Asset Tracker 3
BLE Handbook

11/04/2025

Revision History

Revision	Date	Author	Description of Change
Draft	20th June 2024	SAADI Hamza	• Draft release
Version 1	24th January 2025	SAADI Hamza	• Update section 6
V2.0	11th April 2025	SAADI Hamza	• Update section 11.2/11.3 Tracker configuration payload • Update section 13: Add binary type to the begin firmware update command • Update section 12: Add new command save config

Table of Contents

Definition / Acronym	**Description**
TBD	To be defined
FW	Firmware
SW	Software
API	Application Program Interface
HW	Hardware
BLE	Bluetooth Low Energy
WRITE_REQ	ATT write request
WRITE_CMD	ATT write command
READ	ATT read request
NOTIFY	ATT notification

1 General

1.1 Purpose

This document describe the BLE firmware and the messages exchanged between the tracker device and a mobile application through Bluetooth Low Energy (BLE).
The mobile application is the Client application and are called *Central devices* in the rest of the document.

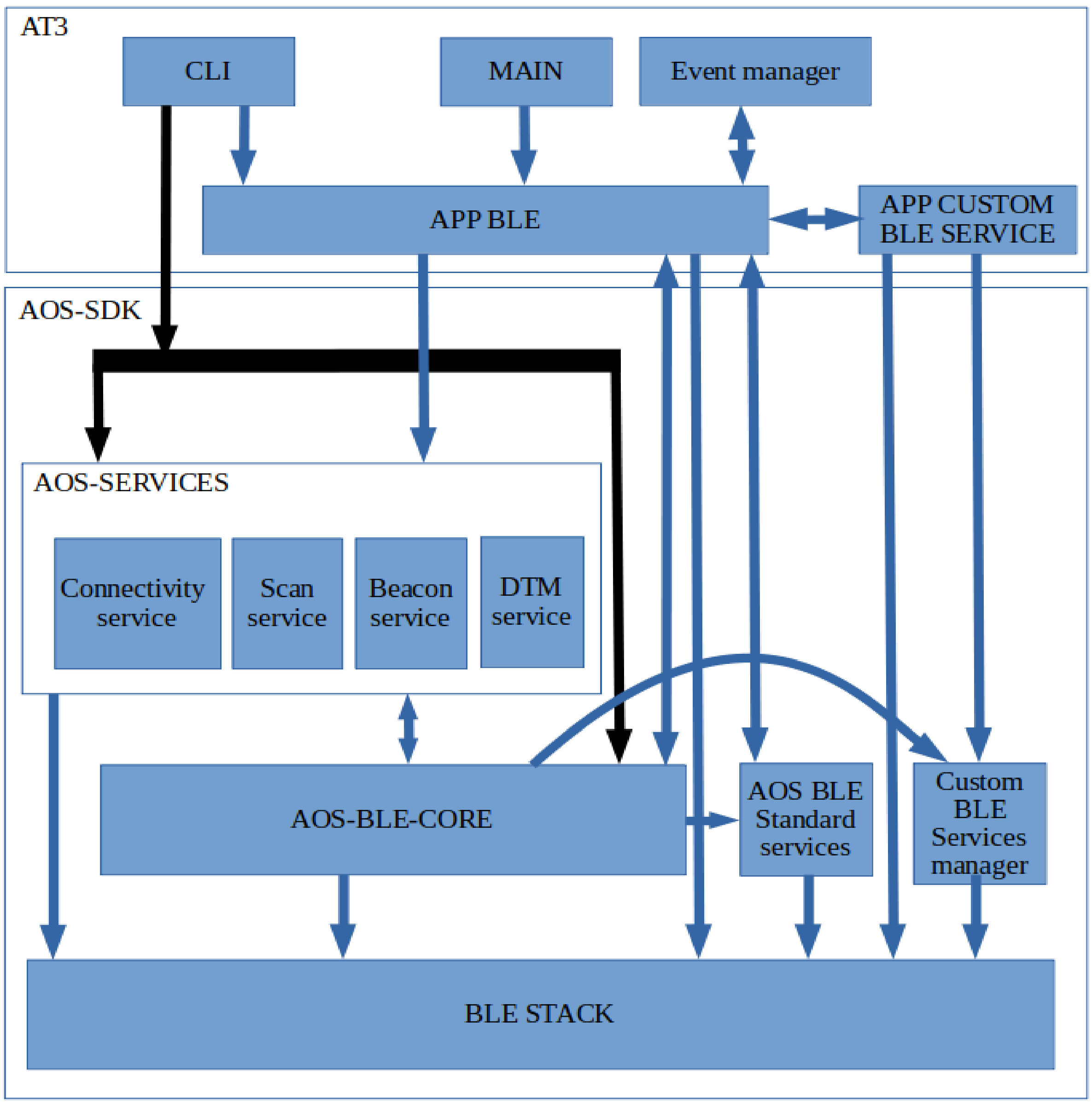
1.2 Intended Audience

The intended audience is third parties software developers implementing client applications and wishing to interface with the Abeeway trackers.

1.3 Endian and String Encoding

Unless otherwise specified, all numerical values transmitted through BLE interface shall be in **big endian**.
Unless otherwise specified, all string values transmitted through BLE interface shall be composed of unicode characters encoded with UTF-8.

2 Architecture design



2.1 BLE in AOS-SDK

The BLE in AOS-SDK have 5 major component:

- **BLE HCI/ACI**: The BLE stack.
- Custom BLE services manager: Offer APIs to create new services and characteristics, and initialize characteristics data. The user can create max 10 custom BLE services and max of 10 characteristics for each service.
- **AOS BLE standard services**: Offer the initialization and characteristic update of different BLE standard service (TX POWER SERVICE , DEVICE INFORMATION SERVICE, IMMEDIATE ALERT SERVICE, LINK LOSS SERVICE, BATTERY SERVICE , ENVIRONMENTAL SENSING SERVICE), for some characteristics when a read operation is done this component will use the app callback already saved at initialization to request data from the APP_BLE layer.
- **AOS-BLE-CORE**: Is responsible of the management of the BLE core initialization and dispatching BLE notifications to connectivity service, scan service and the main application.

- **AOS-SERVICES**: This component offer some BLE services:
  - **Connectivity service**: This service manage the BLE connectivity, it expose API related to BLE connectivity.
  - **Scan service**: This service manage beacon scanning.
  - **Beacon service**: This service manage beaconing advertising (different types of beacons are managed (eddystone UUID, altbeacon, Ibeacon, quuppa, exposure).
  - **DTM service**: Manage Direct Test Mode service (test modes are tone, BLE TX and BLE RX).

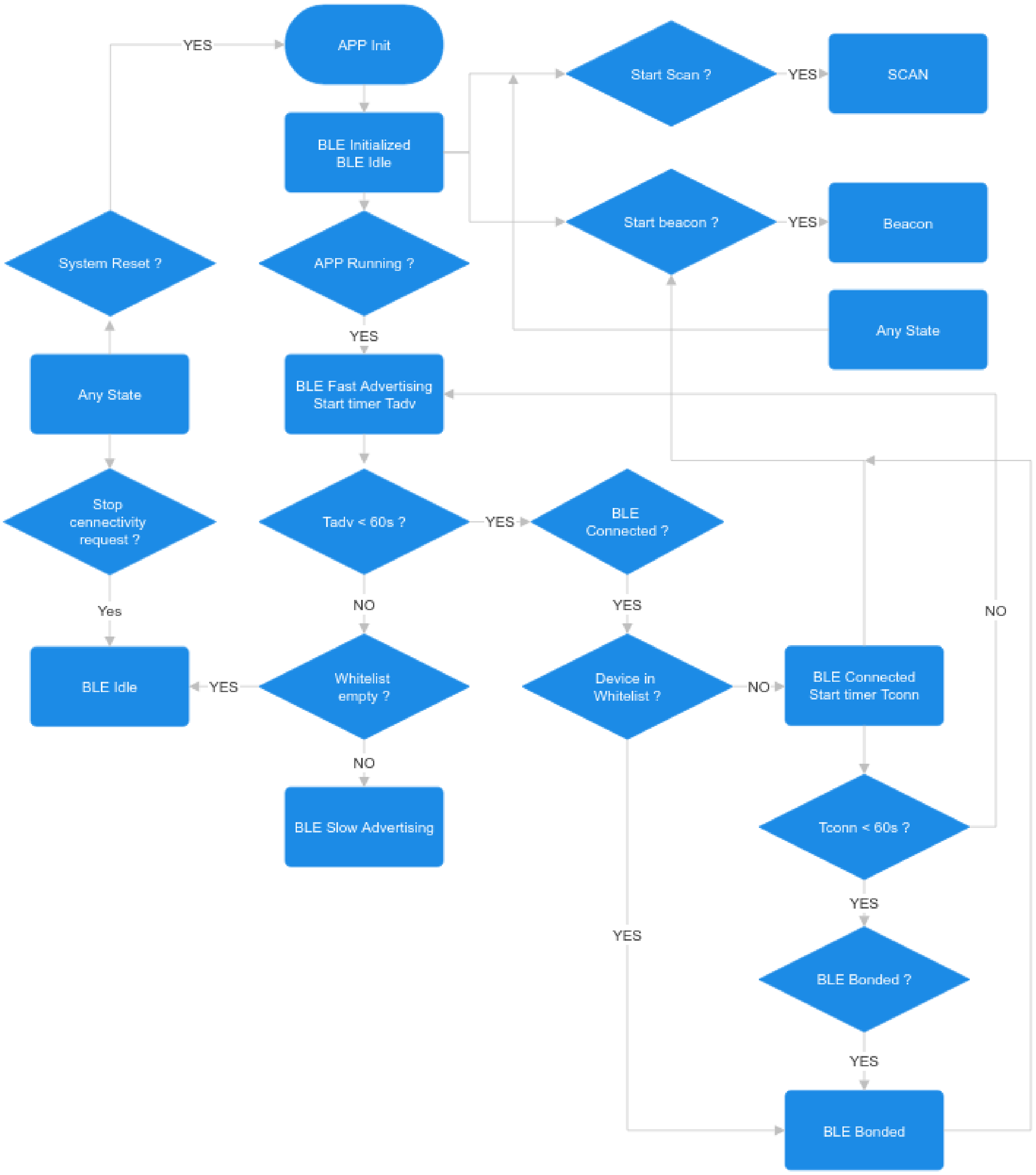
2.2 BLE in AT3

The BLE processing is present in 4 major parts:

- **MAIN**: In the main application the BLE is initialized by calling the API from APP_BLE.
- **CLI**: From the BLE command line interface the firmware call APIs from APP_BLE and AOS services to make BLE initialization or running connectivity/scan/beacon/DTM. The CLI can call AOS-BLE-CORE to get some information like BLE address or to set BLE TX power for example.
- **APP CUSTOM BLE SERVICE**: This component manage the abeeway BLE custom service and its characteristics. Initialization is done by calling APIs from Custom BLE services manager.
- **APP BLE**: This component manage AT3 BLE specifications. BLE initialization data are set here, some of these initialization are:
  - Initialize BLE core data and stack.
  - Specify BLE services to enable.
  - Initialize abeeway custom service.
  - Set callback for BLE standard services.
  - Register to core event manager.

3 General Description – State Machine

At startup the BLE is initialized and kept idle, as soon as the main application switch to running state the BLE is switched to advertising state.
The BLE state machine is described in the figure below:


APP Init
YES
BLE Initialized
BLE Idle
Start Scan ?
YES
SCAN
Start beacon ?
YES
Beacon
Any State
System Reset ?
APP Running ?
YES
Any State
BLE Fast Advertising
Start timer Tadv
Stop
cennectivity
request ?
Tadv < 60s ?
YES
BLE
Connected ?
Yes
NO
YES
NO
BLE Idle
YES
Whitelist
empty ?
Device in
Whitelist ?
NO
BLE Connected
Start timer Tconn
NO
BLE Slow Advertising
Tconn < 60s ?
YES
YES
BLE Bonded ?
YES
BLE Bonded

The BLE can run simultaneously:

- Beacon + Scan: While beaconing the device can run scan procedure, while scanning the beaconing is stopped.
- Connectivity + Scan: While advertising for connectivity or already connected to a mobile app, the device can run scan procedure.
- Connected bonded + Beacon: When the device is already connected and bonded, it can run beaconing, in this case if the device is disconnected to the central device, it will continue to beacon and the connectivity advertising will fail. Note to have these two features simultaneously the device should be connected first then beaconing start.

The BLE cannot run simultaneously:

- Connectivity advertising + Beacon: We can't run two types of advertising simultaneously.

3.1 Connectivity

The tracker device communicates with the Client application using BLE. It will act as a Bluetooth peripheral and uses GAP and GATT profiles. The client application should configure its BLE stack as central.
The BLE is always ready to start connectivity beside the case where beaconing is already running.
The main connectivity states of the BLE are:

1. Idle
2. Fast advertising
3. Slow advertising
4. Connected
5. Bonded

Idle state

In Idle state the connectivity is stopped.
This state is entered if the advertising timeout and no bonded device saved or if the user hit the command to stop connectivity.

Advertising state

In this state, the BLE advertising name is in the form of: **ABW** + 9 last DEVEUI digits (e.g. ABW10200XXXX).
Fast advertising :

- The device switch to fast advertise after device start or if the user hit the command to start advertising.
- The device advertise every 500ms on the three main advertising channels.
- If the device have a bonded central device saved, the device will switch to slow advertising after 60s, if no central device saved the device will stop connectivity after the 60s.

In slow advertising the device advertise every 2s on the three main advertising channels without any timeout. This lets the client to use the auto-reconnection.

Connected state

This state is entered from the *Advertising* state if an unknown central device initiated a connection request and bond data are not present.
Once the connection is established, the device acts as a BLE Peripheral.
In connected state:

- The tracker does not accept any other connections.
- If the bonded central device lose it's security keys, so the repairing could be done.
- If the device don't have a bond saved the app will wait 60s for a bond request, if the bond request is not received the device will disconnect and restart fast advertising.

- If the connected central device and the bonded central device saved are not the same, the device will disconnect and restart fast advertising.
- If the connected central device and the bonded central device saved are the same, the device switch to bonded state.

Bonded state

This state is entered from the *Connected* state when a security request is received. In this state, security keys are stored inThe tracker device communicates with the Client application using BLE. It will act as a Bluetooth peripheral and uses GAP and GATT profiles. The client application should configure its BLE stack as central. both BLE module and Central device. The connection is encrypted.
In bonded state the central device can access to all BLE services and characteristics (BLE CLI, Configuration, Simple commands ….).

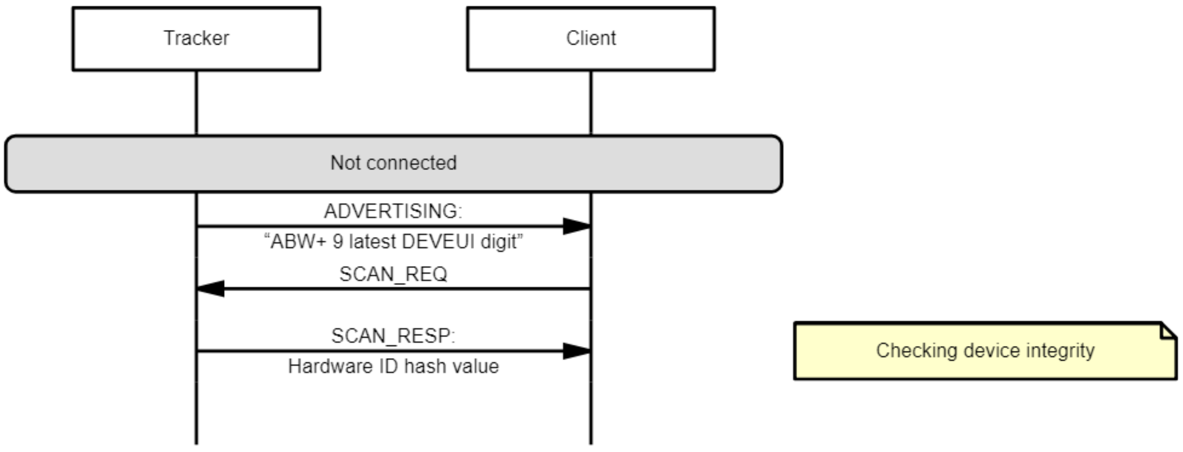
3.2 Scanning Tracker device

Advertising interval:
When the tracker device is in advertising mode. The configured advertisement interval depends on its bonding state.

- In absence of bond, the tracker advertise each 500ms during the fast advertisement duration.
- If the device is bonded but not connected, it sends advertisements each 2 seconds without any timeout. This lets the client to use the auto-reconnection.

The fast advertisement duration is set to 1 minutes by default but it could be configured when the main mcu send start advertisement command.

Advertisement packets are sent on each of the 3 advertising channels.



Advertising data:
The Client application should scan for all BLE devices containing advertising data with Complete Local Name: "**ABW** + 9 latest DEVEUI digit" (e.g. ABW012345678).

Scan response:
The tracker device is publishing an encrypted value of its unique Hardware ID within advertising SCAN_RSP. Client application has to proceed to an ACTIVE SCAN in order to request this remaining advertising data.

3.3 Connection

When a Client application detects a tracker advertisement packet, it should reply with a connection request.

Connection interval:

There is no limitation regarding connection interval within the connection update request. However, we advise to use a fast connection interval during service discovery:

- Connection interval: 7.5ms
- Connection latency: 0ms
- Supervision timeout: 2000ms

Connection update request:

The tracker will send a connection update request 60s seconds after bond establishment using the following parameters:

- Minimum connection interval: 980ms
- Maximum connection interval: 1000ms
- Slave latency: 0
- Supervision timeout: 6000ms

The Client application has to accept the above parameters for energy saving purpose.
This request is canceled if the central device send a command to set the connection parameters using the characteristic simple command of the abeeway custom service (UUID 0x273D).

3.4 Services

The access to most services requires a bonded connection (encrypted). The services that does not require a secure connection are the simple command service and firmware update service.

4 Retrieving Device Information

The tracker device exposes the *Device Information Service* allowing a bonded *Client application* to read information summarized in the table below:

Device Information Service (UUID 0x180A)			
Characteristics			
UUID	Value	Perm	Default content
0x2A24	Model Number String	READ	Module
0x2A25	Serial Number String	READ	Device EUI-64 (DevEUI) (ex: 20635f01020015f2)
0x2A26	Firmware Revision String	READ	AOS SDK version (ex: 1.0.0)
0x2A28	Software Revision String	READ	Main Application Firmware version (ex: 1.0.0)
0x2A29	Manufacturer Name String	READ	ABEEWAY

5 Retrieving Tx Power Level

The tracker device exposes the *Tx Power Service* allowing a bonded *Client application* to retrieve the Tx power level.

Tx Power Service (UUID 0x1804)			
Characteristic			
UUID	Value	Perm	Content
0x2a07	Tx Power Level	Read	Description

6 Retrieving Battery Information

Tracker device exposes Battery Service allowing bonded *Client application* to retrieve percentage of battery level and battery charging state.

Battery Service (UUID 0x180F)			
Characteristic			
UUID	Value	Perm	Content
0x2A19	Battery Level	READ NOTIFY	Value: • 1% represents a battery that is fully discharged. • 100% represents a battery that is fully charged. • Notifications are sent for every evolution of the battery level.
0x2A1A	Battery Power State	READ NOTIFY	Value: • 0x77: Charger present and charging (Re-chargeable Battery). • 0x67: Charger present but not charging (Re-chargeable Battery). • 0x66: Charger not present and discharging (Re-chargeable Battery). • 0x56: Charger not present charging not supported (primary battery) • 0x57: Charger present charging not supported (primary battery) • 0x55: Charger not supported, charging not supported (VBUS detection disabled) • Notifications are sent for every state changed.

Standard Battery Power State data field:

Battery Power State data field		
Bits	**Value**	**Comment**
Power Information		
[0,1]	0b00 0b01 0b10 0b11	• PWR_INFO_UNKNOWN • PWR_INFO_NOT_SUPPORTED • PWR_INFO_NOT_PRESENT • PWR_INFO_PRESENT
Discharging State		
[2,3]	0b00 0b01 0b10 0b11	• DISC_STATE_UNKNOWN • DISC_STATE_NOT_SUPPORTED • DISC_STATE_DISCHARGING • DISC_STATE_CHARGING
Charging State		
[4,5]	0b00 0b01 0b10 0b11	• CHAR_STATE_UNKNOWN • CHAR_STATE_NOT_SUPPORTED • CHAR_STATE_DISCHARGING • CHAR_STATE_CHARGING
Level		
[6,7]	0b00 0b01 0b10 0b11	• LEVEL_UNKNOWN • LEVEL_NOT_SUPPORTED • LEVEL_GOOD • LEVEL_CRITICAL

7 Retrieving Temperature

Tracker device exposes Environmental Sensing Service allowing bonded *Client application* to retrieve current temperature in Celsius.

Environmental Sensing (UUID 0x181A)			
Characteristic			
UUID	Value	Perm	Content
0x2A1F	Temperature Celsius	READ NOTIFY	Current temperature, unit is [0.1 °C]

8 Sending Immediate Alerts

The *Client application* can send *immediate alerts* to the bonded tracker device.

Immediate Alert Service (UUID 0x1802)			
Characteristic			
UUID	Value	Perm	Content
0x2a06	Alert Level	WRITE_CMD	0x00: No Alert 0x01: Mild Alert 0x02: High Alert

9 Receiving Immediate Alerts

The *Client application* can receive *alerts* from the bonded tracker device.

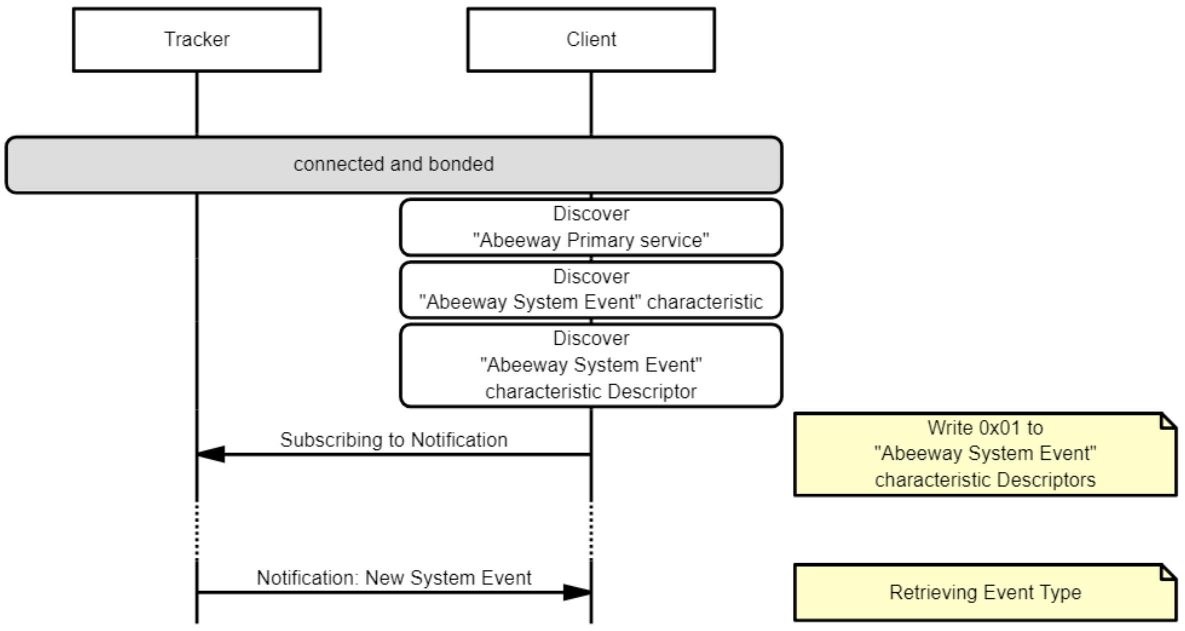
Link Loss Service (UUID 0x1803)			
Characteristic			
UUID	Value	Perm	Content
0x2a06	Alert Level	WRITE_CMD	0x00: No Alert 0x01: Mild Alert 0x02: High Alert

10 Receiving System Event

At any time, client application must be ready to handle Notification on Characteristic System Event (UUID 0000**2742**-1212-efde-1523-785feabcd123). This characteristic is embedded within Abeeway Service:

Abeeway Primary Service (UUID 00008A45-1212-efde-1523-785feabcd123)	
System Event Characteristic	
UUID	Permission
0000**2742**-1212-efde-1523-785feabcd123	NOTIFY

Following diagram explains procedure to respect:


Tracker
Client
connected and bonded
Discover "Abeeway Primary service"
Discover "Abeeway System Event" characteristic
Discover "Abeeway System Event" characteristic Descriptor
Subscribing to Notification
Write 0x01 to "Abeeway System Event" characteristic Descriptors
Notification: New System Event
Retrieving Event Type

Once a System Event Notification has been received, client application has to retrieve event type by reading Notification content and look at table below for corresponding meaning:

System Event Notification content	
Value	Description
0x00	SOS start. Enter the SOS mode.
0x01	Device start moving
0x02	Motion end
0x03	SOS stop. Leave the SOS mode

11 Tracker Configuration

11.1 Introduction

The *Client application* can update and retrieve Tracker device parameters by sending the corresponding ATT Write command.

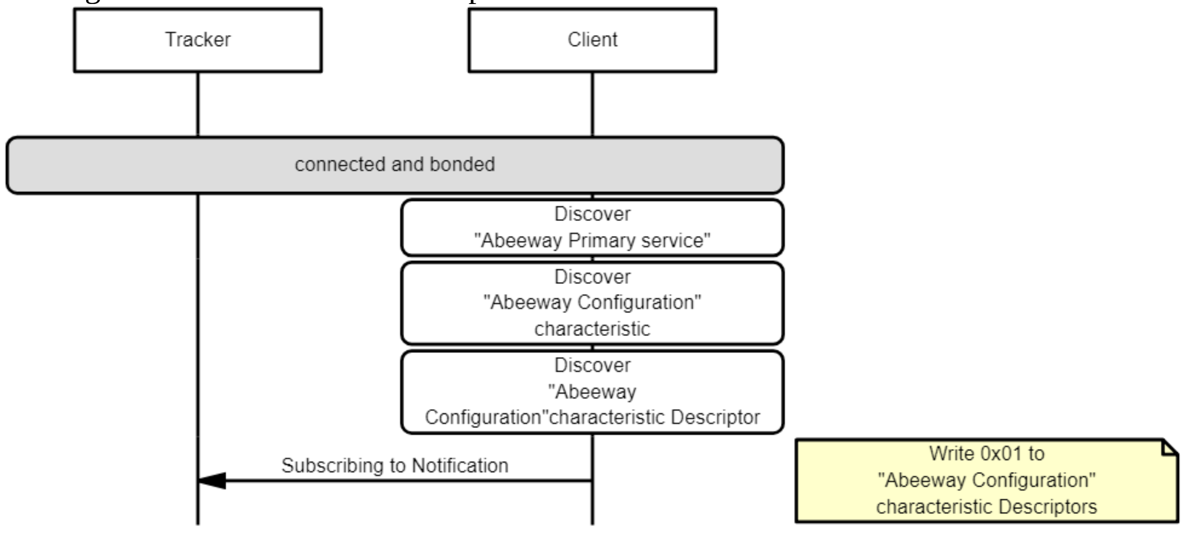
Abeeway Primary Service (UUID 00008A45-1212-efde-1523-785feabcd123)	
Configuration Characteristic	
UUID	Permission
0000**2740**-1212-efde-1523-785feabcd123	WRITE_CMD - NOTIFY

The Client application must discover the Abeeway Primary service (UUID: 0000**8A45**-1212-efde-1523-785feabcd123). This service supports a writable and notifiable characteristic called "Configuration" with UUID: 0000**2740**-1212-efde-1523-785feabcd123.

On every write command on its characteristic value, Tracker device will send back the operation response through a Notification.

Before receiving a notification, the *Client application* must subscribe to the notification handler. This is done by writing 0x01 to its related Client Characteristic Configuration Descriptor (cccd). Once done, the *Client application* can receive such notifications.

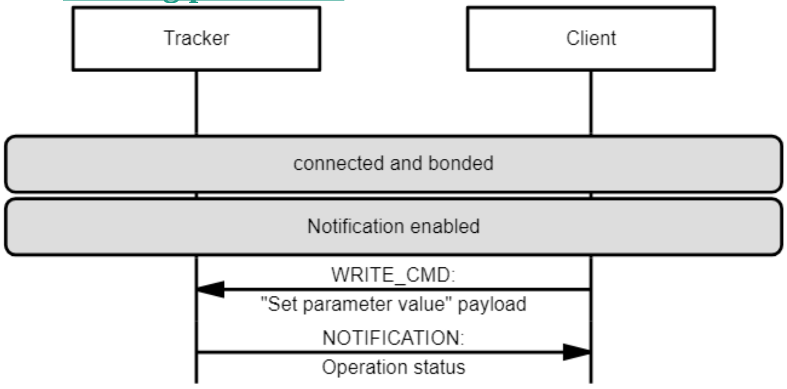
The diagram below summarizes the operation described above.


Tracker
Client
connected and bonded
Discover "Abeeway Primary service"
Discover "Abeeway Configuration" characteristic
Discover "Abeeway Configuration"characteristic Descriptor
Subscribing to Notification
Write 0x01 to "Abeeway Configuration" characteristic Descriptors

Note:

1- To optimize the throughput during the configuration read or write commands, it is highly recommended to **use ATT write commands** instead of write request.

11.2 Writing parameter



<01><Param ID 2Bytes><Type 1Byte><Value N bytes> 2 Bytes
<Status 1 byte>

Once the notification has been enabled, the *Client application* should update **MTU size to at least 43 bytes**, then it can update a parameter by sending a write command containing the parameter identifier to be modified and its new value.
The complete request content shown below.

Writing parameter value with Write Command				
Bytes	0	1:2	3	4:X
Data	Write Command (**always 0x01**)	Parameter ID	Parameter Type	Parameter Value

The parameter value length is variable, it depend on the parameter type:

- Integers/float: 4 bytes
- String and array: length depend on the parameter, check AT3 reference guide.

The parameter types are:

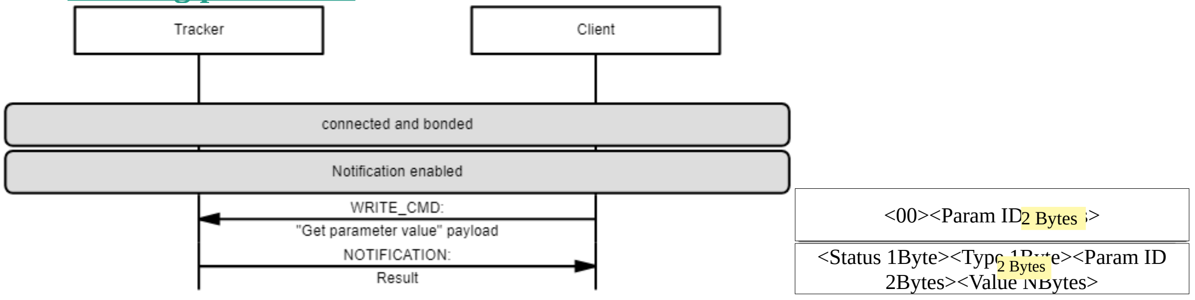
Deprecated	= 0: This parameter has been deprecated
Integer	= 1: Signed integer on 32 bits
Float	= 2: Single precision float number (size 4 bytes)
String	= 3: ASCII string null terminated (max 32 bytes including the NULL)
Byte array	= 4: Stream of bytes (max 32 bytes)

The *Client application* must wait for the notification reception (containing write status), before sending another command. Such message is displayed below.

Writing parameter response from Notification	
Bytes	0
	Status : • 0x00: Success • 0x01: Fail • 0x02: Value not supported • 0x03: Data length error

Example:
To set the parameter ID 0x0100 (0x01 is the group id and 0x00 is the local id) to value 0x15E, knowing that this is an integer, the write command **0x010100010000015E** should be sent.
A Notification will be received in response like 0x00 if write operation succeed.

11.3 Reading parameter



Once the notification has been enabled, the *Client application* can read a parameter value by sending a write request/command containing the parameter identifier to read.

Reading parameter value with Write Command		
Bytes	0	1:2
Data	Read Command (**always 0x00**)	Parameter ID

The response is contained within the following Notification.
Remember that the *Client application* must wait for this Notification before sending another command.

The notification associated to this command is shown below.

			Reading parameter value response	
Bytes	0	1:2	3	4:X
Data	Status • 0x00: Success • 0x01: Fail • 0x02: Value not supported	Parameter ID	Parameter Type	Parameter Value. *Please check the status before taking this value in consideration*

The parameter value length is variable, it depend on the parameter type:

- Integers/Float: 4 bytes

String and array: length depend on the parameter, check AT3 reference guide.

The parameter types are:

Deprecated	= 0: This parameter has been deprecated
Integer	= 1: Signed integer on 32 bits
Float	= 2: Single precision float number (size 4 bytes)
String	= 3: ASCII string null terminated (max 32 bytes including the NULL)
Byte array	= 4: Stream of bytes (max 32 bytes)

Example: To read the value of parameter ID 0x0100 (Integer). The write command 0x000100 should be sent.
A notification will be received in response, containing the parameter value: 0x000100010000015E.

12 Sending command

The *Client application* can send specific commands to the Tracker device, any command sent to this characteristic is forwarded to the main application firmware.
Before accessing this service, the *Client application* has to discover the Abeeway Primary service (UUID: UUID 0000**8A45**-1212-efde-1523-785feabcd123).

Abeeway Primary Service (UUID 00008A45-1212-efde-1523-785feabcd123)	
Simple command Characteristic	
UUID	Permission
0000**273D**-1212-efde-1523-785feabcd123	READ, WRITE_REQ, WRITE_CMD

Possible simple commands	
Command	Action
0x00	Set fast connection parameters
0x01	Set slow connection parameters
0x02	Set very fast connection parameters
0x03	Enable BLE CLI feature
0x04	Disable BLE CLI feature
0x05	Save the AT3 configuration
0x99	Clear bond

12.1 Clearing bond

When bonded, if a *Client application* wants to delete bonding information saved on its own persistent storage, it must request first this operation on the tracker device.
This operation is achieved by writing **0x99** to the custom command characteristic value.

12.2 Requesting fast connection parameter

The *Client application* can request the tracker device to send a Fast Connection Parameter Update Request by writing **0x00** to the simple command characteristic value.
Fast connection parameters are:

- Minimum connection interval: **40**ms
- Maximum connection interval: **40**ms
- Slave latency: 0
- Supervision timeout: 6000ms

12.3 Requesting slow connection parameter

The *Client application* can request the tracker device to send a Slow Connection Parameter Update Request by writing **0x01** to the custom command characteristic value.
Slow connection parameters are:

- Minimum connection interval: **980**ms
- Maximum connection interval: **1000**ms
- Slave latency: 0
- Supervision timeout: 6000ms

12.4 Requesting very fast connection parameter

The *Client application* can request the tracker device to send a Very Fast Connection Parameter Update Request by writing **0x02** to the custom command characteristic value.
Fast connection parameters are:

- Minimum connection interval: **8**ms
- Maximum connection interval: **8**ms
- Slave latency: 0
- Supervision timeout: 6000ms

12.5 Enabling BLE CLI

The Client application could enable BLE CLI feature by sending the simple command 0x03.
The tracker traces will be sent to the Client application and the commands could be received.
When the BLE CLI feature is enabled the customer have to hit the user password to be able to send commands.

12.6 Disabling BLE CLI

The Client application could disable BLE CLI feature by sending the simple command 0x04.
The tracker traces will be then sent over LPUART or USB depending on the hardware used.
When the BLE CLI feature is disabled the customer have to hit the user password to be able to send commands.

12.7 Save the configuration

The client application could save the AT3 configuration by sending 0x05 to the characteristic 0x273D.

12.8 Read CMD

When the central device send a read command to this characteristic it will receive a read response with value set to the last command value sent, for example if the last command sent is a clear bond, the value to read is 0x99.

In the case of setting connection parameters the value read will be in case of:

- Success: the same command value sent (0x00 or 0x01, or 0x02)
- Fail: 0xAA (BLE stack busy, reschedule request after 500ms)

13 Updating the firmware

The device can update the main firmware and the BLE stack via BLE interface. To do this update the tracker must be connected and bonded.
The update process is done in 4 different steps:

- Step 1: Discover necessary services and characteristics.
- Step 2: Perform the initialization of the firmware update.
- Step 3: Send the firmware binary data.
- Step 4: Send the firmware binary CRC.

Note that only binary files (*.bin) can be transferred.

13.1 Service discovery

The application should discover the Abeeway Primary services and should request the two custom characteristics as shown in the table below.

Abeeway Primary Service (UUID 00008A45-1212-efde-1523-785feabcd123)		
Custom service Characteristics		
UUID	Permission	Usage
0000**273d-**1212-efde-1523-785feabcd123	WRITE_CMD / WRITE_REQ READ_REQ	Writing commands
0000**273e-**1212-efde-1523-785feabcd123	WRITE_CMD / WRITE_REQ READ_REQ / NOTIFY	MCU firmware update

The rest of the sections uses the following terms:

- **273d** to refer to UUID 0000**273d-**1212-efde-1523-785feabcd123
- **273e** to refer to the UUID 0000**273e-**1212-efde-1523-785feabcd123

13.2 Application protocol

This section describes the different protocol data unit exchanged during the initialization procedure and the binary data transfer.

Service characteristics 273d

On this characteristic, there is no notification. The protocol used is shown in the table below:

Direction	Type	Payload	Comment
Client → Tracker	WRITE_REQ	Cmd: 0x02	Enable very fast connection parameters
Client → Tracker	READ_REQ	Cmd: 0x02	Read answer
Tracker → Client	READ_RSP	Status (1byte)	Status of the command: 0x00: Success Other: error

Service characteristics 273e

On this service, there are notifications. The protocol used is shown in the table below:

Dir.	Type	Payload				Comment
		len	B0			
C → T	WRITE_REQ	9	0x00	DEVEUI: 8 byte		Enable DFU
C → T	READ_REQ	1	0x00			Read answer
T → C	READ_RSP	2	0x00	S: 1 byte		S: Status 0x00: Success
C → T	WRITE_REQ	6	0x01	Code len 4 bytes	FW type 1 byte	Starting DFU
C → T	READ_REQ	1	0x00			Read answer
T → C	READ_RSP	2	0x01	S: 1 byte		S: Status
C → T	WRITE_CMD	n	0x02	Address 3 bytes A2A1A0	Data n-4 bytes	Write binary data A2A1A0: Address data: n – 4 binary data.
T → C	NOTIF	5	0x02	S: 1 byte	Address 3 bytes A2A1A0	Write acknowledge. S: Status A2..A0: Next expected address.
C → T	WRITE_REQ	3	0x03	CRC 2 bytes C1C0		Send binary data CRC C1.. C0: CRC16

Notes

- The payload length is not carried in the payload.
- The Address is coded on 3 bytes and sent MSB first (A2=MSB and A0=LSB) => sent in big endian.
- The CRC is coded on 2 bytes and sent MSB first => sent in big endian.

The following table provides the status values and meanings.

Value	**Recoverable**	**Meaning**
0x00	-	The operation completed successfully
0x01	N	Service not initialized
0x02	Y	The parameter contains invalid data or data length error
0x03	N	Operation timed-out
0x04	N	Internal error
0x05	N	Operation aborted by user
0x06	Y	CRC mismatch
0x07	Y	Offset mismatch
0x08	N	Binary length error
0x09	Y	Device mismatch
0x0A	N	Battery too low
0x0B	N	Flash storage error

Recoverable means that the full DFU process can be recovered Otherwise it's not and the tracker will leave DFU mode. In non recoverable state, the app should restart the full process.

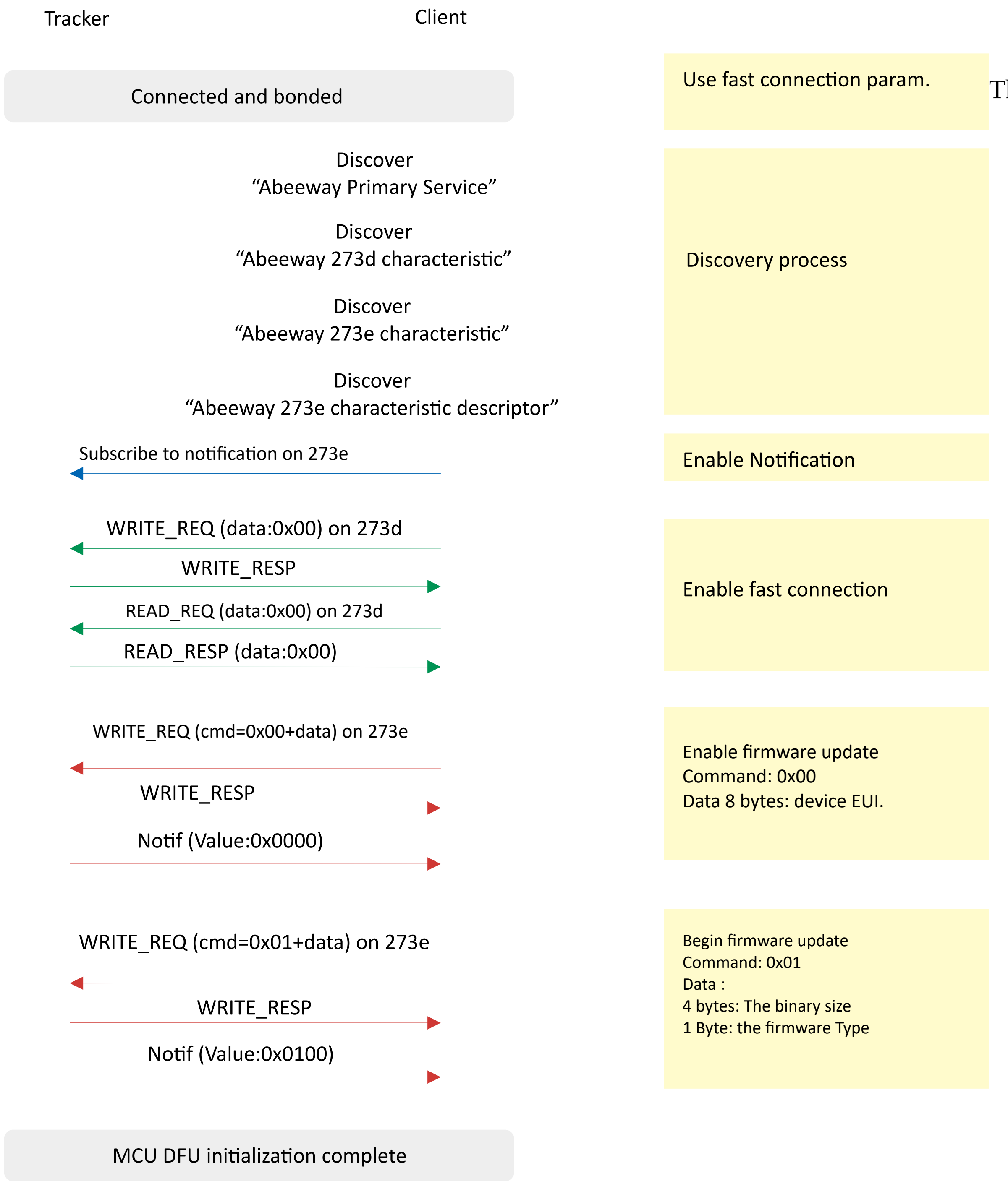
The DFU status message response is like below:

uint8_t dfu_cmd	:	Response to this DFU command
uint8_t status	:	Response Status
uint8_t offset[3]	:	Chunk offset

dfu_cmd	status	offset
1 Byte	1 Byte	3 Bytes
dfu_cmd_enable : 0x00 dfu_cmd_begin : 0x01 dfu_cmd_update : 0x02 dfu_cmd_final : 0x03 dfu_cmd_abort : 0x04	See previous table	If the dfu_cmd is dfu_cmd_update, this field is the next chunk offset expected

13.3 Initialization

The initialization consists of setting the MCU firmware in a state where it will accept only binary firmware packets.
The diagram below illustrates the procedure to initialize the MCU firmware update.


Tracker
Client
Connected and bonded
Use fast connection param.
Discover "Abeeway Primary Service"
Discover "Abeeway 273d characteristic"
Discover "Abeeway 273e characteristic"
Discover "Abeeway 273e characteristic descriptor"
Discovery process
Subscribe to notification on 273e
Enable Notification
WRITE_REQ (data:0x00) on 273d
WRITE_RESP
READ_REQ (data:0x00) on 273d
READ_RESP (data:0x00)
Enable fast connection
WRITE_REQ (cmd=0x00+data) on 273e
WRITE_RESP
Notif (Value:0x0000)
Enable firmware update
Command: 0x00
Data 8 bytes: device EUI.
WRITE_REQ (cmd=0x01+data) on 273e
WRITE_RESP
Notif (Value:0x0100)
Begin firmware update
Command: 0x01
Data :
4 bytes: The binary size
1 Byte: the firmware Type
MCU DFU initialization complete

The initialization process is summarized below:

1. Enable the Notification reception on characteristic 273e.
2. Write request with 0x02 to **273d** to enable higher speed BLE connection parameters.

3. Read request value of **273d**. The read request should be sent after receiving the write response of step 2. Check for success (0x02) or failure (0xaa)
    - In case of failure, usually due to the module being busy with an ongoing negotiation, you can wait and retry, or disconnect and reconnect to the device. The new connection parameters will be used for the new connection. It is recommended to disconnect since it is faster.
4. Write request with 0x00xxxxxxxxxxxxxxxx to **273e** to enable EFM DFU
    - Where the first byte is the command and xx..xx is the device EUI-64 (DevEUI) of the tracker, which should be known by the application.
5. Read the notification on **273e** and check for success
    - Possible error causes: incorrect message format, Incorrect Device EUI-64.
6. Write request with 0x01xxxxxxxxyy to **273e** to begin EFM DFU, where
    - xxxxxxxx is the binary size to send
    - yy is the firmware type to send (0: main MCU; 1: BLE stack)
7. Read notification **273e** and check for success
    - Possible error causes: incorrect message format, EFM DFU not previously enabled, wrong physical flash size (old boards with smaller flash)

Note:

- If any of the steps 4,5,6,7 fail, then the central device should abort EFM DFU (see section 16.7).
- If the EFM DFU is aborted in initialization procedure the central device should set connection parameters to slow to reduce current consumption (see section 13.5).
- If any notification is not received the central device can make a read request to check the status, if there is an error the central can resend the command, and if after some retries there is always an error, the central should abort MCU DFU then restart the procedure (in worst case disconnect and reconnect then restart procedure).

13.4 Writing the binary data

Once initialized, the firmware binary data can be sent to the tracker.

Due to the Maximum Transmit Unit (MTU) limitation (23 bytes), the firmware binary has to be split into multiple chunks of 16 bytes.

Each write message are transmitted through ATT write command, and acknowledges are received via notifications.

Notes 1:

- If there is a critical error or if the central device stopped sending the binary file for 12 seconds so the tracker reset.

Notes 2:

A notification will be sent for each write command sent to the tracker, this is described as below:
Central → Tracker: Write request

CMD (1 Byte)	Address (3 Bytes)	Data (16 Bytes)
0x02	0xXXXXXX	0xDD...DD

Where 0xXXXXXX is the address of the chunk

Where 0xDD...DD is the chunk binary data

Tracker → Central: Notification

CMD (1 Byte)	Status (1 Byte	Address (3 Bytes)
0x02	0xSS	0xYYYYYY

Where 0xSS is the status as described in the table status values and meanings.

Where 0xYYYYYY is the address of the next address waited.

This means that for write request of the chunk address 0x000000 the notif will be:

- 0x0200000010: if the chunk is successfully received.
- 0x020B000000: if the chunk address mismatch.

Notes 1:

Two strategies can be implemented here:

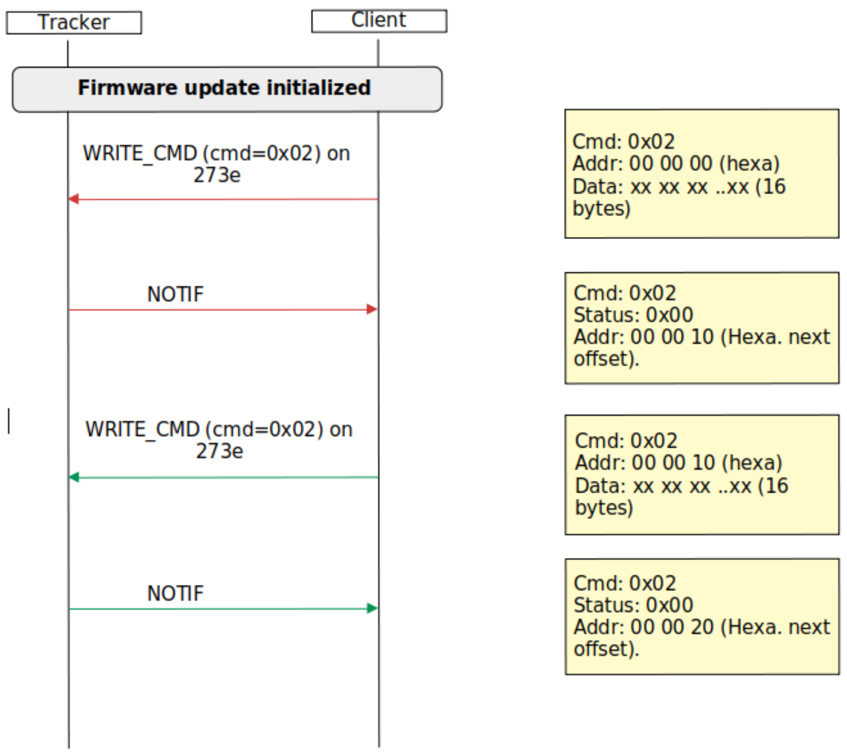
1. The basic strategy consists of sending a chunk and waiting for its acknowledge (notification). This simple method is not very efficient in term of throughput.
2. The optimized strategy consists of sending one chunk and another one in advance (before waiting for the first acknowledge). Once the first acknowledge is received, the next chunk is sent. At the end, the last acknowledge must be received before ending the process.

Notes 2:

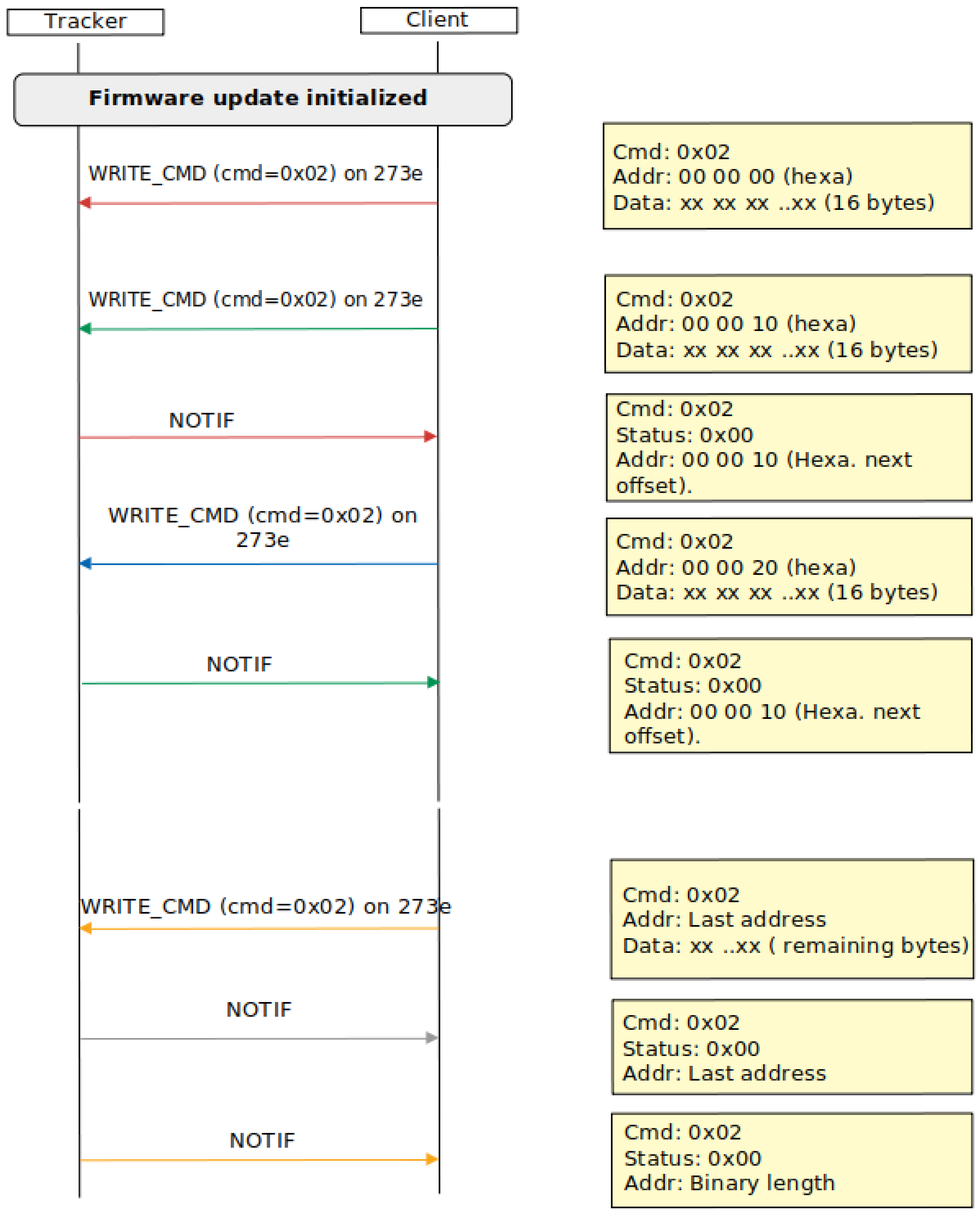
- Writing a chunk should be done using a WRITE_CMD not a WRITE_REQ

These strategies can be pictured as follow:

1- The diagram below illustrates the writing process, using the strategy 1.

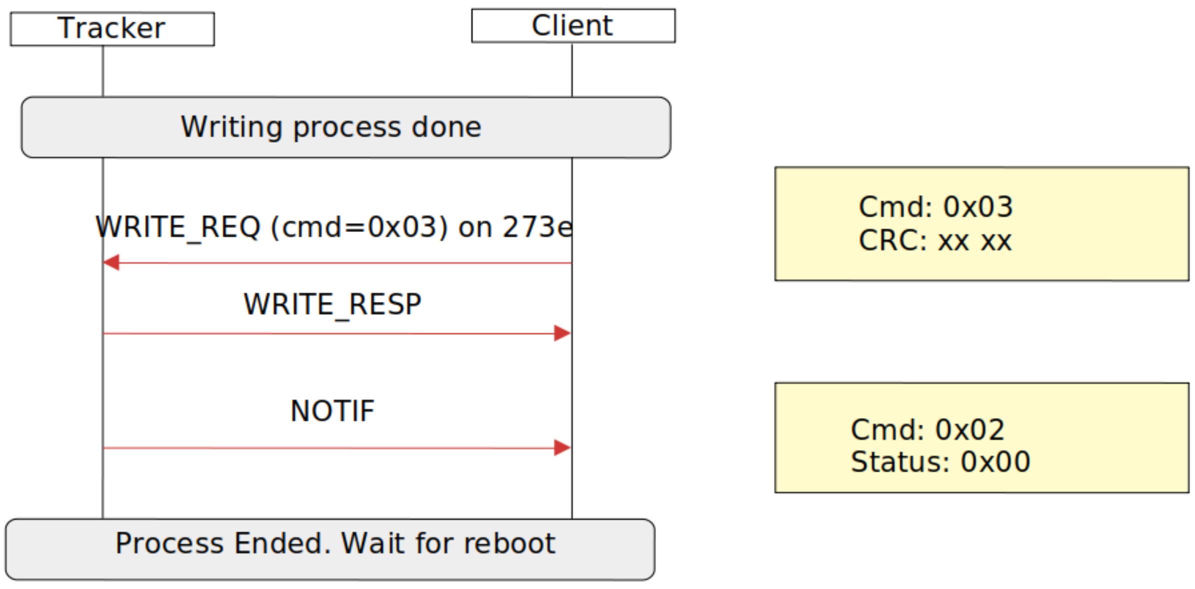

Tracker
Client
Firmware update initialized
WRITE_CMD (cmd=0x02) on 273e
NOTIF
WRITE_CMD (cmd=0x02) on 273e
NOTIF
Cmd: 0x02
Addr: 00 00 00 (hexa)
Data: xx xx xx ..xx (16 bytes)
Cmd: 0x02
Status: 0x00
Addr: 00 00 10 (Hexa. next offset).
Cmd: 0x02
Addr: 00 00 10 (hexa)
Data: xx xx xx ..xx (16 bytes)
Cmd: 0x02
Status: 0x00
Addr: 00 00 20 (Hexa. next offset).

2- The diagram below illustrates the writing process, using the strategy 2:


Tracker
Client
Firmware update initialized
WRITE_CMD (cmd=0x02) on 273e
Cmd: 0x02
Addr: 00 00 00 (hexa)
Data: xx xx xx ..xx (16 bytes)
WRITE_CMD (cmd=0x02) on 273e
Cmd: 0x02
Addr: 00 00 10 (hexa)
Data: xx xx xx ..xx (16 bytes)
NOTIF
Cmd: 0x02
Status: 0x00
Addr: 00 00 10 (Hexa. next offset).
WRITE_CMD (cmd=0x02) on 273e
Cmd: 0x02
Addr: 00 00 20 (hexa)
Data: xx xx xx ..xx (16 bytes)
NOTIF
Cmd: 0x02
Status: 0x00
Addr: 00 00 10 (Hexa. next offset).
WRITE_CMD (cmd=0x02) on 273e
Cmd: 0x02
Addr: Last address
Data: xx ..xx (remaining bytes)
NOTIF
Cmd: 0x02
Status: 0x00
Addr: Last address
NOTIF
Cmd: 0x02
Status: 0x00
Addr: Binary length

13.5 Sending the CRC

The diagram below illustrates the end of the writing process. This is done by sending the CRC of the binary data.



The CRC is calculated over the full binary size (not per chunk). This is the xmodem predefined CRC algorithm.
Example:

```
uint16_t CRC_calc(uint8_t *start, uint8_t *end)
{
  uint16_t crc = 0x0;
  uint8_t  *data;
  for (data = start; data < end; data++) {
    crc  = (crc >> 8) | (crc << 8);
    crc ^= *data;
    crc ^= (crc & 0xff) >> 4;
    crc ^= crc << 12;
    crc ^= (crc & 0xff) << 5;
  }
  return crc;


}
```

Once the process is done, the tracker reboot. This will disconnect the BLE connection.

Note

In case of process failure, the device will reboot with the previous firmware.

13.6 Abort MCU DFU

The Central device can abort MCU DFU at any step of the procedure.
In initialization step the abort will not reset the device just the mcu dfu procedure is reset.
In the other steps the abort will reset the tracker.

1. Write request with 0x04 to **273e** to abort EFM DFU

2. Read request **273e** and check for success

14 MCU Command Line Interface

14.1 Overview

Add an option to direct the MCU command line interface over a BLE link, and traces can be sent to the mobile app.
The mobile app can send commands to the tracker on the characteristic 0x2748 and receive traces as notification on the characteristic 0x2749.

CUSTOM_UUID_CLI_INPUT 0x2748

Write characters to the device CLI

CUSTOM_UUID_CLI_OUTPUT 0x2749

Notify characters to the mobile app.

14.2 Service discovery

The application should discover the Abeeway Primary services and should request the two custom characteristics as shown in the table below.

Abeeway Primary Service (UUID 00008A45-1212-efde-1523-785feabcd123)		
Custom service Characteristics		
UUID	Permission	Usage
0000**2748**-1212-efde-1523-785feabcd123	WRITE_CMD	Send CLI command to the device
0000**2749**-1212-efde-1523-785feabcd123	NOTIFY	Receive serial data from the device

14.3 BLE CLI specification

This feature is divided in 2 parts, Commands and Logs.
Logs:
Logs are sent from the tracker to the mobile app, as a result of a command or a simple information message. The logs are decomposed to multiple chunks of 20 bytes each (chunks size could be increased in future releases) and sent to the Client application using notifications on the characteristic 0x2749.
The data sent to the mobile app are in ASCII and are printable, the Client application should print on its terminal the received data as soon as received, without waiting for the end of line character. The only non-printable character sent by the tracker are CR ('\r') and LF ('\n'). They should be interpreted as such by the Client application.
Commands:
The CLI commands are sent by the mobile app to the tracker by writing on characteristic 0x2748. Any complete command should end with the characters "\r\n", and the max size of a command is **128** bytes including termination characters "\r\n".
As the BLE limits the data transferred between the mobile app and the tracker to 20 bytes, the mobile app should fragment the commands exceeding the 20 bytes to multiple chunks of 20 bytes max (chunks size could be increased in future releases).
The BLE will transfer all chunks received to the main application (AT3) where the complete command will be recomposed and executed,
Control characters are managed by the AT3 firmware so the mobile app should not send such bytes. Only CR and LF control bytes are allowed.

14.4 BLE CLI Activation/Deactivation

To be able to print correctly the logs on the mobile app, the connection parameters should be set to very fast connection params by sending 0x02 to the characteristic 0x273D.

To reduce the power consumption we propose the below procedure:

- Tracker and mobile app securely connected, the BLE CLI feature is inactive by default

1. When the user open the Client application terminal page, 2 commands should be sent to the tracker:
   1. Update connection parameters to very fast conn params
   2. Activate BLE CLI feature, by sending the simple command 0x03 using the characteristic 0x273D
2. When the user leave the Client application terminal page, 2 commands should be sent to the tracker:
   1. Deactivate BLE CLI feature, by sending the simple command 0x04 using the characteristic 0x273D
   2. Update connection parameters to slow conn params