_get	y	0x04	0x00	5.3.4
dwm_cfg_tag_set	y	0x05	0x02	5.3.5
dwm_cfg_anchor_set	y	0x07	0x01	5.3.6
dwm_cfg_get	y	0x08	0x00	5.3.7
dwm_sleep	y	0x0a	0x00	5.3.8
dwm_anchor_list_get	y	0x0b	0x01	5.3.9
dwm_loc_get	y	0x0c	0x00	5.3.10
dwm_baddr_set	y	0x0f	0x00	5.3.11
dwm_baddr_get	y	0x10	0x00	5.3.12
dwm_stnry_cfg_set	y	0x11	0x01	5.3.13
dwm_stnry_cfg_get	y	0x12	0x00	5.3.14
dwm_factory_reset	y	0x13	0x00	5.3.15
dwm_reset	y	0x14	0x00	5.3.16
dwm_ver_get	y	0x15	0x00	5.3.17
dwm_uwb_cfg_set	y	0x17	0x05	5.3.18
dwm_uwb_cfg_get	y	0x18	0x00	5.3.19
dwm_usr_data_read	y	0x19	0x00	5.3.20
dwm_usr_data_write	y	0x1a	0xff	5.3.21
dwm_label_read	y	0x1c	0x00	5.3.22
dwm_label_write	y	0x1d	0x10	5.3.23
dwm_gpio_cfg_output	y	0x28	0x02	5.3.24
dwm_gpio_cfg_input	y	0x29	0x02	5.3.25
dwm_gpio_value_set	y	0x2a	0x02	5.3.26
dwm_gpio_value_get	y	0x2b	0x01	5.3.27
dwm_gpio_value_toggle	y	0x2c	0x01	5.3.28
dwm_panid_set	y	0x2e	0x02	5.3.29
dwm_panid_get	y	0x2f	0x00	5.3.30
dwm_nodeid_get	y	0x30	0x00	5.3.31
dwm_status_get	n	0x32	0x00	5.3.32
dwm_int_cfg_set	n	0x34	0x02	5.3.33
dwm_int_cfg_get	n	0x35	0x00	5.3.34

dwm_enc_key_set	y	0x3c	0x10	5.3.35
dwm_enc_key_clear	y	0x3d	0x00	5.3.36
dwm_nvm_usr_data_set	y	n/a	n/a	5.3.37
dwm_nvm_usr_data_get	y	n/a	n/a	5.3.38
dwm_gpio_irq_cfg	y	n/a	n/a	5.3.39
dwm_gpio_irq_dis	y	n/a	n/a	5.3.40
dwm_i2c_read	y	n/a	n/a	5.3.41
dwm_i2c_write	y	n/a	n/a	5.3.42
dwm_evt_listener_register	y	n/a	n/a	5.3.43
dwm_evt_wait	y	n/a	n/a	5.3.44
dwm_wake_up	y	n/a	n/a	5.3.45
dwm_bh_status_get	n	0x3a	0x00	5.4.1
dwm_backhaul_xfer	n	0x37	0x02	5.4.2

The definition and usage of the API functions are described below in individual sub-sections. In introducing each API function, all possible accesses are described, including c code, UART/SPI TLV.

Note: Some API functions are only provided in limited accesses.

Note: There are more TLV type than listed in Table 3. The TLV types that do not relate to API functions are not introduced here, e.g. the TLV response types. See Section 7 for the full TLV type list.

5.2 *Usage of the APIs*

Examples of how the UART Generic, SPI and C code API can be used is introduced in the DWM1001 Firmware User Guide [3]. The examples intend to give a brief view of the API usage and only include very simple API functions. The code in the source files of the examples can be changed to modify/add functionalities. The full list of API functions as shown in Table 3 can also be found in the included header files of the API source code packages. Table 4 lists the packages with examples of the corresponding APIs, the locations of the header files and the locations of the source files.

Table 4 API examples location

API	Package with examples	Header file location	Example source file location
C code	DWM1001 on-board package [5]	dwm\include\dwm.h	dwm\examples\dwm-simple\dwm-simple.c
UART Generic	DWM1001 Host API package [6]	include\dwm_api.h	examples\ex1_TWR_2Hosts\tag\tag_cfg.c
SPI	DWM1001 Host API package [6]	include\dwm_api.h	examples\ex1_TWR_2Hosts\tag\tag_cfg.c

5.3 *Details of the API functions*

The details of each API function listed in Table 3 are described in the below sub-sections.

5.3.1 dwm_pos_set

5.3.1.1 Description

This API function set the default position of the node. Default position is not used when in tag mode but is stored anyway so the module can be configured to anchor mode and use the value previously set by dwm_pos_set. This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds.

Parameter			Description
Field	**Name**	**Size**	
Input	position	13-byte array	Position information, see Section 4.4.2
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.1.2 C code

Declaration:

```
int dwm_pos_set(dwm_pos_t* p_pos);
```

Example:

```
dwm_pos_t pos;
pos.qf = 100;
pos.x = 121;
pos.y = 50;
pos.z = 251;
dwm_pos_set(&pos);
```

5.3.1.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x01	0x0D	position

Example:

TLV request					
Type	**Length**	**Value**			
		Position			
		32 bit value in little endian is x coordinate in millimeters	32 bit value in little endian is y coordinate in millimeters	32 bit value in little endian is z coordinate in millimeters	8 bit value is quality factor in percents (0-100)
0x01	0x0D	0x79 0x00 0x00 0x00	0x32 0x00 0x00 0x00	0xfb 0x00 0x00 0x00	0x64

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.2 dwm_pos_get

5.3.2.1 Description

This API function obtain position of the node. If the current position of the node is not available, the default position previously set by dwm_pos_set will be returned.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	position	13-byte array	position information, see Section 4.4.2

5.3.2.2 C code

Declaration:

```
int dwm_pos_set(dwm_pos_t* p_pos);
```

Example:

```
dwm_pos_t pos;
dwm_pos_get(&pos);
```

5.3.2.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x02	0x00

Example:

TLV request	
Type	**Length**
0x02	0x00

TLV response								
Type	**Length**	**Value**	**Type**	**Length**	**Value**			
		err_code			**Position**			
					32 bit value in little endian is x coordinate in millimeters	32 bit value in little endian is y coordinate in millimeters	32 bit value in little endian is z coordinate in millimeters	8 bit value is quality factor in percents (0-100)
0x40	0x01	0x00	0x41	0x0D	0x79 0x00 0x00 0x00 0x32 0x00 0x00 0x00 0xfb 0x00 0x00 0x00 0x64			

5.3.3 dwm_upd_rate_set

5.3.3.1 Description

This API function sets the update rate and the stationary update rate of the position in unit of 100 milliseconds. Stationary update rate must be greater or equal to normal update rate. This call does a write to the internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds.

Parameter			Description
Field	**Name**	**Size**	
Input	update_rate	16-bit integer	position publication interval in multiples of 100 milliseconds
	update_rate_stationary	16-bit integer	position publication interval when node is not moving in multiples of 100 milliseconds, maximum is 2 minutes, must be greater or equal to normal update_rate
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.3.2 C code

Declaration:

```
int dwm_upd_rate_set(uint16_t ur, uint16 urs);
```

Example:

```
dwm_upd_rate_set(10, 50); // update rate 1 second. 5 seconds stationary
```

5.3.3.3 SPI/UART Generic

Declaration:

TLV request			
Type	**Length**	**Value**	
0x03	0x04	update_rate	update_rate_stationary

Example:

TLV request			
Type	**Length**	**Value**	
		update_rate	**update_rate_stationary**
		16 bit value in little endian, which is update rate in multiples of 100 ms (e.g. 0x0A 0x00 means 10)	16 bit value in little endian, which is stationary update rate in multiples of 100 ms. E.g. 0x0A 0x00 means 5.0.
0x03	0x04	0x0A 0x00	0x32 0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.4 dwm_upd_rate_get

5.3.4.1 Description

This API function gets position update rate.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	update_rate	16-bit integer	position update interval in multiples of 100 milliseconds,
	update_rate_stationary	16-bit integer	Stationary position update interval in multiples of 100 milliseconds

5.3.4.2 C code

Declaration:

```
int dwm_upd_rate_get(uint16_t* p_ur, uint16_t* p_urs);
```

Example:

```
uint16_t ur, urs;

dwm_upd_rate_get(&ur, &urs);
```

5.3.4.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x04	0x00

Example:

TLV request	
Type	**Length**
0x04	0x00

TLV response						
Type	**Length**	**Value**	**Type**	**Length**	**Value**	
		err_code			**update_rate**, first 2 bytes, indicating the position update interval in multiples of 100 ms. E.g. 0x0A 0x00 means 1.0s interval.	**update_rate_stationary**, second 2 bytes, indicating the stationary position update interval in multiples of 100 ms. E.g. 0x32 0x00 means 5.0s interval.
0x40	0x01	0x00	0x45	0x04	0x0A 0x00	0x32 0x00

5.3.5 dwm_cfg_tag_set

5.3.5.1 Description

This API function configures the node as tag with given options.

BLE option can't be enabled together with encryption otherwise the configuration is considered invalid and it is refused. Encryption can't be enabled if encryption key is not set.

This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds. Note that this function only sets the configuration parameters. To make effect of the settings, users should issue a reset command (dwm_reset()), see section 5.3.13 for more detail.

Parameter			Description
Field	**Name**	**Size**	
Input	cfg_tag	16-bit integer	2-byte, configuration of the tag, see Section 4.4.7
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.5.2 C code

Declaration:

```
int dwm_cfg_tag_set(dwm_cfg_tag_t* p_cfg);
```

Example:

```
dwm_cfg_tag_t cfg;

cfg.stnry_en = 1;
cfg.meas_mode = DWM_MEAS_MODE_TWR;
cfg.low_power_en = 0;
cfg.loc_engine_en = 1;
cfg.common.enc_en = 1;
cfg.common.led_en = 1;
cfg.common.ble_en = 0;
cfg.common.fw_update_en = 0;
cfg.common.uwb_mode = DWM_UWB_MODE_ACTIVE;
dwm_cfg_tag_set(&cfg);
```

5.3.5.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
		(* BYTE 1 *) (bits 3-7) reserved (bit 2) stnry_en (bits 0-1) meas_mode : 0 - TWR, 1-3 reserved (* BYTE 0 *) (bit 7) low_power_en (bit 6) loc_engine_en (bit 5) enc_en (bit 4) led_en (bit 3) ble_en (bit 2) fw_update_en (bits 0-1) uwb_mode
0x05	0x02	cfg_tag

Example:

TLV request		
Type	**Length**	**Value**
		cfg_tag: low_power_en, loc_engine_en, ble_en, DWM_UWB_MODE_ACTIVE, fw_update_en
0x03	0x04	0xCE 0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.6 dwm_cfg_anchor_set

Description

Configure node as anchor with given options. BLE option can't be enabled together with encryption otherwise the configuration is considered invalid and it is refused.

Encryption can't be enabled if encryption key is not set. This call requires reset for new configuration to take effect (dwm_reset). Enabling encryption on initiator will cause automatic enabling of encryption of all nodes that have the same encryption key set (dwm_enc_key_set). This allows to enable encryption for whole network that has the same pan ID (network ID) and the same encryption key remotely from one place.

This call does a write to internal flash in case of new value being set, hence should not be used frequently and can take in worst case hundreds of milliseconds.

Parameter			Description
Field	**Name**	**Size**	
Input	cfg_anchor	8-bit integer	1-byte, configuration of the tag, see Section 4.4.8
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

C code

Declaration:

```
int dwm_cfg_anchor_set(dwm_cfg_anchor_t* p_cfg)
```

Example:

```
dwm_cfg_anchor_t cfg;
int rv;

cfg.initiator = 1;
cfg.bridge = 0;
cfg.common.enc_en = 1;
cfg.common.led_en = 1;
cfg.common.ble_en = 1;
cfg.common.fw_update_en = 1;
cfg.common.uwb_mode = DWM_UWB_MODE_OFF;
rv = dwm_cfg_anchor_set(&cfg);
if (rv == DWM_ERR_PERM)
printf("Error: either encryption or BLE can be enabled, encryption can be enabled only if encryption key is set\n");
dwm_reset();
```

SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
		(* BYTE 1 *) (bits 2-7) reserved (bits 0-1) reserved (* BYTE 0 *) (bit 7) initiator (bit 6) bridge (bit 5) enc_en (bit 4) led_en (bit 3) ble_en (bit 2) fw_update_en

		(bits 0-1) uwb_mode
0x07	0x02	cfg_anchor

Example:

TLV request		
Type	**Length**	**Value**
		cfg_anchor: initiator, led_en, ble_en, fw_update_en, UWB_MODE_ACTIVE
0x07	0x02	0x9e 0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.7 dwm_cfg_get

5.3.7.1 Description

This API function obtains the configuration of the node.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	cfg_node	16-bit integer	configuration of the node, see sections 4.4.7 and 4.4.8

5.3.7.2 C code

Declaration:

```
int dwm_cfg_get(dwm_cfg_t* p_cfg);
```

Example:

```
dwm_cfg_t cfg;

dwm_cfg_get(&cfg);

printf("mode %u \n", cfg.mode);
printf("initiator %u \n", cfg.initiator);
printf("bridge %u \n", cfg.bridge);
printf("motion detection enabled %u \n", cfg.stnry_en);
printf("measurement mode %u \n", cfg.meas_mode);
printf("low power enabled %u \n", cfg.low_power_en);
printf("internal location engine enabled %u \n", cfg.loc_engine_en);
printf("encryption enabled %u \n", cfg.common.enc_en);
printf("LED enabled %u \n", cfg.common.led_en);
printf("BLE enabled %u \n", cfg.common.ble_en);
printf("firmware update enabled %u \n", cfg.common.fw_update_en);
printf("UWB mode %u \n", cfg.common.uwb_mode);
```

5.3.7.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x08	0x00

Example:

TLV request	
Type	**Length**
0x08	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			(* BYTE 1 *) (bit 5) mode : 0 - tag, 1 - anchor (bit 4) initiator (bit 3) bridge (bit 2) stnry_en (bits 0-1) meas_mode : 0 - TWR, 1-3 not supported (* BYTE 0 *) (bit 7) low_power_en (bit 6) loc_engine_en (bit 5) enc_en

					(bit 4) led_en (bit 3) ble_en (bit 2) fw_update_en (bits 0-1) uwb_mode
0x40	0x01	0x00	0x46	0x02	0x07 0x04

5.3.8 dwm_sleep

5.3.8.1 Description

This API function puts the module into sleep mode. Low power option must be enabled otherwise an error will be returned.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.8.2 C code

Declaration:

```
int dwm_sleep(void);
```

Example:

```
/* THREAD 1: sleep and block*/
dwm_sleep();
/*do something*/
...

/*THREAD 2: wait until event */
dwm_evt_wait(&evt);
/*unblock dwm_sleep()*/
dwm_wake_up();
```

5.3.8.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x0A	0x00

Example:

TLV request	
Type	**Length**
0x0A	0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.9 dwm_anchor_list_get

5.3.9.1 Description

This API function reads list of surrounding anchors. Works for anchors only. Anchors in the list can be from the same network or from the neighbor network as well.

Parameter			Description
Field	**Name**	**Size**	
Input (for UART/SPI only)	page_number	8-bit integer	Page number, valid numbers are 0 and 1.
Input (for C code only)	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	cnt		

5.3.9.2 C code

Declaration:

```
int dwm_anchor_list_get(dwm_anchor_list_t* p_list);
```

Example:

```
dwm_anchor_list_t list;
int i;

dwm_anchor_list_get(&list);

for (i = 0; i < list.cnt; ++i) {
        printf("%d. id=0x%04X pos=[%ld,%ld,%ld] rssi=%d seat=%u neighbor=%d\n", i,
                                list.v[i].node_id,
                                list.v[i].x,
                                list.v[i].y,
                                list.v[i].z,
                                list.v[i].rssi,
                                list.v[i].seat,
                                list.v[i].neighbor_network);
}
```

5.3.9.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x0B	0x01	**page_number**

Example1:

TLV request		
Type	**Length**	**Value**
0x0B	0x01	0x01

TLV response		
Type	**Length**	**Value**
		error_code
0x40	0x01	0x00

TLV response (residue of the frame from previous table)							
Type	**Length**	**Value**					
		uint8_t - number of elements encoded in the value	uint16_t - UWB address in little endian	3 x int32_t: position coordinates x,y,z in little endian	int8_t - RSSI	uint8_t - (bits 0-4) seat number (bit 5) neighbor_network (bits 6-7) reserved	…
0x40	0x01	0x0F	anchor nr. 1				nr. 2 ... nr. 15

Example 2:

TLV request		
Type	**Length**	**Value**
0x0B	0x01	0x01

TLV request					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		error_code			uint8_t - number of elements encoded in the value
0x40	0x01	0x00	0x56	0x01	0x00

5.3.10 dwm_loc_get

5.3.10.1 Description

Get last distances to the anchors (tag is currently ranging to) and the associated position. The interrupt is triggered when all TWR measurements have completed and the LE has finished. If the LE is disabled, the distances will just be returned. This API works the same way in both Responsive and Low-Power tag modes.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	position	13-byte array	Position information of the current node, see Section 4.4.2
	dist.cnt	1-byte	Number of distances to the anchors, max 15.
	dist.addr	8-byte/2-byte	public UWB address in little endian. (1)
	dist.dist	4-byte	Distances to the anchors. (1)
	dist.qf	1-byte	Quality factor of distances to the anchors. (1)
	an_pos.cnt	1-byte	Number of anchor positions, max 15.
	an_pos	13-byte array	Anchor positions information, see Section 4.4.2. (2)

(1) This data can appear more than once according to the value of dist.cnt. This data is 8-byte long in C code API, 8-byte long in SPI/UART response for Anchor, and 2-byte long for SPI/UART response for Tag.
(2) This data can appear more than once according to the value of an_pos.cnt.

5.3.10.2 C code

Declaration:

```
int dwm_loc_get(dwm_loc_data_t* p_loc);
```

Example:

```
dwm_loc_data_t loc;
int rv, i;

/* if pos_available is false, position data are not read and function returns without error */
rv = dwm_loc_get(&loc);

if (0 == rv) {
        if (loc.pos_available) {
                printf("[%ld,%ld,%ld,%u] ", loc.pos.x, loc.pos.y, loc.pos.z,
                loc.pos.qf);
        }

        for (i = 0; i < loc.anchors.dist.cnt; ++i) {
                printf("%u)", i);
                printf("0x%04x", loc.anchors.dist.addr[i]);
                if (i < loc.anchors.an_pos.cnt) {
                        printf("[%ld,%ld,%ld,%u]", loc.anchors.an_pos.pos[i].x,
                                                   loc.anchors.an_pos.pos[i].y,
                                                   loc.anchors.an_pos.pos[i].z,
                                                   loc.anchors.an_pos.pos[i].qf);
                }

                printf("=%lu,%u ", loc.anchors.dist.dist[i], loc.anchors.dist.qf[i]);
        }
        printf("\n");
} else {
        printf("err code: %d\n", rv);
}
```

5.3.10.3 **SPI/UART Generic**

Declaration:

TLV request	
Type	**Length**
0x0C	0x00

Example 1 (Tag node):

TLV request	
Type	**Length**
0x0C	0x00

TLV response									
Type	**Length**	**Value**	**Type**	**Length**	**Value**				**Type**	**Length**	**Value**
		err_code			**position**, 13 bytes						1 byte, number of distances encoded in the value
					4-byte x	4-byte y	4-byte z	1-byte quality factor			
0x40	0x01	0x00	0x41	0x0D	0x4b 0x0a 0x00 0x00 0x1f 0x04 0x00 0x00 0x9c 0x0e 0x00 0x00 0x64				0x49	0x51	0x04

TLV response (residue of the frame from previous table)								
Value								
2 bytes UWB address	4-byte distance	1-byte distance quality factor	position in standard 13 byte format	…	2 bytes UWB address	4-byte distance	1-byte distance quality factor	position in standard 13 byte format
position and distance AN1				AN2, AN3	position and distance AN4			

Example 2 (Anchor node):

TLV request	
Type	**Length**
0x0C	0x00

TLV response									
Type	**Length**	**Value**	**Type**	**Length**	**Value**				**Type**	**Length**	**Value**
		err_code			**position**, 13 bytes						1 byte, number of distances encoded in the value
					4-byte x	4-byte y	4-byte z	1-byte quality factor			
0x40	0x01	0x00	0x41	0x0D	0x4b 0x0a 0x00 0x00 0x1f 0x04 0x00 0x00 0x9c 0x0e 0x00 0x00 0x64				0x48	0xC4	0x0F

TLV response (residue of the frame from previous table)						
Value						
8 bytes UWB address	4-byte distance	1-byte distance quality factor	…	8 bytes UWB address	4-byte distance	1-byte distance quality factor
distance AN1			AN2, …, AN14	distance AN15		

5.3.11 dwm_baddr_set

5.3.11.1 Description

Sets the public Bluetooth address used by device. New address takes effect after reset. This call does a write to internal flash in case of new value being set, hence should not be used frequently as can take, in worst case, hundreds of milliseconds.

Parameter			Description
Field	**Name**	**Size**	
Input	ble_addr	6-bytes	48-bit long public BluetoothE address in little endian.
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.11.2 C code

Declaration:

```
int dwm_baddr_set(dwm_baddr_t* p_baddr);
```

Example:

```
dwm_baddr_t baddr;
baddr.byte[0] = 1;
baddr.byte[1] = 2;
baddr.byte[2] = 3;
baddr.byte[3] = 4;
baddr.byte[4] = 5;
baddr.byte[5] = 6;
dwm_baddr_set(&baddr);
```

5.3.11.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x0F	0x06	ble_addr

Example:

TLV request		
Type	**Length**	**Value**
		ble_addr, 6 bytes in little endian
0x0F	0x06	0x01 0x23 0x45 0x67 0x89 0xab

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.12 dwm_baddr_get

5.3.12.1 Description

Get Bluetooth address currently used by device.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	ble_addr	6-bytes	48-bit long public Bluetooth address in little endian.

5.3.12.2 C code

Declaration:

```
int dwm_baddr_get(dwm_baddr_t* p_baddr);
```

Example:

```
dwm_baddr_t baddr;
int i;

if (DWM_OK == dwm_baddr_get(&baddr)) {
        printf("addr=");
        for (i = DWM_BLE_ADDR_LEN - 1; i >= 0; --i) {
                printf("%02x%s", baddr.byte[i], (i > 0) ? ":" : "");
        }
        printf("\n");
} else {
        printf("FAILED");
}
```

5.3.12.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x10	0x00

Example:

TLV request	
Type	**Length**
0x10	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**ble_addr**, 6 bytes in little endian
0x40	0x01	0x00	0x5F	0x06	0x01 0x23 0x45 0x67 0x89 0xab

5.3.13 dwm_stnry_cfg_set

5.3.13.1 Description

Writes configuration of the stationary mode which is used by tag node. The configuration can be written even if stationary detection is disabled (see dwm_cfg_tag_set). Writes internal nonvolatile memory so should be used carefully. New sensitivity setting takes effect immediately if stationary mode is enabled. Default sensitivity is “HIGH”.

Parameter			Description
Field	**Name**	**Size**	
Input	stnry_sensitivity	8-bit integer	0, 1 or 2, see Section 4.4.10
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.13.2 C code

Declaration:

```
int dwm_stnry_cfg_set(dwm_stnry_sensitivity_t sensitivity);
```

Example:

```
dwm_stnry_cfg_set(DWM_STNRY_SENSITIVITY_HIGH);
```

5.3.13.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x11	0x01	**stnry_sensitivity**

Example:

TLV request		
Type	**Length**	**Value**
		Stnry_sensitivity
0x11	0x01	0x01

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.14 dwm_stnry_cfg_get

5.3.14.1 Description

Reads configuration of the stationary mode which is used by tag node. The configuration can be read even if stationary detection is disabled (see dwm_cfg_tag_set).

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	stnry_sensitivity	8-bit integer	0, 1 or 2, see Section 4.4.10

5.3.14.2 C code

Declaration:

```
int dwm_stnry_cfg_get(dwm_stnry_sensitivity_t* p_sensitivity);
```

Example:

```
dwm_stnry_sensitivity_t sensitivity;

dwm_stnry_cfg_get(&sensitivity);
```

5.3.14.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x12	0x00

Example:

TLV request	
Type	**Length**
0x12	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**Stnry_sensitivity**
0x40	0x01	0x00	0x4A	0x01	0x01

5.3.15 dwm_factory_reset

5.3.15.1 Description

This API function puts node to factory settings. Environment is erased and set to default state. Resets the node. This call does a write to internal flash, hence should not be used frequently and can take in worst case hundreds of milliseconds.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.15.2 C code

Declaration:

```
int dwm_factory_reset(void);
```

Example:

```
dwm_factory_reset();
```

5.3.15.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x13	0x00

Example:

TLV request	
Type	**Length**
0x13	0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.16 dwm_reset

5.3.16.1 Description

This API function reboots the module.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.16.2 C code

Declaration:

```
int dwm_reset(void);
```

Example:

```
dwm_reset();
```

5.3.16.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x14	0x00

Example:

TLV request	
Type	**Length**
0x14	0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.17 dwm_ver_get

5.3.17.1 Description

This API function obtains the firmware version of the module.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	fw_version	4-byte array	firmware version, see Section 4.4.6
	cfg_version	4-byte array	configuration version, 32-bits integer
	hw_version	4-byte array	hardware version, 32-bits integer

5.3.17.2 C code

Declaration:

```
int dwm_ver_get(dwm_ver_t* p_ver);
```

Example:

```
dwm_ver_t ver;
dwm_ver_get(&ver);
```

5.3.17.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x15	0x00

Example:

TLV request	
Type	**Length**
0x15	0x00

TLV response								
Type	**Length**	**Value**	**Type**	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**fw_version**: maj = 1 min = 2 patch = 5 var = 1			**cfg_version**
0x40	0x01	0x00	0x50	0x04	0x01 0x05 0x02 0x01	0x51	0x04	0x00 0x07 0x01 0x00

TLV response (residue of the frame from previous table)		
Type	**Length**	**Value**
		hw_version
0x52	0x04	0x2a 0x00 0xca 0xde

5.3.18 dwm_uwb_cfg_set

5.3.18.1 Description

Sets UWB configuration parameters.

Parameter			Description
Field	**Name**	**Size**	
Input	pg_delay	1 byte	Transmitter Calibration – Pulse Generator Delay
	tx_power	4 bytes	TX Power Control
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.18.2 C code

Declaration:

```
int dwm_uwb_cfg_set(dwm_uwb_cfg_t *p_cfg);
```

Example:

```
dwm_uwb_cfg_t cfg;

cfg.pg_delay = 197;
cfg.tx_power = 0xD0252525;
dwm_uwb_cfg_set(&cfg);
```

5.3.18.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x17	0x05	**1st byte: pg_delay** **2nd -5th bytes: tx_power**

Example:

TLV request		
Type	**Length**	**Value**
		1st byte: pg_delay **2nd -5th bytes: tx_power**
0x17	0x05	0xC3 0x85 0x65 0x45 0x25

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.19 dwm_uwb_cfg_get

5.3.19.1 Description

Reads configuration of the stationary mode which is used by tag node. The configuration can be read even if stationary detection is disabled (see dwm_cfg_tag_set).

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	pg_delay	1 byte	Transmitter Calibration – Pulse Generator Delay
	tx_power	4 bytes	TX Power Control
	pg_delay_comp	1 byte	Transmitter Calibration – Pulse Generator Delay, compensated
	tx_power_comp	4 bytes	TX Power Control, compensated

5.3.19.2 C code

Declaration:

```
int dwm_uwb_cfg_get(dwm_uwb_cfg_t *p_cfg);
```

Example:

```
dwm_uwb_cfg_t uwb_cfg;

dwm_uwb_cfg_get(&uwb_cfg);

printf("delay=%x, power=%lx compensated(%x,%lx)\n",
                                    uwb_cfg.pg_delay,
                                    uwb_cfg.tx_power,
                                    uwb_cfg.compensated.pg_delay,
                                    uwb_cfg.compensated.tx_power);
```

5.3.19.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x18	0x00

Example:

TLV request	
Type	**Length**
0x18	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**1st byte: pg_delay** **2nd-5th byte: tx_power** **6th byte: pg_delay_comp** **7th-10th byte: tx_power_comp**
0x40	0x01	0x00	0x4A	0x01	0xC3 0x85 0x65 0x45 0x25 0xC4 0x85 0x65 0x45 0x25

5.3.20 dwm_usr_data_read

5.3.20.1 Description

Reads downlink user data from the node. The new data cause setting of dedicated flag in the status and also cause generation of an event in user application (see dwm_evt_listener_register).

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	Length	8-bit integer	Length of the user data
	Data	1-34 bytes	User data

5.3.20.2 C code

Declaration:

```
int dwm_usr_data_read(uint8_t* p_data, uint8_t* p_len);
```

Example:

```
uint8_t data[DWM_USR_DATA_LEN_MAX];
uint8_t len;

len = DWM_USR_DATA_LEN_MAX;
dwm_usr_data_read(data, &len);
```

5.3.20.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x19	0x00

Example:

TLV request	
Type	**Length**
0x19	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**label**, max 16 bytes
0x40	0x01	0x00	0x4B	0x22	0x01 0x02 0x03 … 0x21 0x22

5.3.21 dwm_usr_data_write

5.3.21.1 Description

Writes user data to be sent through uplink to the network.

Parameter			Description
Field	**Name**	**Size**	
Input	Length	8-bit integer	Length of the data
	data	1-34 bytes	Uplink data
	Overwrite	bool	Forced write. Will overwrite data that is not yet sent through uplink
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.21.2 C code

Declaration:

```
int dwm_usr_data_write(uint8_t* p_data, uint8_t len, bool overwrite);
```

Example:

```
uint8_t len, data[DWM_USR_DATA_LEN_MAX];

len = DWM_USR_DATA_LEN_MAX;
dwm_usr_data_write(data, len, false);
```

5.3.21.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x1A	**length**	**User_data**

Example:

TLV request			
Type	**Length**	**Value**	
		overwrite	**user_data**
0x1A	0x23	0x01	0x01 0x02 0x03 … 0x21 0x22

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.22 dwm_label_read

5.3.22.1 Description

Reads the node label.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	Length	8-bit integer	Length of the label
	label	0-16 bytes	Label

5.3.22.2 C code

Declaration:

```
int dwm_label_read(uint8_t* p_label, uint8_t* p_len);
```

Example:

```
uint8_t label[DWM_LABEL_LEN_MAX];
uint8_t len;

len = DWM_LABEL_LEN_MAX;
dwm_label_read(label, &len);
```

5.3.22.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x1c	0x00

Example:

TLV request	
Type	**Length**
0x1c	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**label**, max 16 bytes
0x40	0x01	0x00	0x4c	0x06	0x01 0x23 0x45 0x67 0x89 0xab

5.3.23 dwm_label_write

5.3.23.1 Description

Writes the node label.

Parameter			Description
Field	**Name**	**Size**	
Input	Length	8-bit integer	Length of the label
	label	0-16 bytes	Label
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.23.2 C code

Declaration:

```
int dwm_label_write(uint8_t* p_label, uint8_t len);
```

Example:

```
uint8_t len, label[DWM_LABEL_LEN_MAX];

len = DWM_LABEL_LEN_MAX;
rv = dwm_label_write(label, len);
if ( len == rv )
        printf("ok\n");
else
        printf("error, %d", rv);
```

5.3.23.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x1D	**length**	**label**

Example:

TLV request		
Type	**Length**	**Value**
		label
0x1D	0x06	0x01 0x23 0x45 0x67 0x89 0xab

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.24 dwm_gpio_cfg_output

5.3.24.1 Description

This API function configures a specified GPIO pin as an output and also sets its value to 1 or 0, giving a high or low digital logic output value.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	**Name**	**Size**	
Input	gpio_idx	8-bit integer	GPIO number, see Section 4.4.3 for valid values
	gpio_value	8-bit integer	0 or 1
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.24.2 C code

Declaration:

```
int dwm_gpio_cfg_output(dwm_gpio_idx_t idx, bool value);
```

Example:

```
dwm_gpio_cfg_output(DWM_GPIO_IDX_13, 1); // set pin 13 as output and to 1 (high voltage)
```

5.3.24.3 SPI/UART Generic

Declaration:

TLV request			
Type	**Length**	**Value**	
0x28	0x02	gpio_idx	gpio_value

Example:

TLV request			
Type	**Length**	**Value**	
		gpio_idx	**gpio_value**
0x28	0x02	0x0d	0x01

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.25 dwm_gpio_cfg_input

5.3.25.1 Description

This API function configure GPIO pin as input.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	**Name**	**Size**	
Input	gpio_idx	8-bit integer	GPIO number, see Section 4.4.3 for valid values.
	gpio_pull	8-bit integer	GPIO pull status, see Section 4.4.54.4.3 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.25.2 C code

Declaration:

```
int dwm_gpio_cfg_input(dwm_gpio_idx_t idx, dwm_gpio_pin_pull_t pull_mode);
```

Example:

```
dwm_gpio_cfg_input(DWM_GPIO_IDX_13, DWM_GPIO_PIN_PULLUP);
dwm_gpio_cfg_input(DWM_GPIO_IDX_9, DWM_GPIO_PIN_NOPULL);
dwm_gpio_cfg_input(DWM_GPIO_IDX_31, DWM_GPIO_PIN_PULLDOWN);
```

5.3.25.3 SPI/UART Generic

Declaration:

TLV request			
Type	**Length**	**Value**	
0x29	0x02	gpio_idx	gpio_pull

Example:

TLV request			
Type	**Length**	**Value**	
		gpio_idx	**gpio_pull,** 0 -no pull 1 -pull down 3 -pull up
0x29	0x02	0x0D	0x01

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.26 dwm_gpio_value_set

5.3.26.1 Description

This API function sets the value of the GPIO pin to high or low.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	**Name**	**Size**	
Input	gpio_idx	8-bit integer	GPIO number, see Section 4.4.3 for valid values.
	gpio_value	8-bit integer	GPIO value, see Section 0 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.26.2 C code

Declaration:

```
int dwm_gpio_value_set(dwm_gpio_idx_t idx, bool value);
```

Example:

```
dwm_gpio_value_set(DWM_GPIO_IDX_13, 1);
```

5.3.26.3 SPI/UART Generic

Declaration:

TLV request			
Type	**Length**	**Value**	
0x2a	0x02	gpio_idx	gpio_value

Example:

TLV request			
Type	**Length**	**Value**	
		gpio_idx	**gpio_value,** 0 - low 1 - high
0x2a	0x02	0x0D	0x01

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.27 dwm_gpio_value_get

5.3.27.1 Description

This API function reads the value of the GPIO pin.

Parameter			Description
Field	**Name**	**Size**	
Input	gpio_idx	8-bit integer	GPIO number, see Section 4.4.3 for valid values.
Output	gpio_value	8-bit integer	GPIO value, see Section 0 for valid values.

5.3.27.2 C code

Declaration:

```
int dwm_gpio_value_get(dwm_gpio_idx_t idx, bool* p_value);
```

Example:

```
uint8_t value;
dwm_gpio_value_get(DWM_GPIO_IDX_13, &value);
printf("DWM_GPIO_IDX_13 value = %u\n", value);
```

5.3.27.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x2b	0x01	gpio_idx

Example:

TLV request		
Type	**Length**	**Value**
		gpio_idx
0x2b	0x01	0x0D

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**gpio_value**
0x40	0x01	0x00	0x55	0x01	0x01

5.3.28 dwm_gpio_value_toggle

5.3.28.1 Description

This API function toggles the value of the GPIO pin.

Note: During the module reboot, the bootloader (as part of the firmware image) blinks twice the LEDs on GPIOs 22, 30 and 31 to indicate the module has restarted. Thus these GPIOs should be used with care during the first 1s of a reboot operation.

Parameter			Description
Field	**Name**	**Size**	
Input	gpio_idx	8-bit integer	GPIO number, see Section 4.4.3 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.28.2 C code

Declaration:

```
int dwm_gpio_value_toggle(dwm_gpio_idx_t idx);
```

Example:

```
dwm_gpio_value_toggle(DWM_GPIO_IDX_13);
```

5.3.28.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x2c	0x01	gpio_idx

Example:

TLV request		
Type	**Length**	**Value**
		gpio_idx
0x2c	0x01	0x0D

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.29 dwm_panid_set

5.3.29.1 Description

This API function sets UWB network identifier for the node. The ID is stored in nonvolatile memory.

Parameter			Description
Field	**Name**	**Size**	
Input	panid	2-byte unsigned integer	UWB panid
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.29.2 C code

Declaration:

```
int dwm_panid_set(uint16_t panid);
```

Example:

```
dwm_panid_set(0xABCD);
```

5.3.29.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x2e	0x02	**panid**

Example:

TLV request		
Type	**Length**	**Value**
		panid (little endian)
0x2E	0x02	0xCD 0xAB

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.30 dwm_panid_get

5.3.30.1 Description

This API function gets UWB network identifier from the node.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	panid	2-byte unsigned integer	UWB panid

5.3.30.2 C code

Declaration:

```
int dwm_panid_get(uint16_t *p_panid);
```

Example:

```
uint16_t panid;

if (DWM_OK == dwm_panid_get(&panid)) {
        printf("panid=%u\n");
} else {
        printf("FAILED\n");
}
```

5.3.30.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x2F	0x00

Example:

TLV request	
Type	**Length**
0x2F	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**Panid (little endian)**
0x40	0x01	0x00	0x4D	0x02	0x34 0x12

5.3.31 dwm_nodeid_get

5.3.31.1 Description

This API function gets UWB address of the node.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	nodeid	8-byte little endian	UWB node address/ID

5.3.31.2 C code

Declaration:

```
int dwm_node_id_get(uint64_t *p_node_id);
```

Example:

```
uint64_t node_id;

dwm_node_id_get(&node_id);

printf("node id:0x%llx \n", node_id);
```

5.3.31.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x30	0x00

Example:

TLV request	
Type	**Length**
0x30	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**nodeid (little endian)**
0x40	0x01	0x00	0x4E	0x08	0x99 0x0c 0x80 0x8d 0x63 0xef 0xca 0xde

5.3.32 dwm_status_get

5.3.32.1 Description

This API function reads the system status. Flags including:

- Location Data ready
- Node joined the UWB network
- New backhaul data ready
- Backhaul status has changed
- UWB scan result is ready
- User data over UWB received
- User data over UWB sent
- Firmware update in progress

All flags are cleared after the call.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	status	16-bit integer	**status**: bit 0: **loc_ready**: new location data are ready bit 1: **uwbmac_joined**: node is connected to UWB network bit 2: **bh_data_ready**: UWB MAC backhaul data ready bit 3: **bh_status_changed**: UWB MAC status has changed, used in backhaul bit 4: **reserved** bit 5: **uwb_scan_ready**: UWB scan results are ready bit 6: **usr_data_ready**: User data over UWB received bit 7: **usr_data_sent**: User data over UWB sent bit 8: **fwup_in_progress**: Firmware update is in progress bits 9-15 : reserved

5.3.32.2 C code

Declaration:

```
int dwm_status_get(dwm_status_t* p_status);
```

Example:

```
dwm_status_t status;
int rv;

rv = dwm_status_get(&status);
if (rv == DWM_OK) {
        printf("loc_data: %d\n", status.loc_data);
        printf("uwbmac_joined: %d\n", status.uwbmac_joined);
        printf("bh_data_ready: %d\n", status.bh_data_ready);
        printf("bh_status_changed: %d\n", status.bh_status_changed);
        printf("bh_initialized: %d\n", status.bh_initialized);
        printf("uwb_scan_ready: %d\n", status.uwb_scan_ready);
        printf("usr_data_ready: %d\n", status.usr_data_ready);
printf("usr_data_sent: %d\n", status.usr_data_sent);
printf("fwup_in_progress: %d\n", status.fwup_in_progress);
} else {
        printf("error\n");
}
```

5.3.32.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x32	0x00

Example:

TLV request	
Type	**Length**
0x32	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**loc_ready**: yes
0x40	0x01	0x00	0x5A	0x02	0x01 0x00

5.3.33 dwm_int_cfg_set

5.3.33.1 Description

Enables/disables setting of the dedicated GPIO pin in case of an event. Interrupts/events are communicated to the user by setting of GPIO pin CORE_INT1. User can use the pin as source of an external interrupt. The interrupt can be processed by reading the status (dwm_status_get) and react according to the new status. The status is cleared when read. This call is available only on UART/SPI interfaces. This call do a write to internal flash in case of new value being set, hence should not be used frequently and can take in worst case hundreds of milliseconds.

Parameter			Description
Field	**Name**	**Size**	
Input	int_cfg	16-bit integer	Interrupt config flags, see Section 4.4.9
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.33.2 C code

This command is not available for on-board user application. It is used only available on external interfaces (UART/SPI).

5.3.33.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x34	0x02	**int_cfg**

Example:

TLV request		
Type	**Length**	**Value**
		int_cfg spi_data_ready (bit 1) loc_ready (bit 0)
0x34	0x02	0x03 0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.34 dwm_int_cfg_get

5.3.34.1 Description

This API function reads the configuration flags that, if set, enables the setting of dedicated GPIO pin (CORE_INT) in case of an event internal to DWM module. This call is available only on UART/SPI interfaces.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	int_cfg	16-bit integer	Interrupt config flags, see Section 4.4.9

5.3.34.2 C code

This command is not available for on-board user application. It is used only available on external interfaces (UART/SPI).

5.3.34.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x35	0x00

Example:

TLV request	
Type	**Length**
0x35	0x00

TLV response		
Type	**Length**	**Value**
		int_cfg
0x40	0x02	0x03 0x00

5.3.35 dwm_enc_key_set

5.3.35.1 Description

This API function Sets encryption key. The key is stored in non-volatile memory. The key that consists of just zeros is considered as invalid. If key is set, the node can enable encryption automatically. Automatic enabling of the encryption is triggered via UWB network when the node detects encrypted message and is capable of decrypting the messages with the key. BLE option is disabled when encryption is enabled automatically. The encryption can be disabled by clearing the key (dwm_enc_key_clear).

This call writes to internal flash in case of new value being set, hence should not be used frequently and can take in worst case hundreds of milliseconds! Requires reset for new configuration to take effect.

Parameter			Description
Field	**Name**	**Size**	
Input	enc_key	16-byte array	128 bit encryption key
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.35.2 C code

Declaration:

```
int dwm_enc_key_set(dwm_enc_key_t* p_key);
```

Example:

```
dwm_enc_key_t key;

key.byte[0] = 0x00;
key.byte[1] = 0x11;
key.byte[2] = 0x22;
...
key.byte[15] = 0xFF;
dwm_enc_key_set(&key)
```

5.3.35.3 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x3C	0x10	enc_key

Example:

TLV request		
Type	**Length**	**Value**
		enc_key
0x3C	0x10	0x00 0x11 0x22 0x33 0x44 0x55 0x66 0x77 0x88 0x99 0xaa 0xbb 0xcc 0xdd 0xee 0xff

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.36 dwm_enc_key_clear

5.3.36.1 Description

This API function clears the encryption key and disables encryption option if enabled. Does nothing if the key is not set.

This call writes to internal flash in case of new value being set, hence should not be used frequently and can take in worst case hundreds of milliseconds! Requires reset for new configuration to take effect.

Parameter			Description
Field	**Name**	**Size**	
Input	none		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.36.2 C code

Declaration:

```
int dwm_enc_key_clear(void);
```

Example:

```
rv = dwm_enc_key_clear();
if (rv == DWM_ERR_INTERNAL)
printf("internal error\n");
```

5.3.36.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x3D	0x00

Example:

TLV request	
Type	**Length**
0x3D	0x00

TLV response		
Type	**Length**	**Value**
		err_code
0x40	0x01	0x00

5.3.37 dwm_nvm_usr_data_set

5.3.37.1 Description

Stores user data to non-volatile memory. Writes internal non-volatile memory so should be used carefully. Old data are overwritten.

Parameter			Description
Field	**Name**	**Size**	
Input	Length	8-bit integer	Length of the data
	data	Max 250 bytes	User data to be written
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.37.2 C code

Declaration:

```
int dwm_nvm_usr_data_set(uint8_t* p_data, uint8_t len);
```

Example:

```
uint8_t buf[DWM_NVM_USR_DATA_LEN_MAX];
uint8_t len = DWM_NVM_USR_DATA_LEN_MAX;

rv = dwm_nvm_usr_data_set(buf, len);
if ( DWM_OK == rv )
        printf("ok\n");
else
        printf("error, %d", rv);
```

5.3.37.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.38 dwm_nvm_usr_data_get

5.3.38.1 Description

Reads user data from non-volatile memory. Reads from internal flash.

Parameter			Description
Field	**Name**	**Size**	
Input	p_Length	Pointer of length	
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	Length	8-bit integer	Length of the user data
	Data	1-34 bytes	User data

5.3.38.2 C code

Declaration:

```
int dwm_usr_data_read(uint8_t* p_data, uint8_t* p_len);
```

Example:

```
uint8_t data[DWM_USR_DATA_LEN_MAX];
uint8_t len;

len = DWM_USR_DATA_LEN_MAX;
dwm_usr_data_read(data, &len);
```

5.3.38.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.39 dwm_gpio_irq_cfg

5.3.39.1 Description

This API function registers GPIO pin interrupt call back functions.

Parameter			Description
Field	**Name**	**Size**	
Input	gpio_idx	8-bit integer	GPIO number, see Section 4.4.3 for valid values.
	Irq_type	8-bit integer	interrupt type, 1 = rising, 2 = falling, 3 = both.
	p_cb	Function pointer	Pointer to the callback function on the interrupt.
	p_data	Data pointer	Pointer to data passed to the callback function.
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.39.2 C code

Declaration:

```
int dwm_gpio_irq_cfg(dwm_gpio_idx_t idx, dwm_gpio_irq_type_t irq_type, dwm_gpio_cb_t* p_cb, void* p_data);
```

Example:

```
void gpio_cb(void* p_data)
{
        /* callback routine */
}

dwm_gpio_irq_cfg(DWM_GPIO_IDX_13, DWM_IRQ_TYPE_EDGE_RISING, &gpio_cb, NULL);
```

5.3.39.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.40 dwm_gpio_irq_dis

5.3.40.1 Description

This API function disables GPIO pin interrupt on the selected pin.

Parameter			Description
Field	**Name**	**Size**	
Input	gpio_idx	8-bit integer	GPIO number, see Section 4.4.3 for valid values.
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.40.2 C code

Declaration:

```
int dwm_gpio_irq_dis(dwm_gpio_idx_t idx);
```

Example:

```
dwm_gpio_irq_dis(DWM_GPIO_IDX_13);
```

5.3.40.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.41 dwm_i2c_read

5.3.41.1 Description

This API function read data from I2C slave.

Parameter			Description
Field	**Name**	**Size**	
Input	addr	8-bit integer	Address of a slave device (only 7 LSB)
	p_data	8-bit integer	Pointer to a receive data buffer
	len	8-bit integer	Number of bytes to be received
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.41.2 C code

Declaration:

```
int dwm_i2c_read(uint8_t addr, uint8_t* p_data, uint8_t len);
```

Example:

```
uint8_t data[2];
const uint8_t addr = 0x33; // some address of the slave device

dwm_i2c_read(addr, data, 2);
```

5.3.41.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.42 dwm_i2c_write

5.3.42.1 Description

This API function writes data to I2C slave.

Parameter			Description
Field	**Name**	**Size**	
Input	addr	8-bit integer	Address of a slave device (only 7 LSB)
	p_data	8-bit integer	Pointer to a receive data buffer
	len	8-bit integer	Number of bytes to be received
	no_stop	8-bit integer	0 or 1. If set to 1, the stop condition is not generated on the bus after the transfer has completed (allowing for a repeated start in the next transfer)
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.42.2 C code

Declaration:

```
int dwm_i2c_write(uint8_t addr, uint8_t* p_data, uint8_t len, bool no_stop);
```

Example:

```
uint8_t data[2];
const uint8_t addr = 1; // some address of the slave device

data[0] = 0xAA;
data[1] = 0xBB;
dwm_i2c_write(addr, data, 2, true);
```

5.3.42.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.43 dwm_evt_listener_register

5.3.43.1 Description

Registers events listener. User application can wait for the events that are registered to listen to by dwm_evt_wait. The event can be triggered for example when LE finishes position calculation and when distances are calculated. This call applies only for end user application. Can not be used with SPI or UART. In low power mode in order to wake up from the sleep, the event listener has to be registered, otherwise the user application will remain sleeping.

Parameter			Description
Field	**Name**	**Size**	
Input	event_id_map	32-bit wide bitmap	See Section 4.4.11 (evt_id_map)
	user_context_data	Data	data that the user wants inside the callback context
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1

5.3.43.2 C code

Declaration:

```
void dwm_evt_listener_register(uint32_t evt_id_map, void* p_context);
```

Example:

```
dwm_evt_listener_register(DWM_EVT_LOC_READY | DWM_EVT_USR_DATA_READY | DWM_EVT_USR_DATA_SENT |
DWM_EVT_UWB_SCAN_READY | DWM_EVT_BH_INITIALIZED | DWM_EVT_UWBMAC_JOINED_CHANGED, NULL);
```

5.3.43.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.44 dwm_evt_wait

5.3.44.1 Description

Used to wait for an event from DWM module. The event listener must be first registered by dwm_evt_listener_register. If event listener is registered and dwm_evt_wait is not used to consume events the event buffer will overflow. When there are no events in the buffer, the dwm_evt_wait will block and sleep until next event.

Parameter			Description
Field	**Name**	**Size**	
Input	None		
Output	err_code	8-bit integer	0, 1 or 2, see Section 4.4.1
	event_id_map	32-bit wide bitmap	See Section 4.4.11 (evt_id_map)

5.3.44.2 C code

Declaration:

```
int dwm_evt_wait(dwm_evt_t *p_evt);
```

Example:

```
void on_dwm_evt(dwm_evt_t *p_evt)
{
	int i;

	switch (p_evt->header.id) {
	/* New location data */
	case DWM_EVT_LOC_READY:
		printf("\nT:%lu ", dwm_systime_us_get());
		if (p_evt->loc.pos_available) {
			printf("POS:[%ld,%ld,%ld,%u] ", p_evt->loc.pos.x,
					p_evt->loc.pos.y, p_evt->loc.pos.z,
					p_evt->loc.pos.qf);
		} else {
			printf("Location engine is disabled\n");
		}

		for (i = 0; i < p_evt->loc.anchors.dist.cnt; ++i) {
			printf("DIST%d:", i);

			printf("0x%04X", (unsigned int)(p_evt->loc.anchors.dist.addr[i] & 0xffff));
			if (i < p_evt->loc.anchors.an_pos.cnt) {
				printf("[%ld,%ld,%ld]",
						p_evt->loc.anchors.an_pos.pos[i].x,
						p_evt->loc.anchors.an_pos.pos[i].y,
						p_evt->loc.anchors.an_pos.pos[i].z);
			}

			printf("=[%lu,%u] ", p_evt->loc.anchors.dist.dist[i],
					p_evt->loc.anchors.dist.qf[i]);
		}
		printf("\n");
		break;

	case DWM_EVT_USR_DATA_READY:
		printf("hex:");
		for (i = 0; i < p_evt->header.len - sizeof(dwm_evt_hdr_t); ++i) {
			printf("%02x ", p_evt->usr_data[i]);
		}
		printf("\n");
		break;

	case DWM_EVT_USR_DATA_SENT:
		printf("iot sent\n");
		break;
```

```
		case DWM_EVT_UWB_SCAN_READY:
			printf("[mode,rssi]: ");
			for (i = 0; i < p_evt->uwb_scan.cnt; ++i) {
				printf("[%u, %d]", p_evt->uwb_scan.mode[i], p_evt->uwb_scan.rssi[i]);
			}
			printf("\n");
			break;

		case DWM_EVT_BH_INITIALIZED_CHANGED:
			printf("backhaul available = %d\n", p_evt->bh_initialized);
			break;

		case DWM_EVT_UWBMAC_JOINED_CHANGED:
			printf("UWBMAC joined = %d\n", p_evt->uwbmac_joined);
			break;

		default:
			break;
		}

		/* Indicate the application has finished the tasks and can now */
		dwm_sleep();
}
...
int rv;
dwm_evt_t evt;

rv = dwm_evt_wait(&evt);

if (rv == DWM_ERR_OVERRUN) {
		printf("event buffer overflow\n");
} else {
		on_dwm_evt(&evt);
}
```

5.3.44.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.3.45 dwm_wake_up

5.3.45.1 Description

Prevents entering of the sleep state (if in low power mode). Should be called only from the thread context. Wakes up dwm_sleep().

5.3.45.2 C code

Declaration:

```
int dwm_wake_up(void);
```

Example:

```
/* THREAD 1: sleep and block*/
dwm_sleep();
/*do something*/
...

/*THREAD 2: wait until event */
dwm_evt_wait(&evt);
/*unblock dwm_sleep()*/
dwm_wake_up();
```

5.3.45.3 SPI/UART Generic

This command is not available on external interfaces (UART/SPI).

5.4 Backhaul API functions

This section describes API commands that are used to periodically transfer data over SPI interface when the DWM module is configured as “bridge” (see API call dwm_cfg_anchor_set).

5.4.1 dwm_bh_status_get

5.4.1.1 Description

Get current UWBMAC backhaul status. The node must be configured as bridge.

Parameter				Description
Field	**Name**		**Size**	
Input	None			
Output	err_code		8-bit integer	0, 1 or 2, see Section 4.4.1
	sf_number		16-bit integer	current superframe number
	bh_seat_map		32-bit integer	seat map in the current superframe
	origin_cnt		8-bit integer	range is from 0 to 8
	origin _info	node_id	16-bit integer	origin address
		bh_seat	8-bit integer	seat that the origin occupies, range from 0 to 8
		hop_lvl	8-bit integer	hop level

5.4.1.2 C code

Not available for user application.

5.4.1.3 SPI/UART Generic

Declaration:

TLV request	
Type	**Length**
0x3A	0x00

Example:

TLV request	
Type	**Length**
0x3A	0x00

TLV response					
Type	**Length**	**Value**	**Type**	**Length**	**Value**
		err_code			**sf_number, bh_seat_map and node_id are in little endian**
0x40	0x01	0x00	0x5D	0x13	0x6c 0x00 0x07 0x00 0x00 0x00 0x03 0xf3 0x11 0x00 0x01 0xc3 0x11 0x01 0x01 0x66 0x11 0x02 0x01

5.4.2 dwm_backhaul_xfer

5.4.2.1 Description

Writes downlink data and reads uplink data chunks. The DWM module must be configured as bridge. This API must be used with SPI Scheme: TLV communication using data ready pin.

Both the uplink and the downlink data are encoded into TLV frames when transferred by SPI interface as described in SPI Scheme: TLV communication using data ready pin.

SPI master tells slave how many downlink bytes it wants to transfer by **downlink_byte_cnt**. The **downlink_byte_cnt** is read by slave in first SPI transfer. Slave has some uplink data ready that it wants to transfer to the master as it is reading the downlink. In order to transfer both the downlink from the master to the slave and the uplink from the slave to the master, the slave has to calculate how many bytes and how many SPI transfers are needed. The master reads SIZE (the number of the bytes) and NUM (the number of the transfers) in the second SPI transfer as explained in SPI Scheme: TLV communication using data ready pin. Finally, the transfers are executed and both uplink and downlink are transferred. Maximum number of transfers currently supported is 5 with maximum payload 253 bytes, which is 255 - sizeof(TLV header). At most 5 uplink frames and at most 2 downlink frames are supported in one call to dwm_backhaul_xfer.

TLV types 100-104 (0x64-0x68) are reserved for uplink data chunks. TLV types 110-114 (0x6E-0x72) are reserved for downlink data chunks.

Parameter			**Description**
Field	**Name**	**Size**	
Input	**downlink_byte_cnt**	16 bit unsigned integer	Number of downlink bytes without TLV header, 506 bytes max
	downlink_chunk	Max 2 downlink chunks Each max 253 bytes	Opaque data send as downlink to the slave
	dummy_chunk	Max 4 downlink chunks Each max 253 bytes	Dummy Master Out data sent for Slave In uplink data.
Output	**SIZE**	1 byte unsigned integer	Number of each data chunk
	NUM	1 byte unsigned integer	Number of transfers the
	uplink_chunk	Max 5 uplink chunks Each max 253 bytes	Opaque data send as uplink to the master

5.4.2.2 C code

Not available for user application.

5.4.2.3 Uart Generic

Not available for UART interface.

5.4.2.4 SPI/UART Generic

Declaration:

TLV request		
Type	**Length**	**Value**
0x37	0x02	**downlink_byte_cnt = size of downlink data**

Example 1: communication from SPI master (normally the RPi) to node other than Bridge (as SPI slave)

TLV request

Type	Length	Value
0x37	0x02	0xf4 0x00

TLV response		
Type	Length	Value
		err_code
0x40	0x01	0x00

Example 2: communication from SPI master (normally the RPi) with Bridge (as SPI slave)

Downlink bytes count: 244
Uplink bytes count: 980: SIZE = 255, NUM = 4

TLV request		
Type	Length	Value
0x37	0x02	0xF4 0x00

Response	
SIZE	NUM
0xFF	0x04

On receiving SIZE = 255 and NUM= 4, the SPI Master initiates 4 transfers, each of 255 bytes.

Master Out data: TLV downlink number 1, 2, 3, 4			
Type	Length	Value	Dummy bytes
0x64	0xF4	0x01 0x02 … 0xf4 (244 in total)	0xff 0xff … 0xff (9 in total)
0xff 0xff…0xff (255 in total)			
0xff 0xff…0xff (255 in total)			
0xff 0xff…0xff (255 in total)			

Slave Out data: TLV uplink number 1, 2, 3, 4			
Type	Length	Value	
0x6E	0xFD	0x01 0x02 … 0xf4 (253 in total)	
0x6F	0xFD	0x01 0x02 … 0xf4 (253 in total)	
0x70	0xFD	0x01 0x02 … 0xf4 (253 in total)	
0x71	0xDD	0x01 0x02 … 0xf4 (221 in total)	0xff 0xff … 0xff (dummy bytes: 32 in total)

6 SHELL COMMANDS

6.1 Usage of UART Shell Mode

Shell mode shares UART interface with Generic mode. DWM1001 starts by default in UART Generic mode after reset. The Shell mode can be switched to by pressing ENTER twice within 1 second. The Generic mode can be switched to by executing command “quit” when in Shell mode. Shell mode and Generic mode can be switched back and forth.
Enter the Shell command and press “Enter” to execute the command. Press “Enter” without any command in Shell mode to repeat the last command. The following sub-sections provides overview of the Shell commands.

6.2 ?

Displays help.

Example:

```
dwm> ?
Usage: <command> [arg0] [arg1]
Build-in commands:

** Command group: Base **
?: this help
help: this help
quit: quit

** Command group: GPIO **
gc: GPIO clear
gg: GPIO get
gs: GPIO set
gt: GPIO toggle

** Command group: SYS **
f: Show free memory on the heap
ps: Show running threads
pms: Show PM tasks
reset: Reboot the system
si: System info
ut: Show device uptime
frst: Factory reset

** Command group: SENS **
twi: General purpose TWI read
aid: Read ACC device ID
av: Read ACC values

** Command group: LE **
les: Show meas. and pos.
lec: Show meas. and pos. in CSV
lep: Show pos. in CSV
```

```
** Command group: UWBMAC **
nmg: Get node mode
nmp: Set mode to PN (passive)
nmo: Set mode to PN (off)
nma: Set mode to AN
nmi: Set mode to AIN
nmt: Set mode to TN
nmtl: Set mode to TN-LP
bpc: Toggle BW/TxPWR comp
la: Show AN list
stg: Get stats
stc: Clear stats

** Command group: API **
tlv: Send TLV frame
aurs: Set upd rate
aurg: Get upd rate
apg: Get pos
aps: Set pos
acas: Set anchor config
acts: Set tag config

** Tips **
Press Enter to repeat the last command
```

6.3 *help*

Displays help, same as command “?”.

Example:

```
dwm> help
… /*same output as command ? */
```

6.4 *quit*

Quit shell and switch UART into API mode.

Example:

```
dwm> quit
/* press enter twice to switch to shell mode again*/
```

6.5 *gc*

Clears GPIO pin. See section 4.4.3 for valid GPIO number.

Example:

```
dwm> gc
Usage: gc <pin>
dwm> gc 13
```

gpio13: 0

6.6 gg

Reads GPIO pin level. See section 4.4.3 for valid GPIO number.

Example:

dwm> gg
Usage: gg <pin>
dwm> gg 13
gpio13: 0

6.7 gs

Sets GPIO as output and sets its value. See section 4.4.3 for valid GPIO number.

Example:

dwm> gs
Usage: gs <pin>
dwm> gs 13
gpio13: 1

6.8 gt

Toggles GPIO pin (must be output). See section 4.4.3 for valid GPIO number.

Example:

dwm> gt
Usage: gt <pin>
dwm> gt 13
gpio13: 0

6.9 f

Show free memory on the heap.

Example:

dwm> f
[000014.560 INF] mem: free=3888 alloc=9184 tot=13072

6.10 reset

Reboot the system.

Example:

dwm> reset
/* node resets and boots in binary mode */

6.11 ut

Show device uptime.

Example:

dwm> ut
[000003.680 INF] uptime: 00:07:49.210 0 days (469210 ms)

6.12 *frst*

Factory reset.

Example:

dwm> frst

6.13 *twi*

General purpose I2C/TWI read.

Example: use twi to read accelerometer id. This should return the same value as using “aid” command. See section 6.14.

dwm> twi
Usage: twi <addr> <reg> [bytes to read (1 or 2)]
dwm> twi 0x33 0x0f 1
twi: addr=0x33, reg[0x0f]=0x33

6.14 *aid*

Read ACC device ID

Example:

dwm> aid
acc: 0x33

6.15 *av*

Read ACC values.

Example:

dwm> av
acc: x = 240, y = -3792, z = 16240
dwm> av
acc: x = 32, y = -3504, z = 15872
dwm> av
acc: x = 160, y = -3600, z = 16144

6.16 *scs*

Stationary configuration set. Sets sensitivity (valid args are 0,1,2).

Example:

dwm> scs 1
ok

dwm> scs 4
error: invalid arg

6.17 scg

Stationary configuration get.

Example:

dwm> scg
sensitivity=1

6.18 les

Show distances to ranging anchors and the position if location engine is enabled. Sending this command multiple times will turn on/off this functionality.

Example:

dwm> les
1151[5.00,8.00,2.25]=6.48 0CA8[0.00,8.00,2.25]=6.51 111C[5.00,0.00,2.25]=3.18
1150[0.00,0.00,2.25]=3.16 le_us=2576 est[2.57,1.98,1.68,100]

6.19 lec

Show distances to ranging anchors and the position if location engine is enabled in CSV format. Sending this command multiple times will turn on/off this functionality.

Example:

dwm> lec
DIST,4,AN0,1151,5.00,8.00,2.25,6.44,AN1,0CA8,0.00,8.00,2.25,6.50,AN2,111C,5.00,0.00,2.25,
3.24,AN3,1150,0.00,0.00,2.25,3.19,POS,2.55,2.01,1.71,98

6.20 lep

Show position in CSV format. Sending this command multiple times will turn on/off this functionality.

Example:

dwm> lep
POS,2.57,2.00,1.67,97

6.21 utpg

Get transmit power of the DW1000.

Example:

dwm> utpg
utpg: pg_delay=xC5 tx_power=x29496989 (pg_delay=xC4 tx_power=x29496989)
/* Calibration values from the OTP/settings before the brackets. The current values are displayed in the brackets. */

6.22 utps

Set transmit power of the DW1000.

Example:

```
dwm> utps
Usage: utps <pg_delay> <tx_power>
dwm> utps 0xc2 0x28486888
utps: pg_delay=xC2 tx_power=x28486888
```

6.23 si

System info

Example:

```
dwm> si
[001762.590 INF] sys: fw2 fw_ver=x01020001 cfg_ver=x00010700
[001762.590 INF] uwb0: panid=x1234 addr=xDECADF01465011E4
[001762.590 INF] mode: ani (act,real)
[001762.600 INF] uwbmac: connected
[001762.600 INF] uwbmac: bh connected
[001762.610 INF] cfg: sync=1 fwup=0 ble=1 leds=1 init=1 bh=0 upd_rate_stat=64
label=DW11E4
[001762.620 INF] enc: off
[001762.630 INF] ble: addr=CB:65:C2:DD:D3:D4
```

6.24 nmg

Get node mode info.

Example:

```
dwm> nmg
mode: tn (act,twr,np,nole)
```

6.25 nmo

Enable passive offline option and resets the node.

Example:

```
dwm> nmo
/* press enter twice to switch to shell mode after reset*/
DWM1001 TWR Real Time Location System

Copyright :  2016-2017 LEAPS and Decawave
License   :  Please visit https://decawave.com/dwm1001_license
Compiled  :  Jul  3 2017 12:33:40

Help      :  ? or help

dwm> nmg
mode: tn (off,twr,np,nole)
```

6.26 nma

Configures node to as anchor, active and reset the node.

Example:

```
dwm> nma
/* press Enter twice */

DWM1001 TWR Real Time Location System

Copyright :  2016-2017 LEAPS and Decawave
License   :  Please visit https://decawave.com/dwm1001_license
Compiled  :  Jul  3 2017 12:33:40

Help      :  ? or help

dwm> nmg
mode: an (act,-,-)
```

6.27 nmi

Configures node to as anchor - initiator, active and reset the node.

Example:

```
dwm> nmi
/* press enter twice */

DWM1001 TWR Real Time Location System

Copyright :  2016-2017 LEAPS and Decawave
License   :  Please visit https://decawave.com/dwm1001_license
Compiled  :  Jul  3 2017 12:33:40

Help      :  ? or help

dwm> nmg
mode: ain (act,real,-)
```

6.28 nmt

Configures node to as tag, active and reset the node.

Example:

```
dwm> nmt
/* press enter twice */
DWM1001 TWR Real Time Location System

Copyright :  2016-2017 LEAPS and Decawave
License   :  Please visit https://decawave.com/dwm1001_license
Compiled  :  Jul  3 2017 12:33:40

Help      :  ? or help

dwm> nmg
```

mode: tn (act,twr,np,le)

6.29 nmtl

Configures node to as tag, active, low power and resets the node.

Example:

```
dwm> nmtl
/* press enter twice */
DWM1001 TWR Real Time Location System

Copyright :  2016-2017 LEAPS and Decawave
License   :  Please visit https://decawave.com/dwm1001_license
Compiled  :  Jul  3 2017 12:33:40

Help      :  ? or help

dwm> nmg
mode: tn (act,twr,lp,le)
```

6.30 nmb

Set mode to BN

6.31 la

Show anchor list.

Example:

```
dwm> la
[003452.590 INF] AN: cnt=4 seq=x03
[003452.590 INF]   0) id=DECA5419E2E01151 seat=0 idl=0 seens=0 col=0 cl=000F nbr=000F
fl=4002 opt=03C1B046 hw=DE41010
0 fw=01010101 fwc=7B033F01
[003452.600 INF]   1) id=DECA78A55EB00CA8 seat=1 idl=1 seens=119 col=0 cl=000F nbr=0000
fl=0000 opt=03C1B042 hw=DE410
100 fw=01010101 fwc=7B033F01
[003452.620 INF]   2) id=DECAFCED8D50111C seat=2 idl=1 seens=103 col=0 cl=000F nbr=0000
fl=0000 opt=03C1B042 hw=DE410
100 fw=01010101 fwc=7B033F01
[003452.640 INF]   3) id=DECA51421A901150 seat=3 idl=0 seens=81 col=0 cl=000F nbr=0000
fl=0000 opt=03C1B042 hw=DE4101
00 fw=01010101 fwc=7B033F01
```

6.32 lb

Show BN list

Example:

```
dwm> lb
[007922.440 INF] BN: cnt=2 seq=x01
```

```
[007922.440 INF] 0) id=DECAAE5D14830CB2 seat=1 seens=0 rssi=-127
[007922.450 INF] 1) id=0000000000000D35 seat=2 seens=170 rssi=-82
[007922.450 INF]
```

6.33 *nis*

Set Network ID

6.34 *nls*

Set node label

6.35 *stg*

Displays statistics.

Statistic	**Description**
uptime	System time since restart in seconds
rtc_drift	Estimated RTC drift against the UWB network clock (used during production)
ble_con_ok	Each BLE connect events increments this counter
ble_dis_ok	Each BLE disconnect events increments this counter
ble_err	Number that identifies last internal BLE error
api_err	Number that identifies last internal API error
api_err_cnt	Counter of errors on API
api_dl_cnt	Number of received backhaul DownLink packets via API (BN only)
uwb0_intr	Number of interrupts from the DW1000
uwb0_rst	Number of attempts to reset the DW1000 to recover from error
uwb0_bpc: 1	Number of bandwidth/temperature compensation
rx_ok	Number of enabling the reception on time
rx_err	Number of failures to enable the reception on time
tx_err	Number of failures to enable the transmission on time
tx_errx	Number of errors related with the TX buffer
bcn_tx_ok	Number of transmitted beacons
bcn_tx_err	Number of failures during transmission of beacons
bcn_rx_ok	Number of received beacons
alma_tx_ok	Number of transmitted almanacs
alma_tx_err	Number of failures during transmission of almanacs
alma_rx_ok	Number of received almanacs
cl_rx_ok	Number of received cluster join
cl_tx_ok	Number of transmitted cluster join
cl_coll	Number of detected cluster collisions

fwup_tx_ok	Number of transmitted firmware update frames
fwup_tx_err	Number of failures to transmit firmware update frames
fwup_rx_ok	Number of received firmware update frames
svc_tx_ok	Number of transmitted service frames
svc_tx_err	Number of failures to transmit service frames
svc_rx_ok	Number of received service frames
clk_sync	Number of times the node has synchronized
bh_rt	Number of times the AN was routing during routing table switch
bh_nort	Number of times the AN was not routing during routing table switch
bh_ev	Number of events sent to the DWM Kernel Module
bh_buf_lost[0]	Number of lost buffers ready for the Kernel Module
bh_buf_lost[1]	Number of lost buffers ready for the Kernel Module
bh_tx_err	Number of failures to transmit backhaul frames
bh_dl_err	Number of failures during processing of downlink backhaul frames
bh_dl_ok	Number of received downlink backhaul frames
bh_ul_err	Number of failures during processing of uplink backhaul frames
bh_ul_ok	Number of received uplink backhaul frames
fw_dl_tx_err	Number of failures during sending downlink data to the edge nodes
fw_dl_iot_ok	Number of sent downlink data to the edge nodes
fw_ul_loc_ok	Number of received uplink location data from the edge nodes
fw_ul_iot_ok	Number of received uplink IoT data from the edge nodes
ul_tx_err	Number of failures during sending uplink data from the edge node
dl_iot_ok	Number of sent downlink data to the edge node
ul_loc_ok	Number of receive uplink location data from the edge node
ul_iot_ok	Number of received uplink IoT data from the edge nodes
enc_err	Number of encryption errors
reinit	Number of node reinitialization
twr_ok	Number of succeeded TWR cycle
twr_err	Number of failed TWR cycle
res[0] x00000000	Reserved
res[1] x00000000	Reserved
res[2] x00000000	Reserved

res[3] x00000000	Reserved
res[4] x00000000	Reserved
res[5] x00000000	Reserved

Example:

```
dwm> stg
uptime: 6146
rtc_drift: 0.000000
ble_con_ok: 0
ble_dis_ok: 0
ble_err: 0
api_err: 0
api_err_cnt: 0
api_dl_cnt: 0
uwb0_intr: 3927517
uwb0_rst: 0
uwb0_bpc: 0
rx_ok: 3863996
rx_err: 4
tx_err: 0
tx_errx: 0
bcn_tx_ok: 61332
bcn_tx_err: 0
bcn_rx_ok: 61320
alma_tx_ok: 1095
alma_tx_err: 0
alma_rx_ok: 0
cl_rx_ok: 0
cl_tx_ok: 1
cl_coll: 0
fwup_tx_ok: 0
fwup_tx_err: 0
fwup_rx_ok: 0
svc_tx_ok: 0
svc_tx_err: 0
svc_rx_ok: 0
clk_sync: 0
bh_rt: 0
bh_nort: 0
bh_ev: 0
bh_buf_lost[0]: 0
bh_buf_lost[1]: 0
bh_tx_err: 0
bh_dl_err: 0
bh_dl_ok: 0
```

bh_ul_err: 0
bh_ul_ok: 0
fw_dl_tx_err: 0
fw_dl_iot_ok: 0
fw_ul_loc_ok: 0
fw_ul_iot_ok: 0
ul_tx_err: 0
dl_iot_ok: 0
ul_loc_ok: 0
ul_iot_ok: 1096
enc_err: 0
reinit: 1
twr_ok: 0
twr_err: 0
res[0]: 0 x00000000
res[1]: 0 x00000000
res[2]: 0 x00000000
res[3]: 0 x00000000
res[4]: 0 x00000000
res[5]: 0 x00000000
tot_err: 4

6.36 stc

Clears statistics.

Example:

dwm> stc

6.37 udi

Toggle displaying of IoT data received via backhaul.

Example:

dwm> udi
dl: show on
dl: len=8: 01 23 45 67 89 AB CD EF

6.38 uui

Send uplink IoT data via backhaul. Optional parameter: [cnt]. When [cnt] > 0, the node will repeat the same uplink message [cnt] times.

Example:

dwm> uui
usage: uui <hex_string> [cnt]
dwm> uui 11223344 100
ul: len=4 cnt=100 rv=4: 11 22 33 44

6.39 tlv

Parses given TLV frame. See Section 5 for valid TLV commands.

Example: read node configuration

```
dwm> tlv 8 0
OUTPUT FRAME:
40 01 00 46 02 b0 00
```

Example: toggles GPIO pin 13

```
dwm> tlv 44 1 13
OUTPUT FRAME:
40 01 00
```

6.40 aurs

Set position update rate. See section 5.3.3 for more detail.

Example:

```
dwm> aurs
Usage aurs <ur> <urs>
dwm> aurs 10 20
err code: 0
```

6.41 aurg

Get position update rate. See section 5.3.4 for more detail.

Example:

```
dwm> aurg
err code: 0, upd rate: 10, 20(stat)
```

6.42 apg

Get position of the node. See section 4.4.2 for more detail.

Example:

```
dwm> apg
x:100 y:120 z:2500 qf:100
```

6.43 aps

Set position of the node. See section 4.4.2 for more detail.

Example:

```
dwm> aps
Usage aps <x> <y> <z>
dwm> aps 100 120 2500
err code: 0
```

6.44 acas

Configures node as anchor with given options. See section 4.4.8 for more detail. Note that this command only sets the configuration parameters. To make effect of the settings, users should issue s reset command, see section 6.10 for more detail.

Example:

dwm> acas
Usage acas <inr> <bridge> <leds> <ble> <uwb> <fw_upd>
dwm> acas 0 0 1 1 2 0
err code: 0

6.45 acts

Configures node as tag with given options. See section 4.4.7 for more detail. Note that this command only sets the configuration parameters. To make effect of the settings, users should issue s reset command, see section 6.10 for more detail.

Example:

dwm> acts
Usage acts <meas_mode> <accel_en> <low_pwr> <loc_en> <leds> <ble> <uwb> <fw_upd>
dwm> acts 0 1 0 1 1 0 0 0
err code: 0

6.46 aks

Sets encryption key, takes encryption key as argument.

Example:

dwm> aks 00112233445566778899aabbccddeeff
key_set: 00112233445566778899AABBCCDDEEFF
dwm>

6.47 akc

Clears encryption key and disables encryption.

Example:

dwm> akc
ok

6.48 ans

Writes user data to non-volatile memory.

Example:

dwm> ans 00112233445566778899aabbccddeeff00112233

data: 00112233445566778899AABBCCDDEEFF00112233
len=20

6.49 ang

Reads data from non-volatile memory.

Example:

```
dwm> ang
data: 00112233445566778899AABBCCDDEEFF00112233
len=20
dwm>
```

7 BLE API

The BLE central device connects directly with the network nodes to set up and retrieve parameters. It needs to connect to each device individually to configure/control it.

7.1 BLE GATT Model

The ***network node service*** UUID is **680c21d9-c946-4c1f-9c11-baa1c21329e7**. All characteristic values are encoded as little endian as BLE spec suggests. RW – read/write; RO – read only; WO – write only.

7.1.1 Network Node Characteristics

uuid	name	length	value	flags
Std. GAP service, label **0x2A00**	Label	Var	UTF-8 encoded string	RW
3f0afd88-7770-46b0-b5e7-9fc099598964	Operation mode	2 bytes	See Section 7.1.2 for details on data encoding	RW
80f9d8bc-3bff-45bb-a181-2d6a37991208	Network ID	2 bytes	Unique identification of the network (PAN ID)	RW
a02b947e-df97-4516-996a-1882521e0ead	Location data mode	1 byte	0 - Position 1 - Distances 2 - Position + distances	RW
003bbdf2-c634-4b3d-ab56-7ec889b89a37	Location data	106 bytes max	See Section 7.1.3 for details on data encoding	RO
f4a67d7d-379d-4183-9c03-4b6ea5103291	Proxy positions	76 bytes max	Used by the module as a notification about new tag positions for the BLE central	RO
1e63b1eb-d4ed-444e-af54-c1e965192501	Device info	29bytes	Node ID (8 bytes), HW version (4 bytes), FW1 version (4 bytes), FW2 version (4 bytes), FW1 checksum (4 bytes), FW2 checksum (4 bytes), Operation flags (1 byte)	RO
0eb2bc59-baf1-4c1c-8535-8a0204c69de5	Statistics	120 bytes	Node statistics	

5955aa10-e085-4030-8aa6-bdfac89ac32b	FW update push	Max 37 bytes	Used to send structured data (FW update packets) to the module (BLE peripheral), the size is set according to max transmission unit (MTU). See Section 7.4.4 for details on data encoding.	WO
9eed0e27-09c0-4d1c-bd92-7c441daba850	FW update poll	9 bytes	Used by the module as a response/notification for the BLE central, See Section 7.4.4 for details on data encoding	RO
ed83b848-da03-4a0a-a2dc-8b401080e473	Disconnect	1 byte	Used to explicitly disconnect from BLE peripheral by writing value=1 (workaround due to android behavior)	WO

Note: The label characteristic is a special one. It is part of the standard mandatory GAP service (0x1800) under the standard name characteristic (0x2A00).

7.1.2 Operation mode characteristic

Operation mode characteristic is of 2 bytes and contains the configuration information of the nodes. The format is defined as follows:

1st byte (bit 7 down to 0)	
Bit	**value**
7	tag (0), anchor (1)
6 - 5	UWB - off (0), passive (1), active (2)
4	firmware 1 (0), firmware 2 (1)
3	accelerometer enable (0, 1)
2	LED indication enabled (0, 1)
1	firmware update enable (0, 1)
0	reserved
2nd byte (bit 7 down to 0)	
Bit	**value**
7	initiator enable, anchor specific (0, 1)
6	low power mode enable, tag specific (0, 1)
5	location engine enable, tag specific (0, 1)
4 - 0	reserved

7.1.3 Location data characteristic

Location data characteristic can contain position, distances or both. The format of the position and distances are defined as follows:

type (1 byte)	value
0 - Position only	X, Y, Z coordinates (each 4 bytes) and quality factor (1 byte), total size: 13 bytes
1 - Distances	First byte is distance count (1 byte) Sequence of node ID (2 bytes), distance (4 bytes) and quality factor (1 byte) Max value contains 15 elements, size: 14 - 106
2 - Position and Distances	Encoded Position (as above, 13 bytes) Encoded Distances (as above, 8 - 29 bytes) - position and distances are sent by tag, number of ranging anchors is max 4

Note 1: the characteristic value might be completely empty (zero length) meaning that there are neither known positions nor known distances.
*Note 2: although **location data mode** includes position and distances, it is still possible to receive distances only in the characteristic in case when position is not known.*

7.1.4 Proxy Positions Characteristic

This characteristic is provided to overcome limitation of concurrently connected nodes to the BLE central (mobile device). A passive node uses this characteristic to stream/notify about tag position updates.
Data are encoded in this characteristic as follows:

- 1 byte: number of elements (max 5)
- [sequence] tag position: 2-byte node id, 13-byte position

Thus, the maximum size of 5 tag positions is 76 bytes long.

7.1.5 Anchor-specific Characteristics

An anchor may operate as either an anchor or anchor initiator.

uuid	name	length	value	flags
3f0afd88-7770-46b0-b5e7-9fc099598964	Operation Mode	2 bytes	Bit 7 in 2nd byte: initiator enable (0, 1) (see Section 7.1.3 for detail)	RW
1e63b1eb-d4ed-444e-af54-c1e965192501	Device info		RD only operation flags: 0XXXXXXX	RO
f0f26c9b-2c8c-49ac-ab60-fe03def1b40c	Persisted position	13 bytes	X,Y,Z coordinates each 4-byte precision + quality factor (1 byte, value 1 - 100)	WO
28d01d60-89de-4bfa-b6e9-651ba596232c	MAC stats	4 bytes	Reserved for internal debug MAC statistics	RO
17b1613e-98f2-	Cluster info	5 bytes	Seat number (1 byte)/Cluster map (2	RO

4436-bcde-23af17a10c72			bytes)/Cluster neighbor map (2 bytes)	
5b10c428-af2f-486f-aee1-9dbd79b6bccb	Anchor list	129 bytes	list of node IDs, (1 byte) each ID is 64 bits long, max 16 anchors in list, first element contains array element count	RO

7.1.6 Tag-specific Characteristics

Each tag determines its own position based on the information sent by 4 surrounding anchors. The tag provides complete information of how its position is computed

uuid	name	length	value	flags
3f0afd88-7770-46b0-b5e7-9fc099598964	Operation Mode	2 bytes	Bit 6 in 2nd byte: low power mode enable (0, 1) Bit 5 in 2nd byte: location engine enable (0, 1) (see Section 7.1.3 for detail)	RO
7bd47f30-5602-4389-b069-8305731308b6	Update rate	8 bytes	Broadcast new position each *U1* ms when moving, broadcast new position each *U2* ms when stationary. U1 (4 bytes), U2 (4 bytes)	RO

7.2 Auto-positioning

It is possible to initiate an auto-positioning process through the BLE API. The auto-positioning process is finished (positions are computed) on the mobile device. The BLE API just makes it possible to initiate distance measurement and retrieval. The workflow is as follows:

1. Measure:
    a. Initiator is found and verified (the node must be a *real initiator*, not just *configured as initiator*)
    b. Initiator/network is put to measure distances mode:
        i. Make sure that **location data mode** is configured for distances or position_and_distances
        ii. Start observing on **location data** characteristic (setup cccd notification)
        iii. Received all measured distances from initiator, save the measured distances to matrix
        iv. Stop observation
    c. Distances from all other (non-initiator) nodes are retrieved:
        i. Connect
        ii. Make sure that **location data mode** is configured for distances or position_and_distances
        iii. Retrieve value stored in **location data** characteristic (directly), save the measured distances to matrix
        iv. Disconnect
2. Evaluate: evaluate the measure distances, check orthogonality, compute positions
3. Save the computed positions to nodes (on user request)

7.3 *BLE Advertisements*

BLE advertisements are a common way for a peripheral device to let others know its presence. The broadcast payload is made of triplets according to BLE spec, i.e. [length, type, <data>]. Both anchors and tags will broadcast basic information about their **presence and operation mode**. The BLE advertisement is not long enough to also include the position info.
In BLE advertisement a maximum payload of 31 bytes can be used:

- First 3 bytes are mandatory flags (one AD triplet).
- The rest 28 bytes can be used by the app to fill in AD records (each record has 2 bytes overhead of length and type)

7.3.1 Presence Broadcast

The BLE on the DWM1001 module works in connectable undirected mode. It advertises the presence with a presence broadcast which contains the availability of service and some service data. The presence broadcast follows the BLE advertisement frame structure and makes use of the 28 bytes to present information.

Because the presence broadcast has connectable flag set to true, a shortened local name AD record of 8 bytes must be included to overcome potential Android BLE stack bug. (As described in https://devzone.nordicsemi.com/question/55309/connection-issues-with-android-60-marshmallow-and-nexus-6/). The remaining bytes are filled with service data: 2 bytes for the AD record header, 16 bytes UUID, 1 byte shortened operation mode and 1-byte change counter.

The presence broadcast frame has 3 + 20 + 8 bytes in total, i.e. 31 bytes. The frame structure is shown in the table below.

AD triplet - part identification	value
LEN	0x02
TYPE	0x01 (Flags)
DATA	Device/Advertisement flags - connectable
LEN	0x13 (19 in decimal)
TYPE	0x21 (SERVICE_DATA)
DATA	**680c21d9-c946-4c1f-9c11-baa1c21329e7** (16 bytes)
	Bit layout: OXXEFFUU (1 byte) O - operation mode (tag 0, anchor 1) XX - reserved E - error indication FF - flags: initiator, bridge UU - UWB: off (0), passive (1), active (2)
	Change counter (1 byte) - change counter changes each time a characteristic gets changed (except for node statistics and specifically for Tag: position and ranging anchor)
LEN	0x07 (max)
TYPE	0x08 (Shortened local name)
DATA	First 6 letters (or less) of device local name as defined by GATT spec.

7.4 Firmware Update

The firmware update functionality is used to update the module's firmware. It can be performed either over UWB or over BLE. This section describes the control and data flow over BLE.
During FW update two characteristics, **FW update push** and **FW update poll**, are used to implement the request/response protocol.

7.4.1 Initiating FW Update

7.4.1.1 Steps:

- The *Android device* (BLE central) sets up an indication on **FW update poll** Client Characteristic Configuration Descriptor (CCCD).
- The *Android device* asks the network node if it is willing to perform the update by sending the update request/offer packet to **FW update push** characteristic. This initialization packet contains: firmware version, firmware checksum, overall firmware binary size (in bytes). This is reliable write, aka. write with response.
- The *Network node* responds with indication on *FW update poll* in two cases:
  Case 1: YES, "send me the first data buffer", see *Transmitting the FW binary* section for more information;
  Case 2: NO, and *error code* provides refuse reason.

7.4.1.2 Error states:

- **Android device**: received explicit NO indication along with error code/reason
  Resolution: the *Android device* disables CCCD indication on *FW update poll* and notifies the upper layer about the refuse reason.
- **Network node**: sudden disconnect
  Resolution: leave the FW update mode, reset current state as if the FW update did not happen.
- **Mobile device**: detects that connection has been closed.
  Resolution: Retry. If still unsuccessful after 30 seconds from FW update initialization, report to upper layer. Let the user re-initiate the firmware update on request.

7.4.2 Transmitting the FW binary

This section is inspired by http://stackoverflow.com/questions/37151579/schemes-for-streaming-data-with-ble-gatt-characteristics.

7.4.2.1 Steps

The data transmission is initiated by a network node. The network node tells the mobile device precisely which data buffer it wants using so called *FW buffer request*: size and offset. The mobile device starts sending the requested buffer in small chunks using write without response so there is no full round trip involved. The elementary chunk size is equal to MTU so that it fits into a single transmitted packet. The chunk consists of:

- Data: size should be rounded to power of 2. The current chunk size is set to 32 bytes.
- Relative offset (from the very beginning): 4 bytes.
- Identification of the message type: FIRMWARE_DATA_CHUNK (= 0x1): 1 byte

The data transmission is completely driven by the network node. After the data buffer is sent,

mobile device waits for further instructions. During the transmission, the network node normally asks for data buffer sequentially one by one to get continuous byte sequence of firmware. The node might ask for an unexpected buffer if exceptions happen, for example the current buffer transmission fails.

7.4.2.2 Error states:

- **Network node**: data chunk is missing upon receiving (non-continuous sequence), or out-of-order chunks.
  Resolution: send *FW buffer request* specifying the missing chunk and the rest of the buffer.
- **Mobile device**: receives *FW buffer request* during ongoing data transmission.
  Resolution: stop sending data, set current offset to the one in the *FW buffer request* and restart data transmission.

7.4.3 Finishing the transmission

Once the last data buffer has been successfully received, the network node will let the mobile device know through indication on *FW update poll* that it has received the full firmware binary. Upon its reception, the mobile device:

1. disconnects from the network node;
2. waits 500 ms;
3. tries to connect to network node again and check its firmware status.

7.4.4 FW update push/poll format

FW update push					
Update offer/request-Firmware data	type == 0 (1 byte)	HW version (4 bytes)	FW version (4 bytes)	FW checksum (4 bytes)	size (4 bytes)
Firmware data chunk	type == 1 (1 byte)	offset (4 bytes)	data (max 32 bytes)		

FW update poll			
Firmware buffer request	type == 1 (1 byte)	offset (4 bytes)	size (4 bytes)
Signals	type == 0 (upload refused), 2 (upload complete), 3 (save failed) 14 (save failed, invalid checksum) (1 byte)	0 bytes	

8 APPENDIX A – TLV TYPE LIST

Table 5 DWM1001 TLV type list

Type		Comment
Name	Value	
TLV_TYPE_DUMMY	0	Reserved for SPI dummy byte
TLV_TYPE_CMD_POS_SET	1	command set position coordinates XYZ
TLV_TYPE_CMD_POS_GET	2	command get position coordinates XYZ
TLV_TYPE_CMD_UR_SET	3	command set position update rate
TLV_TYPE_CMD_UR_GET	4	command get position update rate
TLV_TYPE_CMD_CFG_TN_SET	5	command set configuration for the tag
TLV_TYPE_CMD_CFG_AN_SET	7	command set configuration for the anchor
TLV_TYPE_CMD_CFG_GET	8	command get configuration data
TLV_TYPE_CMD_SLEEP	10	command sleep
TLV_TYPE_CMD_AN_LIST_GET	11	command anchor list get
TLV_TYPE_CMD_LOC_GET	12	command location get
TLV_TYPE_CMD_BLE_ADDR_SET	15	command BLE address set
TLV_TYPE_CMD_BLE_ADDR_GET	16	command BLE address get
TLV_TYPE_CMD_STNRY_CFG_SET	17	command stationary configuration set
TLV_TYPE_CMD_STNRY_CFG_GET	18	command stationary configuration get
TLV_TYPE_CMD_FAC_RESET	19	command factory reset
TLV_TYPE_CMD_RESET	20	command reset
TLV_TYPE_CMD_VER_GET	21	command FW version get
TLV_TYPE_CMD_UWB_CFG_SET	23	command UWB configuration set
TLV_TYPE_CMD_UWB_CFG_GET	24	command UWB configuration get
TLV_TYPE_CMD_USR_DATA_READ	25	command user data read
TLV_TYPE_CMD_USR_DATA_WRITE	26	command user data write
TLV_TYPE_CMD_LABEL_READ	28	command label read
TLV_TYPE_CMD_LABEL_WRITE	29	command label write
TLV_TYPE_CMD_GPIO_CFG_OUTPUT	40	command configure output pin and set
TLV_TYPE_CMD_GPIO_CFG_INPUT	41	command configure input pin
TLV_TYPE_CMD_GPIO_VAL_SET	42	command set pin value
TLV_TYPE_CMD_GPIO_VAL_GET	43	command get pin value
TLV_TYPE_CMD_GPIO_VAL_TOGGLE	44	command toggle pin value
TLV_TYPE_CMD_PANID_SET	46	command pan id set
TLV_TYPE_CMD_PANID_GET	47	command pan id get
TLV_TYPE_CMD_NODE_ID_GET	48	command node id get
TLV_TYPE_CMD_STATUS_GET	50	command status get
TLV_TYPE_CMD_INT_CFG_SET	52	command configure interrupts
TLV_TYPE_CMD_INT_CFG_GET	53	command get interrupt configuration
TLV_TYPE_CMD_BACKHAUL_XFER	55	command BACKHAUL data transfer
TLV_TYPE_CMD_BH_STATUS_GET	58	command to get UWBMAC status
TLV_TYPE_CMD_ENC_KEY_SET	60	command to set encryption key
TLV_TYPE_CMD_ENC_KEY_CLEAR	61	command to clear encryption key

TLV_TYPE_CMD_VA_ARG_POS_SET	128	VA ARG command set position
TLV_TYPE_RET_VAL	64	response return value of a command
TLV_TYPE_POS_XYZ	65	response position coordinates x,y,z
TLV_TYPE_POS_X	66	response position coordinate x
TLV_TYPE_POS_Y	67	response position coordinate y
TLV_TYPE_POS_Z	68	response position coordinate z
TLV_TYPE_UR	69	response update rate
TLV_TYPE_CFG	70	response general configuration
TLV_TYPE_INT_CFG	71	response interrupt configuration
TLV_TYPE_RNG_AN_DIST	72	response ranging anchor distances
TLV_TYPE_RNG_AN_POS_DIST	73	response ranging anchor distances and positions
TLV_TYPE_STNRY_SENSITIVITY	74	response stationary sensitivity
TLV_TYPE_USR_DATA	75	response user data
TLV_TYPE_LABEL	76	response label
TLV_TYPE_PAN_ID	77	response PAN ID
TLV_TYPE_NODE_ID	78	response node ID
TLV_TYPE_UWB_CFG	79	response UWB configuration
TLV_TYPE_FW_VER	80	response FW version
TLV_TYPE_CFG_VER	81	response CFG version
TLV_TYPE_HW_VER	82	response HW version
TLV_TYPE_PIN_VAL	85	response pin value
TLV_TYPE_AN_LIST	86	response anchor list
TLV_TYPE_STATUS	90	response status
TLV_TYPE_UWB_PREAMBLE_CODE	91	response UWB preamble code
TLV_TYPE_UWB_SCAN_RESULT	92	response UWB scan result
TLV_TYPE_BH_STATUS	93	response UWBMAC status
TLV_TYPE_BLE_ADDR	95	response BLE address
TLV_TYPE_DOWNLINK_CHUNK_0	100	response downlink data chunk nr.1
TLV_TYPE_BUF_IDX	105	response index of the buffer in the container
TLV_TYPE_BUF_SIZE	106	response data size in the buffer of the container
TLV_TYPE_UPLINK_CHUNK_0	110	response uplink data chunk nr.1
TLV_TYPE_UPLINK_LOC_DATA	120	response uplink data type
TLV_TYPE_UPLINK_IOT_DATA	121	response uplink IOT data type
TLV_TYPE_VAR_LEN_PARAM	127	response parameter with variable length

9 APPENDIX B – BIBLIOGRAPHY

1. nRF5 Getting Started, available from www.nordicsemi.com
2. IEEE 802.15.4-2011 or "IEEE Std 802.15.4™-2011" (Revision of IEEE Std 802.15.4-2006). IEEE Standard for Local and metropolitan area networks— Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs). IEEE Computer Society Sponsored by the LAN/MAN Standards Committee. Available from http://standards.ieee.org/
3. DWM1001 Firmware User Guide, available from www.decawave.com
4. DWM1001 System Overview, available from www.decawave.com
5. DWM1001 on-board package, available from www.decawave.com
6. DWM1001 Host API package, available from www.decawave.com

10 DOCUMENT HISTORY

Table 6: Document History

Revision	Date	Description
2.2	29-Mar-2019	Updated for PANS R2
2.1	07-Aug-2018	Logo Change
2.0	28-Mar-2018	Update for planned release
1.0	18-Dec-2017	Initial release

11 FURTHER INFORMATION

Decawave develops semiconductors solutions, software, modules, reference designs - that enable real-time, ultra-accurate, ultra-reliable local area micro-location services. Decawave's technology enables an entirely new class of easy to implement, highly secure, intelligent location functionality and services for IoT and smart consumer products and applications.

For further information on this or any other Decawave product, please refer to our website www.decawave.com.