BTXXXX EN12830 FUNCTIONALITY DESCRIPTION FOR CLIENT

Contents

[Functionality description 2](#_Toc179472528)

[General information 2](#_Toc179472529)

[Functionality 2](#_Toc179472530)

[Parameters 2](#_Toc179472531)

[Characteristic UUIDs 3](#_Toc179472532)

[Record Info 4](#_Toc179472533)

[Random Value 5](#_Toc179472534)

[Record Data 5](#_Toc179472535)

[Command 7](#_Toc179472536)

[Encryption algorithm 14](#_Toc179472537)

[Changelog 14](#_Toc179472538)

# Functionality description

Features:

* Start Recording with periods of 1-, 5- and 15-minutes and time sync.
* Stop Recording.
* Delete Records.
* Download records via characteristic with 2 methods.
* Download records from a specific timestamp.
* Time synchronization.
* Encrypted commands with Challenge-Response mechanism.
* Encrypted Recording Status Info and Records Download Mechanism.

## General information

All EN12830 related communication with the device is encrypted to prevent unauthorized modification. To prevent replay attacks, all commands require a challenge response mechanism.

The unencrypted random value should be read from corresponding characteristic. It should be appended to the beginning of the command. Then data that contains random value and command value should be encrypted. This encrypted data should be written to corresponding command characteristic. The result of this command can be read from same characteristic. If the command is about reading from device, the corresponding encrypted data can be read from Record Data characteristic.

Temperatures are stored in pages. Each page contains a header and can store up to 945 records. If recording into this page is started, it contains starting time. If recording is finished to this page, this header also contains stop time, record count of that page and CRC.

# Functionality

## Parameters

There are 2 parameters need to be given with start command. First one is timestamp, second on is period.

Timestamp is stored for user. It is not needed for temperature recording and operation of this firmware. This way, user can use any time format up to 8 bytes.

## Characteristic UUIDs

Created new service with 4 characteristics.

Table 1:Char UUID table.

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Service UUID** | **Characteristic UUID** | **Read-Write** | **Encrypted** | **Size Flexible** | **Size** |
| Record Info | e61c0000-7df8-4d4e-8e6d-c611745b92e9 | e61c0001-7df8-4d4e-8e6d-c611745b92e9 | Read Only | Yes | Fixed | 16 Bytes |
| Random Value | e61c0000-7df8-4d4e-8e6d-c611745b92e9 | e61c0002-7df8-4d4e-8e6d-c611745b92e9 | Read Only | No | Fixed | 2 Bytes |
| Record Data | e61c0000-7df8-4d4e-8e6d-c611745b92e9 | e61c0003-7df8-4d4e-8e6d-c611745b92e9 | Read Only | Yes | Fixed | 42 Bytes |
| Command | e61c0000-7df8-4d4e-8e6d-c611745b92e9 | e61c0004-7df8-4d4e-8e6d-c611745b92e9 | Write then Read | Write: Yes Read: No | Write: Variable  Read:  Fixed | Write: 4 Bytes Read:  1 Byte |
| SHT4x Serial Number | 0x180A | a610249f-913e-46bd-b14f-c6dedc432165 | Read Only | No | Yes (Max 12 bytes) | String |

### Record Info

This characteristic contains the recent information about current recording. It holds the variables in little-endian format and it is encrypted.

* ‘*is\_recording*’ can be either 0, or 1. Recording is happening, value is 1, otherwise 0. If any reset occurs while recording, current session will stop and this value will be changed to 0.
* ‘interval’ holds the period of recording in unit of seconds. If ‘*is\_recording*’ is 1, this value contains the period of ongoing recording. Otherwise, it holds stored recording.
* ‘*number\_of\_records*’ holds the count of records. If ‘*is\_recording*’ is 1, this value contains the number of records for recording in session. Records are written to flash after every 3 measurements. So, if a reset occurs, latest 1 or 2 record can be lost. If ‘*is\_recording*’ is 0, it holds count of records stored in flash.
* ‘*start\_timestamp*’ holds the timestamp. If ‘*is\_recording*’ is 1, this value contains the timestamp of ongoing recording. Otherwise, it holds timestamp of records stored in the flash.



Figure 1. Structure of “record info” characteristic.

Table 2:Example of decrypted “record info “.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **is\_recording** | **interval** | **number\_of\_records** | **start\_timestamp** |
| Received (Hex) | 01-00 | 2C-01 | 3D-02 | 01-FD-B7-62-00-00-00-00 |
| Meaning | 1 | 300 | 573 | 1656225025 |

### Random Value

This characteristic contains the little-endian uint16\_t type random value that is need to be used while sending command. This value is not encrypted. Firmware compares this value with the value from decrypted command request. If it matches, command is accepted.

### Record Data

This characteristic is for sending encrypted records and other data.



Figure 1. Structure of decrypted “record data” when index is 0 (zero).

When “START\_RECORD\_SEND” command received (if challenge-response is passed), encrypted configuration will be written to characteristic with index at the start. ‘index’ (zero) will not be encrypted but ‘start\_config’ and ‘stop\_config’ is encrypted. ‘reserved’ maybe filled with 0xFF or garbage.

|  |
| --- |
| **typedef struct \_\_attribute\_\_((packed)) {**  int16\_t records[15];  records\_crc\_t crc;  } sRECORDS\_Record\_t;  **typedef struct \_\_attribute\_\_((packed)) {**  uint16\_t chunk\_index;  sRECORDS\_Record\_t record\_structs[4];  } sRECORDS\_Record\_Chunk\_t; |

Figure 2. Structure of decrypted “record data” when index larger than 0 (zero).

Table 3. Example of encrypted and decrypted data of “record data “ (header).

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Bytes** | | | | | | | | | | | | |
|  | timestamp | | | | interval | | CRC | | Record Count | | CRC | | Reserved (24 Bytes) | |
| **Received (hex)** | D4 | 8E | 04 | 2D | 45 | D6 | 0E | AD | 11 | 82 | 29 | 9E | FFFF… | |
| **Deciphered(hex)** | 0E | B5 | 11 | 64 | 3C | 00 | 9F | 64 | 29 | 00 | 1F | 90 | FFFF… | |
| **Meaning** | 1678882062 | | | | 60 | | 0x9F64 | | 41 | | 0x1F90 | | - | |

Until “SEND\_NEXT\_CHUCK” command received and passed the challenge-response, data at the “Record Data” characteristic will be same. If secure “START\_RECORD\_SEND” received, data sending will be restart and data with zero index will be sent.

When “SEND\_NEXT\_CHUCK” command received successfully, with the start of ‘index’ 0, sending temperature records starts. Size of every chunk is 130 bytes. First 2 bytes is little-endian uint16\_t ‘chunk index’ and remaining data structure contains 60 records and 4 CRC. After index there is 15 records and CRC. After that, another 15 records and CRC. Then another 15 records and CRC. Finally last 15 records and CRC.

Except chunk index, all elements of ‘records’ array is encrypted. Also, every member of record\_t is little-endian.

Temperatures are stored in int16\_t as multiplied by 100. For example, 23.42 Celsius is stored as 2342. If recording stopped by user before third temperature measurement, “INT16\_MIN”(-32768) will be written to remaining element and CRC will be calculated with those values. Remaining bytes of characteristic will be filled with 0xFF.

The last data of any page may contain less than 60 records, in this case remaining space will be filled with 0xFF. After all records downloaded from a page, next chunk will be page header. This will contain start timestamp, number of records and CRC. If the recording to this page is finished, the stop timestamp will be valid.

If there is no more data stored in flash, “SEND\_NEXT\_CHUCK” command will return error and data at the “Record Data” characteristic will not change.

Table 4:Example of encrypted and decrypted data of “record data “.

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
|  | **Bytes** | | | | | | | | | | | | | |
|  | index | | 1th temp | | 2nd temp | | 3rd temp | | … | | 60th temp | | CRC | |
| **Received (hex)** | 03 | 00 | 7F | 8B | B8 | 3D | 9D | 3F | 70 | 3F | 7F | 8B | D2 | 68 |
| **Deciphered(hex)** | 03 | 00 | 0C | 09 | 0B | 09 | 0E | 09 | CA | 8D | 0C | 09 | 00 | 80 |
| **Meaning** | 3 | | 2316 | | 2315 | | 2318 | | 0x8DCA | | 2316 | | -32768 | |

### Command

This characteristic is for sending records related commands and 1 response for that command. After every write, 1 byte response to that command can be read from same characteristic. Commands must be encrypted but response is not encrypted.

Every command must start with little-endian uint16\_t random value that received from “Random Value” characteristic. Then, little-endian uint16\_t command value must be present. Except “START\_RECORDING”, total size of all commands are 4 Bytes.

Table 5:Command list can be sent via “Command “ characteristic.

|  |  |  |  |
| --- | --- | --- | --- |
| **Command Name** | **Value** | **Size of parameter** |  |
| START\_RECORD | 0x0001 | 6 Bytes | It starts the recording with the given interval. If recording is already stopped and send interval to start is the same measurement will be continued and stored in next page. If interval is different all records data is deleted. |
| STOP\_RECORD | 0x0002 | 0 | Stops the ongoing recording. |
| DELETE\_RECORD | 0x0003 | 0 | Deletes the stored records and recording state stays. If device was recording it continues recording but from the start. |
| START\_RECORD\_SEND | 0x0004 | 0 | Starts to send Record chunks. |
| SEND\_NEXT\_CHUNK | 0x0005 | 0 | Sends next chunk. |
| TIME\_SYNC | 0x0006 | 4 Bytes | Send UNIX time for syncing the device. With this received timestamp, the device will update its time. If necessary, it will create records or it will skip some records.  Refer: Figure 6 |
| START\_RECORD\_SEND\_TS | 0x0007 | 4 Bytes | Send from what UNIX time data must be sent. |
| SEND\_CURRENT\_TS | 0x0008 | 0 | This command will write current timestamp of the device to Record Data characteristic if the device is recording. Data will be 4 bytes little-endian, unsigned, and encrypted. |
| START\_FAST\_RECORD\_DOWNLOAD | 0x0009 | 0 | Starts to send Record chunks. After this command sent, reading “Record Data” char is enough. After each read, next chunk will be written to same char automatically. |

The received command will be decrypted by firmware and random value will be compared with value of “Random Value” characteristic.



Figure 3. Structure of parameter of “START\_RECORDING” command.

Parameter of “START\_RECORDING” should consist of little-endian interval in unit of seconds and Unix timestamp.

Table 6:An example for “START\_RECORD” command

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  | **Random Value** | **Command** | **interval** | **Unix timestamp** |
| Decrypted Command (Hex) | C4-57 | 01-00 | 2C-01 | 01-FD-B7-62 |
| Meaning | 22468 | 0x0001 | 300 | 1656225025 |

Parameter of “TIME\_SYNC” should consist of little-endian Unix timestamp.

Table 7:An example for “TIME\_SYNC” command.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Random Value** | **Command** | **Unix sync timestamp** |
| Decrypted Command (Hex) | C4-57 | 06-00 | 01-FD-B7-62 |
| Meaning | 22468 | 0x0006 | 1656225025 |

Parameter of “START\_RECORD\_SEND\_TS” should consist of little-endian Unix timestamp.

Table 8:An example for “TIME\_ START\_RECORD\_SEND\_TS” command.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Random Value** | **Command** | **Unix sync timestamp** |
| Decrypted Command (Hex) | C4-57 | 07-00 | 01-FD-B7-62 |
| Meaning | 22468 | 0x0007 | 1656225025 |

Other commands do not require any parameter.

When a command is sent, the result will be written by firmware to the same characteristic. Length of these responses is 1 byte.

Table 9:Command Responses.

|  |  |  |
| --- | --- | --- |
| **Response Name** | **Value** | **Meaning** |
| ERROR\_SUCCESS | 0x00 | There is no error. Command received successfully |
| ERROR\_GENERAL | 0x01 | Unspecified Error. |
| ERROR\_DECRYPTION | 0x02 | Received command could not successfully decrypted. |
| ERROR\_RANDOM\_VALUE | 0x03 | Received random value does not match with correct one. |
| ERROR\_UNKNOWN\_CMD | 0x04 | Received command is not known. |
| ERROR\_LENGTH | 0x05 | Length of received command is wrong. |
| ERROR\_NOT\_STARTED | 0x06 | Recording did not start because of invalid parameter or device is already a recording. |
| ERROR\_NOT\_STOPPED | 0x07 | Recording couldn’t be stopped or Command couldn’t be performed because recording still in session. |
| ERROR\_NO\_MORE\_CHUNCK | 0x08 | No more data chunk to be send. |
| \_ERROR\_NO\_DATA\_TS | 0x09 | There is no data from asked timestamp. |
| ERROR\_SENDING\_NOT\_STARTED | 0x0A | Send next chuck command occurred before send first chuck or send from timestamp. |
| ERROR\_NO\_DATA | 0x0B | There is no data to send. |

Workflow of reading:

1. Send start reading command START\_RECORD\_SEND or START\_RECORD\_SEND\_TS.
2. Read Command char if ERROR\_NO\_DATA or ERROR\_NO\_DATA\_TS was not raised because if they are data would not be present.
3. Read header. Record count value has information how many records there is in page. Header packet will always have index 0.
4. Record count should be subtracted by received record count so the user would know how many packets will be send if after subtracting there is remain 1 should be added to packet number. So calculated packet number + 1 from packet index inside data packet.
5. Sending 0x0005 command for continuing reading from Record Data char. First data packet of the page will always be 0.
6. When all the records from that page have been read;
7. User must read Command char to check if it is not no more chunks left.
8. If flag is ERROR\_NO\_MORE\_CHUNCK all data has been read.
9. If flag is not ERROR\_NO\_MORE\_CHUNCK sending 0x0005 next packet will be header of the next page. Steps from 3 can be repeated.

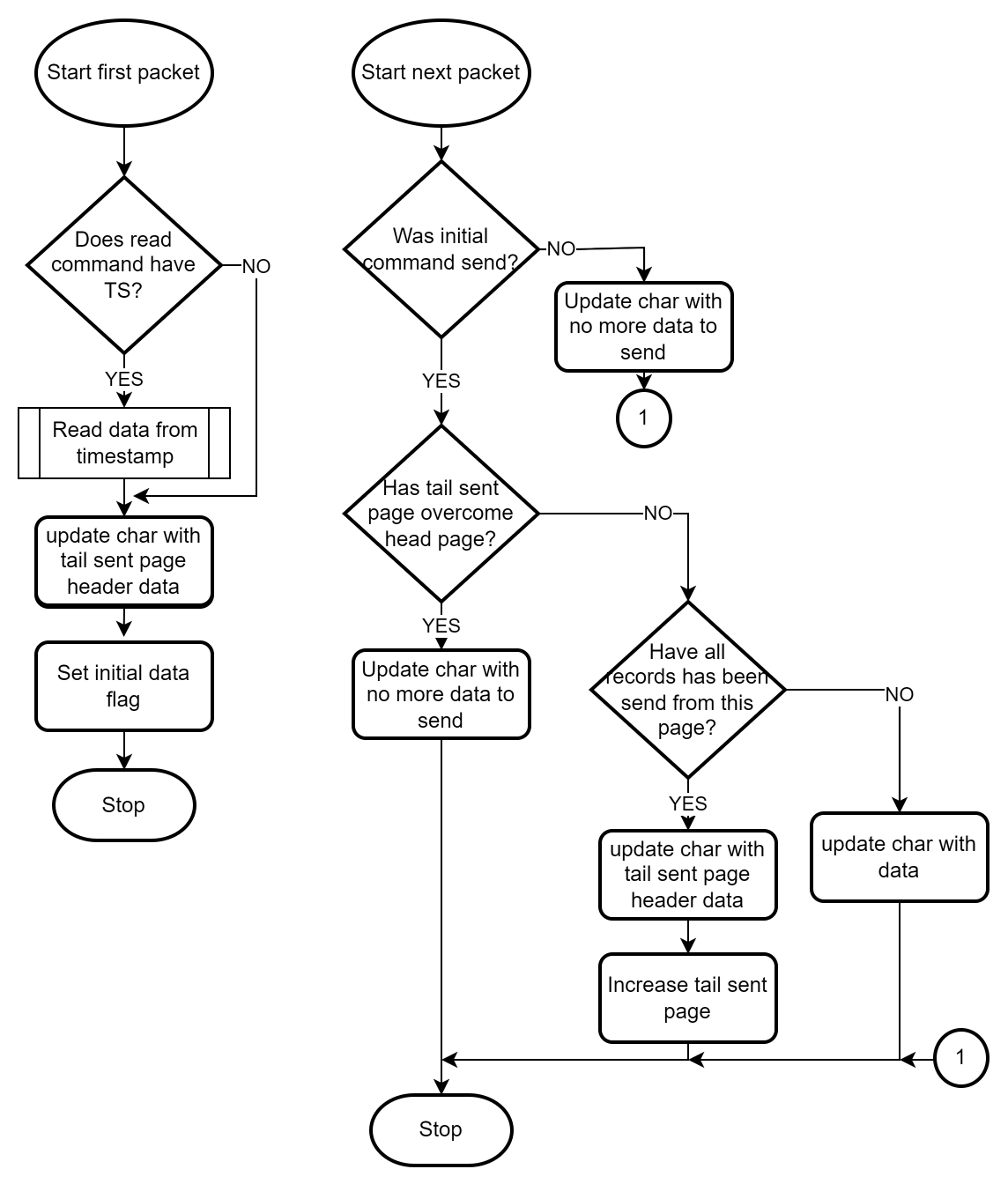


Figure 4. Reading algorithm

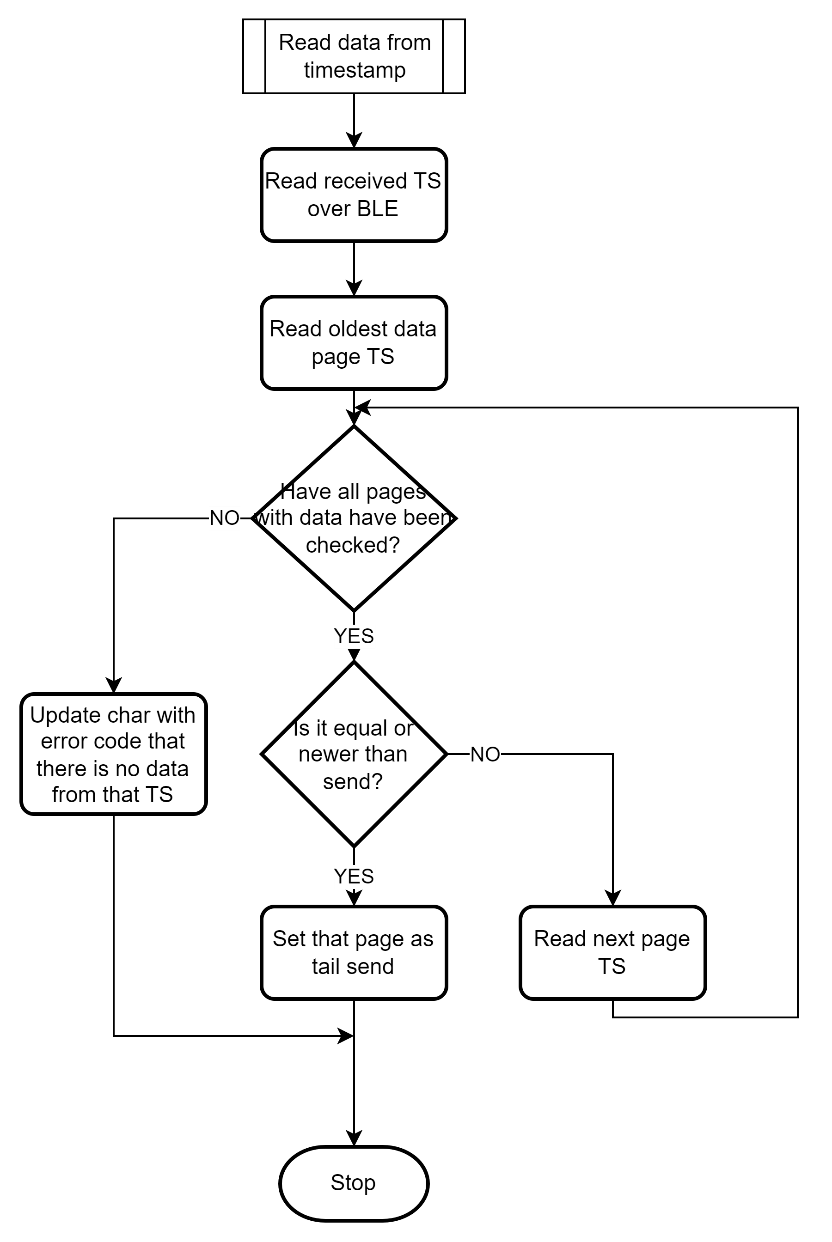


Figure 4. sent timestamp searching

A picture containing text, screenshot, font, design

Description automatically generated

Figure 5: Algorithm for time sync

## Encryption algorithm

The Tiny XTEA algorithm is used for this firmware. All commands, data received from the device uses this encryption method. For detailed information, refer to the document about xTEA algorithm. To receive the document please fill out a HelpDesk case as this document is sensitive and cannot be shared on Wiki.