|  |  |  |  |
| --- | --- | --- | --- |
| **Title** | Local Communication Protocol | | |
| **Distribution** | Software Archive | | |
| **Author** | **Paul Li** | **Date** |  |
| **Review** |  | **Date** |  |
| **Approved** |  | **Date** |  |
| **Remarks** |  | | |

Contents

[Introduction 3](#_Toc511639875)

[Serial port 3](#_Toc511639876)

[Physical layer 3](#_Toc511639877)

[Basic frames definition 3](#_Toc511639878)

[Address 4](#_Toc511639879)

[ID 4](#_Toc511639880)

[Function code 4](#_Toc511639881)

[CRC 5](#_Toc511639882)

[CAN2.0 7](#_Toc511639883)

[Physical layer 7](#_Toc511639884)

[Head 7](#_Toc511639885)

[Address: 7](#_Toc511639886)

[REV: 7](#_Toc511639887)

[Frame type 7](#_Toc511639888)

[Ex – multi-frames 8](#_Toc511639889)

[Function code 8](#_Toc511639890)

[Data 8](#_Toc511639891)

[Example about Can Realization 8](#_Toc511639892)

[Design Decisions 9](#_Toc511639893)

[Application Specific 9](#_Toc511639894)

[CMD\_POLL 9](#_Toc511639895)

[CMD\_READ\_TYPE 10](#_Toc511639896)

[CMD\_READ\_LENGTH 11](#_Toc511639897)

[CMD\_READ\_ATR\_NUM 11](#_Toc511639898)

[CMD\_READ\_ADRINFO 11](#_Toc511639899)

[Read Enumeration Process 12](#_Toc511639900)

[Put Object Process 12](#_Toc511639901)

[Bootloader Realization 14](#_Toc511639902)

[Supported Commands: 14](#_Toc511639903)

[To get device information: 14](#_Toc511639904)

[Step to Update Firmware 14](#_Toc511639905)

[Test Cases 15](#_Toc511639906)

[Revision History 18](#_Toc511639907)

Introduction

The communication is to handle the communication between UI and Measure boards. It supports the features below:

1. Master to Slaves: one to one CMD;
2. Slave burst data (the burst option and time are configurable)

Serial port

## Physical layer

* Biggest gap time between 2 bytes are 3 bytes time (base on the bard rate).
* Max response time is 250ms;
* Uart parameter fixed: 8,1,N; 115200;

## Basic frames definition

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 byte | 1 byte | 1 byte | 1 byte | 2 bytes | Multi bytes | 2 byte |
| Address DST | Address SRC | ID | Function code | length | data | CRC |

**#define** MAX\_BUFF\_DATA\_LEN 512

**#define** EX\_DATA\_LEN (6u)

**#define** CRC\_DATA\_LEN (2u)

**#define** MAX\_CMD\_HEAD\_LEN (0x6u)

**typedef** **union**

{

uint8\_t data[MAX\_BUFF\_DATA\_LEN+EX\_DATA\_LEN+MAX\_CMD\_HEAD\_LEN+CRC\_DATA\_LEN];

**struct** \_St1

{

uint8\_t dst;

uint8\_t src;

uint8\_t id;

uint8\_t function;

uint16\_t length;

uint8\_t dataBuff[MAX\_BUFF\_DATA\_LEN+MAX\_CMD\_HEAD\_LEN+CRC\_DATA\_LEN];

}St;

}Layer2Frame;

## Address

|  |  |
| --- | --- |
| Value | comments |
| FF | Reserved for broadcasts |
| 0~0xFE | Address for all devices include master and slave |

## ID

The ID is normally increase by itself, the ID is unique for one request and response;

## Function code

|  |  |  |
| --- | --- | --- |
| bit 0~ bit 5 | Bit 6 | Bit 7 |
| Function code | Frame type | Status |
| 0 ~ 0x1F | * 0: request * 1: response | * 0: OK; * 1: Error |

Currently support commands below:

* *CMD\_POLL* = 0x0,
* *CMD\_READ\_OBJ* = 0x01,
* *CMD\_WRITE\_OBJ* = 0x02,
* *CMD\_READ\_TYPE* = 0x03,
* *CMD\_READ\_LENGTH* = 0x04,
* *CMD\_READ\_ATR\_NUM* = 0x05,
* *CMD\_READ\_RANGE* = 0x06,
* *CMD\_READ\_NAME* = 0x07,
* *CMD\_READ\_ADRINFO*= 0x08,
* *CMD\_READ\_MEM* = 0x11,
* *CMD\_WRITE\_MEM* = 0x12,
* *CMD\_BURST\_VALUE* = 0x21,

|  |  |  |  |
| --- | --- | --- | --- |
| CMD name | Request Data(Multi-data parts in frame) | Response | Comments |
| CMD\_POLL | Byte 0: request id low bye;  Byte 1: request id high byte; | See application specific |  |
| CMD\_READ\_OBJ | Byte 0: Sub ID  Byte 1: Object ID  Byte 2: Attribute ID | Byte 0: Sub ID  Byte 1: Object ID  Byte 2: Attribute ID  Byte 3: Data 0 or error code  Byte 4: Data 1 | Attribute is -1(0xFF) means the whole object;  Attribute is other means the specific attribute |
| CMD\_READ\_TYPE |
| CMD\_READ\_LENGTH |
| CMD\_READ\_ATR\_NUM |
| CMD\_READ\_RANGE |
| CMD\_READ\_NAME |
| CMD\_READ\_ADRINFO |
| CMD\_WRITE\_OBJ | Byte 0: Sub ID  Byte 1: Object ID  Byte 2: Attribute ID  Byte 3: Data Byte 0  Byte 4: Data Byte 1 | Byte 0: Sub ID  Byte 1: Object ID  Byte 2: Attribute ID  Byte 3: Return code |  |
| CMD\_READ\_MEM | Byte 0~3: address  Byte 4~5: length (shall <= 512) | Byte 0~3: address  Byte 4~5: length (shall <= 512)  Byte 6: data 0 or error code  Byte 7: data 1; |  |
| CMD\_WRITE\_MEM | Byte 0~3: address  Byte 4~5: length (shall <= 512)  Byte 6: data 0;  Byte 7: data 1; | Byte 0~3: address  Byte 4~5: length (shall <=512)  Byte 6: error code |  |
| CMD\_BURST\_VALUE | Todo | Todo |  |

## CRC

Use crc16-ccit to handle the CRC check; the CRC is little endian in the CRC part of the frame. The initial value of CRC calculation is 0x1D0E;

Reference source code of C is attached below:

//crc16\_ccit\_table

//0x1021即：

//g(x) = x16+x12+x5+1

uint16\_t InitCRC16\_COMMON(void)

{

return 0x1D0E;

}

static const uint16\_t crc\_tab[256] =

{

0x0000, 0x1189, 0x2312, 0x329b, 0x4624, 0x57ad, 0x6536, 0x74bf,

0x8c48, 0x9dc1, 0xaf5a, 0xbed3, 0xca6c, 0xdbe5, 0xe97e, 0xf8f7,

0x1081, 0x0108, 0x3393, 0x221a, 0x56a5, 0x472c, 0x75b7, 0x643e,

0x9cc9, 0x8d40, 0xbfdb, 0xae52, 0xdaed, 0xcb64, 0xf9ff, 0xe876,

0x2102, 0x308b, 0x0210, 0x1399, 0x6726, 0x76af, 0x4434, 0x55bd,

0xad4a, 0xbcc3, 0x8e58, 0x9fd1, 0xeb6e, 0xfae7, 0xc87c, 0xd9f5,

0x3183, 0x200a, 0x1291, 0x0318, 0x77a7, 0x662e, 0x54b5, 0x453c,

0xbdcb, 0xac42, 0x9ed9, 0x8f50, 0xfbef, 0xea66, 0xd8fd, 0xc974,

0x4204, 0x538d, 0x6116, 0x709f, 0x0420, 0x15a9, 0x2732, 0x36bb,

0xce4c, 0xdfc5, 0xed5e, 0xfcd7, 0x8868, 0x99e1, 0xab7a, 0xbaf3,

0x5285, 0x430c, 0x7197, 0x601e, 0x14a1, 0x0528, 0x37b3, 0x263a,

0xdecd, 0xcf44, 0xfddf, 0xec56, 0x98e9, 0x8960, 0xbbfb, 0xaa72,

0x6306, 0x728f, 0x4014, 0x519d, 0x2522, 0x34ab, 0x0630, 0x17b9,

0xef4e, 0xfec7, 0xcc5c, 0xddd5, 0xa96a, 0xb8e3, 0x8a78, 0x9bf1,

0x7387, 0x620e, 0x5095, 0x411c, 0x35a3, 0x242a, 0x16b1, 0x0738,

0xffcf, 0xee46, 0xdcdd, 0xcd54, 0xb9eb, 0xa862, 0x9af9, 0x8b70,

0x8408, 0x9581, 0xa71a, 0xb693, 0xc22c, 0xd3a5, 0xe13e, 0xf0b7,

0x0840, 0x19c9, 0x2b52, 0x3adb, 0x4e64, 0x5fed, 0x6d76, 0x7cff,

0x9489, 0x8500, 0xb79b, 0xa612, 0xd2ad, 0xc324, 0xf1bf, 0xe036,

0x18c1, 0x0948, 0x3bd3, 0x2a5a, 0x5ee5, 0x4f6c, 0x7df7, 0x6c7e,

0xa50a, 0xb483, 0x8618, 0x9791, 0xe32e, 0xf2a7, 0xc03c, 0xd1b5,

0x2942, 0x38cb, 0x0a50, 0x1bd9, 0x6f66, 0x7eef, 0x4c74, 0x5dfd,

0xb58b, 0xa402, 0x9699, 0x8710, 0xf3af, 0xe226, 0xd0bd, 0xc134,

0x39c3, 0x284a, 0x1ad1, 0x0b58, 0x7fe7, 0x6e6e, 0x5cf5, 0x4d7c,

0xc60c, 0xd785, 0xe51e, 0xf497, 0x8028, 0x91a1, 0xa33a, 0xb2b3,

0x4a44, 0x5bcd, 0x6956, 0x78df, 0x0c60, 0x1de9, 0x2f72, 0x3efb,

0xd68d, 0xc704, 0xf59f, 0xe416, 0x90a9, 0x8120, 0xb3bb, 0xa232,

0x5ac5, 0x4b4c, 0x79d7, 0x685e, 0x1ce1, 0x0d68, 0x3ff3, 0x2e7a,

0xe70e, 0xf687, 0xc41c, 0xd595, 0xa12a, 0xb0a3, 0x8238, 0x93b1,

0x6b46, 0x7acf, 0x4854, 0x59dd, 0x2d62, 0x3ceb, 0x0e70, 0x1ff9,

0xf78f, 0xe606, 0xd49d, 0xc514, 0xb1ab, 0xa022, 0x92b9, 0x8330,

0x7bc7, 0x6a4e, 0x58d5, 0x495c, 0x3de3, 0x2c6a, 0x1ef1, 0x0f78

};

//--------------------------------------------------------------------------------------------------

/\*!

\brief update crc checksum

Reversed CCITT (X^16 + X^12 + X^5 + 1)

Single table lookup

\param data

\param crc previous crc checksum

\return updated crc checksum

\warning

\*/

//--------------------------------------------------------------------------------------------------

uint16\_t CalcCRC16\_COMMON(const uint8\_t data, const uint16\_t crc)

{

return (crc >> 8) ^ crc\_tab[data ^ (crc & 0xff)];

}

//--------------------------------------------------------------------------------------------------

/\*!

\brief update crc checksum

Reversed CCITT (X^16 + X^12 + X^5 + 1)

\author Paul Li

\param data

\param crc previous crc checksum

\return updated crc checksum

\*/

//--------------------------------------------------------------------------------------------------

uint16\_t CalcCrc16Mem\_COMMON(uint8\_t \* ptrData,uint16\_t crcInit ,uint16\_t length)

{

uint16\_t index = 0;

uint16\_t crc = crcInit;

for(index = 0; index<length; index++)

{

crc = CalcCRC16\_COMMON(ptrData[index],crc);

}

return crc;

}

CAN2.0

## Physical layer

Use Can 2.0 extend frames; follow ISO11898.

Fix can baud rate to be 125K;

## Head

Total 29 bit for can 2.0

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0~7 | 8~15 | 16~23 | 24~28 | 26~28 |
| Dst address | Src address | Ex | Frame type | REV |

## Address:

Refer to Serial port

## REV:

Not defined yet!

## Frame type

|  |  |
| --- | --- |
| 0x01 | multi frames start 🡪 the data part not have the frame ID, the EX is 0; |
| 0x02 | Multi frames 🡪 the EX parts will increase by 1; |
| 0x03 | Multi frames ends🡪 the EX parts will increase by 1; |
| 0x00 | No multi frames |

## Ex – multi-frames

The EX parts will increase from 0 when multi frames; the data is sent with 8 bytes max for one frame;

## Function code

Refer to Serial port

## Data

Refer to Serial port

## Example about Can Realization

Note: CRC is for the whole frames (not just length + data)

* The frames in serial port:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 byte | 1 byte | 1 byte | 1 byte | 2 bytes | Multi bytes | 2 byte |
| Address DST | Address SRC | ID | Function code | length | data | CRC |
| 0x05 | 0x03 | xx | FC | 0x10 0x00 | x0 x1 x2 c3 x4 x5 x6 x7 x8 x9 xa xb xc xd xe xf | crc0 crc1 |

* Frame like below:

0x05 0x03 xx FC 0x10 0x00 x0 x1 x2 c3 x4 x5 x6 x7 x8 x9 xa xb xc xd xe xf crc0 crc1

* Realize with Can:

Total 24 bytes; this shall be sent with 3 frames with can;

|  |  |  |
| --- | --- | --- |
| 8 bytes | 8 bytes | 8 bytes |
| 0x05 0x03 xx FC 0x10 0x00 x0 x1 | x2 c3 x4 x5 x6 x7 x8 x9 | xa xb xc xd xe xf crc0 crc1 |

1. Can Frame 1

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0~7 | 8~15 | 16~23 | 24~25 | 26~28 |
| Dst address | Src address | Ex | Frame type | REV |
| 0x05 | 0x03 | 0x0 | 0x01&0x03 | 0x00 |

Head1 -> 0x05 0x03 0x0 {0x01&0x03 0x00}

Data1-> 0x05 0x03 xx FC 0x10 0x00 x0 x1

1. Can Frame 2

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0~7 | 8~15 | 16~23 | 24~25 | 26~28 |
| Dst address | Src address | Ex | Frame type | REV |
| 0x05 | 0x03 | 0x01 | 0x02&0x03 | 0x00 |

Head2 ->0x05 0x03 0x01 {0x02&0x03 0x00}

Data2-> x2 c3 x4 x5 x6 x7 x8 x9

1. Can Frame 3

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 0~7 | 8~15 | 16~23 | 24~25 | 26~28 |
| Dst address | Src address | Ex | Frame type | REV |
| 0x05 | 0x03 | 0x02 | 0x03&0x03 | 0x00 |

Head3-> 0x05 0x03 0x02 {0x03&0x03 0x00}

Data3-> xa xb xc xd xe xf crc0 crc1

Design Decisions

* The can interface between measure board and UI board is reserved for further use. This interface is not planned yet.
* The Burst related functions are reserved for future.

Application Specific

## CMD\_POLL

*Request with two bytes, Response is different due to the request bytes:*

|  |  |  |
| --- | --- | --- |
| *Request ID* | *Comments* | *Response Data (little endian)* |
| *0x0000* | *Device ID ..etc* | *2 Bytes: Device ID -> 0xFFFF reserved for bootloader*  *2 Bytes: Device Status*  *4 bytes: Version Code*  *2 Bytes: Burst Memory Length* |
| *0x0001* | *Device Description* | *<=64 Bytes device name* |
| *… …* | *Leave to extend* |  |
| *0x0002* | *App information*  *(Only available for bootloader)* | *2 Bytes: Device ID*  *2 Bytes: Device Status*  *4 bytes: Version Code*  *2 Bytes: Burst Memory Length* |
| *… …* |  |  |
| *0x1000* | *Burst Address 0* | *4 bytes (address) + 1Byte Length* |
| *… …* |  |  |
| *0x100F* | *Burst Address 0x0F* | *4 bytes (address) + 1Byte Length* |
| *… …* |  |  |
|  | *Leave to extend* |  |
|  |  |  |

## CMD\_READ\_TYPE

*When request with the attribute == -1(0xFF), that means the whole object type; it response:*

enum E\_TYPE\_QUALIFIER

{

SIMPLE, //!< the DATA-OBJ is a simple data type e.g. int16\_t, FLOAT...

STRUCT, //!< structure of simple data type

ARRAY //!< array of simple data type

};

*When request with the attribute != -1(0xFF), that means the specified ttribute type; it response:*

public enum AtrType

{

ST\_NIL = 0, // no data eg. Actions

ST\_U8, // uint8\_t with defined range

ST\_E8, // enumeration uint8\_t

ST\_BIT8, // uint8\_t bit-field with a mask that defined which bits are allowed to be 1

ST\_CHAR, // char with defined range

ST\_NUMBER\_OF\_U8 = ST\_CHAR,

ST\_U16, // uint16\_t with defined range

ST\_I16, // int16\_t with defined range

ST\_WIDECHAR, // TWIDECHAR with defined range

ST\_BIT16, // TWIDECHAR with defined range

// fill in new simple types

// consider the RANGE-type and the corresponding length-tab

ST\_E16,

ST\_NUMBER\_OF\_U16 = ST\_E16,

ST\_U32, // uint32\_t with defined range

ST\_I32, // int32\_t with defined range

ST\_T32,//compressed time:

ST\_NUMBER\_OF\_U32 = ST\_T32,

ST\_FLOAT, // float with defined range

ST\_DOUBLE, // double with defined range

ST\_I64, // long int with defined range

ST\_U64, // long int with defined range

// must be the last enum

ST\_NUMBER\_OF\_SIMPLE\_TYPES

};

Note: ST\_T32 is one new type which is the compressing data for RTC:

**#define** YEAR 26

**#define** MONTH 22

**#define** DATE 17

**#define** HOUR 12

**#define** MINUTES 6

**#define** SECONDS 0

**#define** START\_YEAR 2000

ST\_T32 time = (uint32\_t)((Year – START\_YEAR)<<YEAR) | \

(uint32\_t)(Month<<MONTH) | \

(uint32\_t)(Date<<DATE) |\

(uint32\_t)(Hours<<HOUR) | \

(uint32\_t)(Minutes<<MINUTES) | \

(uint32\_t)(Seconds<<SECONDS);

## CMD\_READ\_LENGTH

*Response bytes is only 1;*

## CMD\_READ\_ATR\_NUM

*Response bytes is only 1;*

## CMD\_READ\_ADRINFO

*Response bytes is 6;defined as below:*

*{*

*U32 adr;*

*U16 len;*

*}*

## Read Enumeration Process

If not known the specific object, user could do the following actions to enumerate the whole subsystem or specified object;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 byte | 1 byte | 1 byte | 1 byte | 2 bytes | Multi bytes | 2 byte |
| Address DST | Address SRC | ID | Function code | length | data | CRC |
|  |  |  | 0x03 | 3 | X Y Z |  |

1. Read whole object type: X is subsystem id; Y is object id; Z is attribute(shall be 0xFF)

If error bit is set in response, means the object does not exist; or continue the next step;

1. Read whole object attribute number: X is subsystem id; Y is object id; Z is attribute(shall be 0xFF)
2. Read specific attribute type: X is subsystem id; Y is object id; Z is attribute(shall not be 0xFF)
3. Read specific data: X is subsystem id; Y is object id; Z is attribute(shall not be 0xFF for whole object type is structure)
4. Decode the raw data according to the attribute type;

## Put Object Process

If not known the specific object, user could do the following actions to enumerate the whole subsystem or specified object;

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| 1 byte | 1 byte | 1 byte | 1 byte | 2 bytes | Multi bytes | 2 byte |
| Address DST | Address SRC | ID | Function code | length | data | CRC |
|  |  |  | 0x02 |  | X Y Z |  |

1. Read specific attribute type: X is subsystem id; Y is object id; Z is attribute(shall not be 0xFF)
2. Encode the raw data according to the attribute type;
3. write specific data: X is subsystem id; Y is object id; Z is attribute(shall not be 0xFF for whole object type is structure)

Note: if the whole object type is structure (according to command “CMD\_READ\_TYPE” with attribute 0xFF), it is not recommended to write the whole object in one command.

1. *Error* *codes*

*Response bytes is only 1, when bit 7 of function code is not 0;*

typedef enum

{

OK = 0, //!< operation was successful

// this warnings are allowed to ignore, cause the subsystem isn’t influenced

WARNING=1, //!< ignorable codes

LESSTHAN\_RANGEMIN=1, //!< at least one value is less than its minimum value

GREATERTHAN\_RANGEMAX, //!< at least one value is greater than its maximum

RANGE\_TOO\_LOW, //!< one value of range parameter is less than the allowed minimum value

RANGE\_TOO\_HIGH, //!< one value of range parameter is greater than the maximum

UPPER\_RANGE\_TOO\_LOW, //!< upper value of range parameter is less than the allowed minimum value

UPPER\_RANGE\_TOO\_HIGH, //!< upper value of range parameter is greater than the maximum

LOWER\_RANGE\_TOO\_LOW, //!< lower value of range parameter is less than the allowed minimum value

LOWER\_RANGE\_TOO\_HIGH, //!< lower value of range parameter is greater than the maximum

SPAN\_TO\_SMALL, //!< then span between two values is too small

NOT\_ON\_GRID, //!< at least one value is not a multiple of its increment

RULE\_VIOLATION, //!< this shows a violation of one or more business rules

READ\_ONLY, //!< written parameter is read only

SENSOR\_INTERFACE\_ARM\_VIOLATION,

WRONG\_STATE, //!< written parameter is read only in current subsystem state

// this errors occur only while writing a parameter, the value was not written to the parameter

ERROR\_CODES=32, //!< start of not ignorable codes

WRITE\_ERROR=32, //!< write access denied

LESSTHAN\_RANGEMIN\_ERR=32, //!< at least one value is less than its minimum value

GREATERTHAN\_RANGEMAX\_ERR, //!< at least one value is greater than its maximum

RANGE\_TOO\_LOW\_ERR, //!< one value of range parameter is less than the allowed minimum value

RANGE\_TOO\_HIGH\_ERR, //!< one value of range parameter is greater than the maximum

UPPER\_RANGE\_TOO\_LOW\_ERR, //!< upper value of range parameter is less than the allowed minimum value

UPPER\_RANGE\_TOO\_HIGH\_ERR, //!< upper value of range parameter is greater than the maximum

LOWER\_RANGE\_TOO\_LOW\_ERR, //!< lower value of range parameter is less than the allowed minimum value

LOWER\_RANGE\_TOO\_HIGH\_ERR, //!< lower value of range parameter is greater than the maximum

SPAN\_TO\_SMALL\_ERR, //!< then span between two values is too small

NOT\_ON\_GRID\_ERR, //!< at least one value is not a multiple of its increment

RULE\_VIOLATION\_ERR=42, //!< this shows a violation of one or more business rules

READ\_ONLY\_ERR, //!< written parameter is read only

WRONG\_STATE\_ERR, //!< written parameter is read only in current subsystem state

EXCESS\_CORRECTION\_ATTEMPED\_ERR,

APPLIED\_PROCESS\_TOO\_LOW\_ERR,

APPLIED\_PROCESS\_TOO\_HIGH\_ERR,

SENSOR\_INTERFACE\_COMMUNICATION\_ERR,//!< fe hardware is not available due to an error

NEW\_LRV\_URV\_OUTSIDE\_SENSOR,

BOTH\_LRV\_URV\_OUTSIDE\_LIMIT,

BUSY\_TIME\_OUT,

// this error codes are impossible if the program work properly. During debugging

// this codes should stop the program-executing immediately

DEBUG\_ERROR=64, //!< beginning of implementation-errors

DECLINE=64, //!< DEBUG, operation isn’t permitted in the actual unit-state

ILLEGAL\_SUB\_IDX, //!< DEBUG, unknown Object

ILLEGAL\_OBJ\_IDX, //!< DEBUG, unknown Object

ILLEGAL\_ATTRIB\_IDX, //!< DEBUG, unknown Attribute

METHOD\_NOT\_SUPPORTED, //!< DEBUG, this data object does not support the called method

STATE\_DENIED, //!< DEBUG, actual state denied the requested state

ILLEGAL\_STATE, //!< DEBUG, the requested state is illegal

ILLEGAL\_DATACLASS\_INDEX,//!< DEBUG, unknown Data-Class

ERROR\_SPI\_DATA,

DEVICE\_BUSY\_STATE,

// this error codes occurs if proper function of the subsystem is guaranteed any longer

// e.g. the subsystem attributes are corrupted.

//@@ maybe an exception for fatal errors?

FATAL\_ERROR=96, //!< start of fatal-errors

ERROR\_NV\_STORAGE=96, //!< attributes inside the nv-storage are corrupted

ERROR\_RAM\_STORAGE, //!< attributes inside the ram are corrupted

ERROR\_HW, //!< hardware used by the subsystem doesn’t work

ERROR\_DATA\_LENGTH,

} FRESULT;

Bootloader Realization

## Supported Commands:

* *CMD\_POLL* = 0x0,

With request ID 0x0000; the response device ID is 0xFFFF means bootloader is running.

* *CMD\_WRITE\_OBJ* = 0x02,

🡺 Only support device reset CMD (put 1 11 0 X):

X = 6000, device reset with bootloader enabled

X != 0, reset device with no bootloader enabled;

X==0, no action

* *CMD\_READ\_MEM* = 0x11,
* *CMD\_WRITE\_MEM* = 0x12,

## To get device information:

Use CMD\_POLL

|  |  |  |
| --- | --- | --- |
| *0x0000* | *Device ID ..etc* | *2 Bytes: Device ID*  *2 Bytes: Device Status*  *4 bytes: Version Code*  *2 Bytes: Burst Memory Length* |
| *0x0001* | *Device Description* | *<=64 Bytes device name* |

## Step to Update Firmware

1. Cmd to enable bootloader functionality

Put 1 11 0 6000 // reset device to bootloader

1. Wait for 3 seconds until device is resets with bootloader;
2. Use CMD: *CMD\_WRITE\_MEM* to update the firmware;

The write address is from 0x08020000 to 0x08100000 🡪 the application address;

The cmd is recommend to send 512 bytes each time;

When the cmd is return with error code “DEVICE\_BUSY\_STATE” (73); the app needs to wait for at least 1500ms to send the next frame (the frame feedback with error code “DEVICE\_BUSY\_STATE” (73) is not required to send again);

1. Send CMD to reset after all data is updated.

Put 1 11 0 1 to reset device to application

Test Cases

1. Master Frame CRC failed, no feedback from this protocol
2. Valid CRC check:

There is one object used for test only: (subsystem 0, Object 0, attribute: x);

The object is defined as below:

**typedef** **struct** \_TEST\_VAL

{

uint8\_t testU8;

int8\_t testE8;

uint8\_t testBIT8;

int8\_t testCHAR;//byte 4

uint16\_t testU16;

int16\_t testI16;//byte 8

uint16\_t testWC;

uint16\_t testBIT16;//byte 12

int16\_t testE16; //byte 14

uint16\_t testU16\_1;//byte 16 for align

uint32\_t testU32;

int32\_t testI32;

uint32\_t testT32;

**float** testFloat;

**double** testDouble;

int64\_t testI64;

uint64\_t testU64;

} TEST\_VAL

With default value:

{

0,1,2,’3’,

4,5,6,7,8,9,

10,11,

12,13,14,

15,16,

};

* 1. Wrong or unsupported CMD:

Feedback with error code 0x2A – RULE\_VIOLATION\_ERR

* 1. CMD: CMD\_READ\_TYPE

For each attribute

Test to check type of each attribute;

Error Attribute:

Feedback with error code: 0x43 – ILLEGAL\_ATTRIB\_IDX

Error Object Idx (like 255)

Feedback with error code: 0x42 – ILLEGAL\_OBJ\_IDX

Error Subsystem Idx (like 255)

Feedback with error code: 0x41 – ILLEGAL\_SUB\_IDX

* 1. CMD: CMD\_READ\_LENGTH

For each attribute

Test to check length of each attribute;

Error Attribute:

Feedback with error code: 0x43 – ILLEGAL\_ATTRIB\_IDX

Error Object Idx (like 255)

Feedback with error code: 0x42 – ILLEGAL\_OBJ\_IDX

Error Subsystem Idx (like 255)

Feedback with error code: 0x41 – ILLEGAL\_SUB\_IDX

* 1. CMD: CMD\_READ\_ATR\_NUM

For this object

Test to check attribute number of this object;

Error Object Idx (like 255)

Feedback with error code: 0x42 – ILLEGAL\_OBJ\_IDX

Error Subsystem Idx (like 255)

Feedback with error code: 0x41 – ILLEGAL\_SUB\_IDX

* 1. CMD: CMD\_READ\_OBJECT

For each attribute

Test to check value of each attribute;

Error Attribute:

Feedback with error code: 0x43 – ILLEGAL\_ATTRIB\_IDX

Error Object Idx (like 255)

Feedback with error code: 0x42 – ILLEGAL\_OBJ\_IDX

Error Subsystem Idx (like 255)

Feedback with error code: 0x41 – ILLEGAL\_SUB\_IDX

* 1. CMD: CMD\_PUT\_OBJECT

For each attribute

Test to change value and read back the value of each attribute;

Error Attribute:

Feedback with error code: 0x43 – ILLEGAL\_ATTRIB\_IDX

Error Object Idx (like 255)

Feedback with error code: 0x42 – ILLEGAL\_OBJ\_IDX

Error Subsystem Idx (like 255)

Feedback with error code: 0x41 – ILLEGAL\_SUB\_IDX

This also test to verify the encoding and decoding functionalities;

* 1. CMD: CMD\_PUT\_MEMORY & CMD\_GET\_MEMORY

Use CMD\_POLL to get the burst memory and length;

Then test with Put/Get function with the burst memory.

* 1. Reset device to recover the device status and also the memory;

Use CMD\_PUT\_OBJECT to set the reset object (subsystem 1, object 11, and attribute 0) to be 1; PutObject 1 11 0 1

* 1. CMD: *CMD\_READ\_MEM & CMD\_WRITE\_MEM*

The ram address start from 0x2002EC00 with 1024Byte length are used for memory operation test. The default value is set as below:

*#define TEST\_BUFFER 1024*

*static uint8\_t ram\_test[TEST\_BUFFER]; //the ram\_test buffer is located at 0x2002EC00*

*for(uint16\_t i=0;i<TEST\_BUFFER;i++)*

*ram\_test[i] = (uint8\_t)I;*

Revision History

|  |  |  |  |
| --- | --- | --- | --- |
| **Rev.** | **Description of Version/Changes** | **Primary Author(s)** | **Date** |
| 0.1 | Initial revision. | Paul Li | **2016-12-23** |
| 0.2 | Add CRC calculation description | Paul Li | **2017-2-8** |
| 0.3 | Add CMD\_READ\_NAME;  Add application specific; | Paul Li | **2017-2-17** |
| 0.4 | Change the CMD definitions according to Code  Add test cases for the communication protocol | Paul Li | **2017-3-7** |
| 0.5 | Add one new type, and change the cmd length to be 2 bytes, max data length; | Paul Li | **2017-4-13** |
| 0.6 | After internal discussion, update Can frames and add can implementation examples | Paul Li | **2017-5-19** |
| 0.7 | Add bootloader realization & read/write mem test cases | Paul Li | **2017-6-27** |
| 0.8 | Change the description of read/write mem function | Paul Li | **2017-7-20** |
| 0.9 | Add poll function special for app information in bootloader functions | Paul Li | **2017-10-26** |
| 1.0 | Add CMD\_READ\_ADRINFO cmd | Paul Li | **2018-04-16** |