smartek vision

User Guide

RAW Commands

for HPSC4

Contents

1 Connection Settings

The following chapter includes basic preferences to be followed to successfully establish the connection and send commands to a strobe controller.

1.1 Ethernet Connection Parameters

When using the physical Ethernet interface, no matter if in a peer-to-peer or network setup, the communication is split into two phases. The UDP protocol is used for device discovery and IP configuration to find devices on the network and prepare them for TCP communication. As soon as a valid IP address has been assigned, the TCP connection can be established to apply the functional configuration.
Strobe controller is set in server mode listening on ports shown in Table 1.

Protocol	Port
UDP	30311
TCP	30313

Table 1: Network protocol port definition

Each message exchange between strobe controller and client must follow the Data Frame structure described in Section 2. Note that each Data Frame is sent as payload in TCP or UDP packet structure.

2 Data Frame Description

Regardless of the utilized physical connection, the Data Frame structure is retained as shown in Figure 1 below.

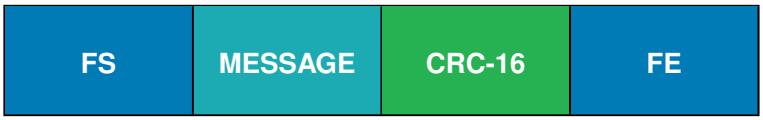

Figure 1: Data Frame structure

Description:

- **FS** - Frame Start = 0x01, uint8
- **MESSAGE** - Described in Section 2.1
- **CRC-16** - Cyclic Redundancy Check, CCITT(XMODEM), uint16
- **FE** - Frame End = 0x04, uint8

The maximum length of the data frame is 510 bytes, a cyclic redundancy check (*CRC-16*) is processed over the *MESSAGE* part of the Data Frame. A source code example is shown in the appendix A.1.

As the special characters *FS* and *FE* might also be repeated inside a Data Frame, the *Escape Character* ***0x10*** must be placed before any appearance of *FS*, *FE* or *Escape Character* itself. Figure 2 demostrates the use of the *Escape Character*.

	FS	MESSAGE	CRC-16	FE	Length
Basic Packet Content	0x01	0x00 0x01 0x02 0x26 0x04	0x10 0xF4	0x04	9 B
Packet Ready to Sent	0x01	0x00 **0x10** 0x01 0x02 0x26 **0x10** 0x04	**0x10** 0x10 0xF4	0x04	12 B

Figure 2: Using the Escape Character

The CRC checksum is calculated without Escape Characters. When a Data Frame is received the user must remove the *Escape Characters* during parsing. Further details can be seen in the attached source code at A.3 (73) and A.5 (64).

Data Types and Coding

The following chapters describe the telegram and payload structure. The actual values are stored into registers in differently coded datatypes:

- **uint8[n]** - uint8 array (n×1 byte(s)), written sequentially (uint8[0] represents first byte)
- **uint8[n] (string)** - uint8 array like uint8[n], containing ASCII coded text with NULL termination
- **uint32** - uint with 4 bytes, little-endian
- **float** - float (4 bytes), little-endian, coded according to IEEE-754 standard (Exponent + Mantissa)

All bytes are coded hexadecimal if not stated differently. little-endian defines that the first byte represents the LSB, the last one the MSB.

2.1 Message

Every *MESSAGE* inside a Data Frame begins with the Request command (REQ). Depending on REQ the rest of the data will be formed. The list below and Table 2 describe the possible *MESSAGE* blocks:

- REQ - *Request command for selecting operation*, uint8
- ACK - *Acknowledgement of successful reception*, uint8
- STATUS - *Status of executed command; OK = 1, NOK = 0*, uint32
- SN - *Strobe Controller serial number*, uint8[8]
- ADDR - *Start address*, uint32
- LEN - *Length of payload in bytes*, uint32
- PAYLOAD - *Part of the MESSAGE to be written into register*, uint8[LEN]; max length is 448 bytes

Name	REQ	ACK	Description
DISCOVERY	0x20	0xA0	Initiate discovery and identification of connected Strobe Controllers
WRITE_NET	0x27	0xA7	Write network parameters
READ_USR	0x40	0xC0	Read User parameters
WRITE_USR	0x41	0xC1	Write User parameters
SAVE_USR	0x42	0xC2	Save User parameters to flash
WRITE_CTRL	0x44	0xC4	Write control parameters

Table 2: Request commands overview

2.1.1 Discovery

The following request can be sent as a broadcast or unicast message. All Strobe Controllers receiving this message will send back their identification package as response, including the full DISCOVERY_PAYLOAD.

REQUEST:

- REQ = 0x20 (uint8)

RESPONSE:

- ACK = 0xA0 (uint8)
- LEN - Size of payload in bytes (uint32)
- DISCOVERY_PAYLOAD - Dataset described in Chapter 3.1, Table 3 in the given order

Example - Fetch Discovery Parameters

Legend:

- FS, FE
- MESSAGE
- CRC-16
- Escape Character
- 0xXX - Adresses of *Discovery Parameters* (Table 3)

REQUEST:

FS	REQ	CRC	FE
01	20	62 24	04

RESPONSE:

FS	ACK	LEN	0x00
01	A0	D4 00 00 00	53 6D 61 72 74 65 6B 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00

0x20
00 00 00 00 00 00 00 00 00 00 48 50 53 43 34 00 00 00 02 07 00 10 01 E8 FF BD 27 14 00 BF AF 8B CC

0x40	0x44	0x48
40 0F 21 20 00 00 14 00 BF 8F	02 07 00 10 01	00 00 10 01 10 01	FF FF FF FF FF 16 00 00

0x50	0x58	0x5C	0x60	0x64
6C D1 46 10 01 2F 16 00 00	32 42 02 10 01	10 01 00 00 00	10 04 00 00 00	10 04 00

0x68	0x6C	0x70	0x74	0x78	0x7C
00 00	00 00 20 42	00 00 20 42	00 00 00 00	00 00 48 42	00 00 16 43	00 00 A0 42

0x80	0x98
00 00 D0 40 00 00 C0 40 00 00 FA 42 55 6A 76 3A 00 87 93 03 FF FF FF FF	45 78 61 6D 70 6C

0xB8	0xBC
65 44 65 76 69 63 65 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00	0A 20 42 11	FF

0xC0	0xC4	0xC8	0xCC	0xD0
FF F0 00	10 01 00 00 00	0A 20 40 10 01	00 00 00 00	00 00 00 00	00 10 01 00 10 01

CRC	FE
56 92	04

Note Remove any Escape Characters before interpreting the payload. They are required in transmission to not interpret *0x01*, *0x10* and *0x04* as FS, FE or Escape Character by mistake.

2.1.2 Write Network Parameters

In order to change network parameters, a broadcast message containing the serial number **SN** of the target Strobe Controller in combination with a network payload describing the new parameters has to be sent.

REQUEST:

FS	REQ	SN	ADDR	LEN	NETWORK_PARAMS_PAYLOAD	CRC-16	FE

- REQ = 0x27 (uint8)
- SN - Serial number of target Strobe Controller (uint8[8])
- ADDR - Start address (uint32)
- LEN - Size of payload in bytes (uint32)
- NETWORK_PARAMS_PAYLOAD - described in Chapter 3.2, Table 4

RESPONSE:

FS	ACK	STATUS	CRC-16	FE

- ACK = 0xA7 (uint8)
- STATUS - OK=1, NOK=0 (uint32)

Example - Change device name to "DEVICE1"

Legend:
FS, FE
MESSAGE
CRC-16
Escape Character
0xXX - Adresses of *Network Parameters* (Table 4)

REQUEST:

FS	REQ	SN	ADDR	LEN	0x00	CRC	FE
01	27	6C D1 46 10 01 2F 37 00 00	00 00 00 00	08 00 00 00	44 45 56 49 43 45 31 00	4A DF	04

Note This request writes the first 8 bytes of the register 0x00, which is 32 bytes in total. The first 7 bytes contain the string while the last byte (0x00) terminates the string.

RESPONSE:

FS	ACK	STATUS	CRC	FE
01	A7	10 01 00 00 00	10 04 3B	04

2.1.3 Read User Parameters

Read User Register requests can be used to read all registers, a specific range or only a single register. A list of those can be found in Chapter 3.3, Table 5.

REQUEST:

FS	REQ	ADDR	LEN	CRC-16	FE

- REQ = 0x40 (uint8)
- ADDR - Start address (uint32)
- LEN - Number of bytes to read (uint32)

RESPONSE:

FS	ACK	LEN	USER_PARAMS_PAYLOAD	CRC-16	FE

- ACK = 0xC0 (uint8)
- LEN - Size of payload in bytes (uint32)
- USER_PARAMS_PAYLOAD - Content of the requested block of User Registers (Chapter 3.3, Table 5)

Example - Read LED Voltage for CH1, CH2, CH3, CH4

Legend:
FS, FE
MESSAGE
CRC-16
Escape Character
0xXX - Adresses of *User Parameters* (Table 5)

REQUEST:

FE	REQ	ADDR	LEN	CRC	FE
01	40	34 02 00 00	10 10 00 00 00	2C 6D	04

RESPONSE:

FS	ACK	LEN	0x0234	CRC	FE
01	C0	10 10 00 00 00	25 11 4F 41 00 00 00 00 00 00 00 00 00 00 00 00	3C 67	04

Result:
CH1 = 12.94 V
CH2 = 0 V
CH3 = 0 V
CH4 = 0 V

Note Float registers like 0x0234 (voltage) are little-endian and coded according to IEEE-754.

2.1.4 Write User Parameters

By writing to User Registers, a user can modify parameters and thus influence the behaviour of the Strobe Controller. The User Registers are described in Chapter 3.3, Table 5.

REQUEST:

FS	REQ	ADDR	LEN	USER_PARAMS_PAYLOAD	CRC-16	FE

- REQ = 0x41 (uint8)
- ADDR - Start address (uint32)
- LEN - Size of payload in bytes (uint32)
- USER_PARAMS_PAYLOAD - Data to be written into User Registers (Chapter 3.3, Table 5)

RESPONSE:

FS	ACK	STATUS	CRC-16	FE

- ACK = 0xC1 (uint8)
- STATUS - OK=1, NOK=0 (uint32)

Example 1 - Set device into Continuous mode

Legend:
FS, FE
MESSAGE
CRC-16
Escape Character
0xXX - Adresses of *User Parameters* (Table 5)

REQUEST:

FS	REQ	ADDR	LEN	0x00	CRC	FE
01	41	00 00 00 00	10 04 00 00 00	10 04 00 00 00	2F DA	04

RESPONSE:

FS	ACK	STATUS	CRC	FE
01	C1	10 01 00 00 00	5D EF	04

Example 2 - Set Max. Voltage CH1

Value = 15V

REQUEST:

FS	REQ	ADDR	LEN	0x08	CRC	FE
01	41	08 00 00 00	10 04 00 00 00	00 00 70 41	CA 5B	04

RESPONSE:

FS	ACK	STATUS	CRC	FE
01	C1	10 01 00 00 00	5D EF	04

Example 3 - Change Current of CH1, CH2, CH3, CH4

Values:

CH1 = 0.01 A	(0A D7 23 3C)
CH2 = 0.1 A	(CD CC CC 3D)
CH3 = 1 A	(00 00 80 3F)
CH4 = 5 A	(00 00 A0 40)

REQUEST:

FS	REQ	ADDR	LEN	0x38		CRC	FE
01	41	38 00 00 00	10	10 00 00 00	0A D7 23 3C CD CC CC 3D 00 00 80 3F 00 00 A0 40	24 7A	04

RESPONSE:

FS	ACK	STATUS		CRC	FE
01	C1	10	01 00 00 00	5D EF	04

2.1.5 Save User Parameters

All written registers of the User Parameters are only applied to the running configuration, they are not stored to permanent memory. To write the current state of the User Registers to permanent flash, the user can send the Save Register request.

Note Please note that due to the limited number of writing cycles supported by the physical flash memory (about 10 000), this option should be used carefully!

REQUEST:

FS	REQ	CRC-16	FE

- REQ = 0x42 (uint8)

RESPONSE:

FS	ACK	STATUS	CRC-16	FE

- ACK = 0xC2 (uint8)
- STATUS - OK=1, NOK=0 (uint32)

Example - Save User Parameters

Legend:
FS, FE
MESSAGE
CRC-16
Escape Character

REQUEST:

FS	REQ	CRC	FE
01	42	86 68	04

RESPONSE:

FS	ACK	STATUS	CRC	FE
01	C2	10 01 00 00 00	8F 10 01	04

2.1.6 Write Control Parameters

The Strobe Controllers provide additional control registers for controlling Start, Fire and Stop trigger states, where each register represents one trigger channel. Changing the state of Start/Stop is done by writing '1' or '0' respectively. When the *Software Trigger* mode is selected, writing '1' to control register will Fire a trigger signal, resulting in a single strobe pulse. The control registers are described in Chapter 3.4, Table 6.

REQUEST:

- REQ = 0x44 (uint8)
- ADDR - Start address (uint32)
- LEN - Size of payload in bytes (uint32)
- CTRL_PAYLOAD - Data to be written into Control registers

RESPONSE:

- ACK = 0xC4 (uint8)
- STATUS - OK=1, NOK=0 (uint32)

Example - Start/Fire CH2 trigger

Legend:

FS, FE
MESSAGE
CRC-16
Escape Character
0xXX - Adresses of *Control Parameters* (Table 6)

REQUEST:

FS	REQ	ADDR	LEN	0x04	CRC	FE
01	44	10 04 00 00 00	10 04 00 00 00	10 01 00 00 00	70 2B	04

RESPONSE:

FS	ACK	STATUS	CRC	FE
01	C4	10 01 00 00 00	0A CC	04

3 Register Descriptions

The following sections describe all available registers and their accessibility, allowing to configure the complete device. Each register map is dedicated to the equivalent command described in the sub-sections of Chapter 2.1.

3.1 Discovery

The following table contains the Discovery register map that is read from each reached device when executing the Discovery command described in Chapter 2.1.1.

Address	Name	Size	Type	R/W	Description
0x0000	Manufacturer Name	32	uint8[32] (string)	R	"Smartek"
0x0020	Model Name	32	uint8[32] (string)	R	"HPSCx"
0x0040	Application Firmware Version	4	uint8[4]	R	Firmware Version
0x0044	Format Version	4	uint8[4]	R	Format Version
0x0048	Serial Number	8	uint8[8]	R	Serial Number
0x0050	HW Address	8	uint8[8]	R	MAC Address
0x0058	HW Version	4	uint32	R	Hardware Version
0x005C	Switch Number	4	uint32	R	Number of Internal PS
0x0060	Channel Number	4	uint32	R	Number of Load Channels
0x0064	Trigger Number	4	uint32	R	Number of Triggers
0x0068	Max Continuous Current	4	float	R	Max Continuous Current [A]
0x006C	Max Trigger Current	4	float	R	Max Trigger Current [A]
0x0070	Min Voltage	4	float	R	Min Voltage [V]
0x0074	Max Voltage	4	float	R	Max Voltage [V]
0x0078	Max Input Power	4	float	R	Max Input Power [W]
0x007C	Max Temperature	4	float	R	Max Temperature [°C]
0x0080	RESERVED	4		R	Reserved
0x0084	RESERVED	4		R	Reserved
0x0088	RESERVED	4		R	Reserved
0x008C	RESERVED	4		R	Reserved
0x0090	RESERVED	4		R	Reserved
0x0094	RESERVED	4		R	Reserved
0x0098	Name	32	uint8[32] (string)	R	Arbitrary Name
0x00B8	IP Address	4	uint8[4]	R	IP Address
0x00BC	Subnet Mask	4	uint8[4]	R	Subnet Mask
0x00C0	DHCP Enable	4	uint32	R	0 - Permanent IP Address 1 - DHCP

Table 3: Discovery

Address	Name	Size	Type	R/W	Description
0x00C4	Default Gateway	4	uint8[4]	R	Default Gateway
0x00C8	Preferred DNS Server	4	uint8[4]	R	Preferred DNS Server
0x00CC	Alternate DNS Server	4	uint8[4]	R	Alternate DNS Server
0x00D0	FSBL Version	4	uint8[4]	R	FSBL Version

Table 3: Discovery

3.2 Network Parameters

The following table contains a register map of writable network parameters, that are accessible executing the Write Network Parameters command described in Chapter 2.1.2.

Address	Name	Size	Type	R/W	Description
0x0000	Name	32	uint8[32] (string)	R/W	Arbitrary Name
0x0020	IP Address	4	uint8[4]	R/W	IP Address
0x0024	Subnet Mask	4	uint32	R/W	Subnet Mask
0x0028	DHCP Enable	4	uint32	R/W	0 - Permanent IP Address 1 - DHCP
0x002C	Default Gateway	4	uint32	R/W	Default Gateway
0x0030	Preferred DNS Server	4	uint32	R/W	Preferred DNS Server
0x0034	Alternate DNS Server	4	uint32	R/W	Alternate DNS Server

Table 4: Network Parameters

3.3 User Parameters

The following table contains a register map of the device's operating parameters. The whole register range is accessible for reading and writing executing the Write and Read Network Parameters command described in Chapter 2.1.3 and 2.1.4.

Address	Name	CH	Index	Size	Type	R/W	Description
0x0000	Running Mode			4	uint32	R/W	1 - Off 2 - External Trigger 4 - Continuous 8 - Software Trigger 16 - External Switch 32 - RESERVED 64 - Internal Trigger
0x0004	Fault Code			4	uint32	R	0 - No Error 1 - Int. Bus Communication Error 2 - Not Used 3 - Wrong Parameters are Used 4 - Device Temperature too High 5 - Temperature Measuring Error 6 - D/A Converter Failure 7 - Input Power Supply Error 8 - Not Used
0x0008	Max Voltage	1	0	4	float	R/W	Set Max. Voltage in [V]
0x000C		2	1	4	float	R/W	
0x0010		3	2	4	float	R/W	
0x0014		4	3	4	float	R/W	
0x0018	Optimal Autosense	1	0	4	uint32	R/W	0 - Fixed Voltage 1 - Autosense ON
0x001C		2	1	4	uint32	R/W	
0x0020		3	2	4	uint32	R/W	
0x0024		4	3	4	uint32	R/W	
0x0028	Trigger	1	0	4	uint32	R/W	Select Trigger Input Channel
0x002C		2	1	4	uint32	R/W	
0x0030		3	2	4	uint32	R/W	
0x0034		4	3	4	uint32	R/W	
0x0038	Current	1	0	4	float	R/W	Set Load Current [mA]
0x003C		2	1	4	float	R/W	
0x0040		3	2	4	float	R/W	
0x0044		4	3	4	float	R/W	

Table 5: User Parameters

Address	Name	CH	Index	Size	Type	R/W	Description
0x0048	Trigger Mode	1	0	4	uint32	R/W	0 - Disabled 1 - Edge
0x004C		2	1	4	uint32	R/W	
0x0050		3	2	4	uint32	R/W	
0x0054		4	3	4	uint32	R/W	
0x0058	Trigger Edge	1	0	4	uint32	R/W	0 - Not Defined 1 - Positive Edge 2 - Negative Edge
0x005C		2	1	4	uint32	R/W	
0x0060		3	2	4	uint32	R/W	
0x0064		4	3	4	uint32	R/W	
0x0068	Trigger Active	1	0	4	uint32	R/W	0 - Disabled 1 - Enabled
0x006C		2	1	4	uint32	R/W	
0x0070		3	2	4	uint32	R/W	
0x0074		4	3	4	uint32	R/W	
0x0078	LED Delay Time	1	0	4	uint32	R/W	LED Delay Time [μs]
0x007C		2	1	4	uint32	R/W	
0x0080		3	2	4	uint32	R/W	
0x0084		4	3	4	uint32	R/W	
0x0088	LED On Time	1	0	4	uint32	R/W	LED On Time [μs]
0x008C		2	1	4	uint32	R/W	
0x0090		3	2	4	uint32	R/W	
0x0094		4	3	4	uint32	R/W	
0x0098	Off Time	1	0	4	uint32	R/W	Off Time [μs]
0x009C		2	1	4	uint32	R/W	
0x00A0		3	2	4	uint32	R/W	
0x00A4		4	3	4	uint32	R/W	
0x00A8	OUT Delay Time	1	0	4	uint32	R/W	Trigger OUT Delay Time [μs]
0x00AC		2	1	4	uint32	R/W	
0x00B0		3	2	4	uint32	R/W	
0x00B4		4	3	4	uint32	R/W	
0x00B8	OUT On Time	1	0	4	uint32	R/W	Trigger On Time [μs]
0x00BC		2	1	4	uint32	R/W	
0x00C0		3	2	4	uint32	R/W	
0x00C4		4	3	4	uint32	R/W	
0x00C8	Set Max Input Power			4	float	R/W	Set Max Input Power [W]
0x00CC	Set Max Temperature			4	float	R/W	Set Max Temperature [C]
0x00D0	RESERVED			16		R	Reserved
0x00E0	RESERVED			16		R	Reserved

Table 5: User Parameters

Address	Name	CH	Index	Size	Type	R/W	Description
0x00F0	RESERVED			16		R	Reserved
0x0100	RESERVED			256		R	Reserved
0x0200	Input Voltage			4	float	R	Input Voltage [V]
0x0204	Read Max Input Power			4	float	R	Read Max. Input Power [W]
0x0208	PCB Temperature			4	float	R	PCB Temperature [°C]
0x020C	Air Temperature			4	float	R	Internal Temperature [°C]
0x0210	Controller Temperature			4	float	R	Controller Temperature [°C]
0x0214	Output Voltage	1	0	4	float	R	Calculated Output Voltage [V]
0x0218		2	1	4	float	R	
0x021C		3	2	4	float	R	
0x0220		4	3	4	float	R	
0x0224	Measured Voltage	1	0	4	float	R	Measured Output Voltage [V]
0x0228		2	1	4	float	R	
0x022C		3	2	4	float	R	
0x0230		4	3	4	float	R	
0x0234	LED Voltage	1	0	4	float	R	Measured Voltage on LED [V]
0x0238		2	1	4	float	R	
0x023C		3	2	4	float	R	
0x0240		4	3	4	float	R	
0x0244	LED Current	1	0	4	float	R	Measured LED Current [A]
0x0248		2	1	4	float	R	
0x024C		3	2	4	float	R	
0x0250		4	3	4	float	R	
0x0254	Event Counter	1	0	4	uint32	R	Number of Fired Triggers
0x0258		2	1	4	uint32	R	
0x025C		3	2	4	uint32	R	
0x0260		4	3	4	uint32	R	

Table 5: User Parameters

3.4 Control Parameters

The following table contains the register map of available control parameters, used to send trigger signals to a device. The corresponding command structure is described in Chapter 2.1.6.

Address	Name	CH	Index	Size	Type	R/W	Description
0x0000	Trigger State	1	0	4	uint32	W	0 - Stop 1 - Start/Fire
0x0004		2	1	4	uint32	W	
0x0008		3	2	4	uint32	W	
0x000C		4	3	4	uint32	W	

Table 6: Control Parameters

4 Revision History

Ver.	Chapter	Changes	Date
1.0.0	All	Initial Document for HPSC4	2017-04-06

Table 7: Revision History

Appendices

Listings

A Code Examples

A.1 CRC-16

```
static const uint16_t crc_table[256] = {

0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5,
0x60c6, 0x70e7, 0x8108, 0x9129, 0xa14a, 0xb16b,
0xc18c, 0xd1ad, 0xe1ce, 0xf1ef, 0x1231, 0x0210,
0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c,
0xf3ff, 0xe3de, 0x2462, 0x3443, 0x0420, 0x1401,
0x64e6, 0x74c7, 0x44a4, 0x5485, 0xa56a, 0xb54b,
0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6,
0x5695, 0x46b4, 0xb75b, 0xa77a, 0x9719, 0x8738,
0xf7df, 0xe7fe, 0xd79d, 0xc7bc, 0x48c4, 0x58e5,
0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969,
0xa90a, 0xb92b, 0x5af5, 0x4ad4, 0x7ab7, 0x6a96,
0x1a71, 0x0a50, 0x3a33, 0x2a12, 0xdbfd, 0xcbdc,
0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03,
0x0c60, 0x1c41, 0xedae, 0xfd8f, 0xcdec, 0xddcd,
0xad2a, 0xbd0b, 0x8d68, 0x9d49, 0x7e97, 0x6eb6,
0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a,
0x9f59, 0x8f78, 0x9188, 0x81a9, 0xb1ca, 0xa1eb,
0xd10c, 0xc12d, 0xf14e, 0xe16f, 0x1080, 0x00a1,
0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c,
0xe37f, 0xf35e, 0x02b1, 0x1290, 0x22f3, 0x32d2,
0x4235, 0x5214, 0x6277, 0x7256, 0xb5ea, 0xa5cb,
0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447,
0x5424, 0x4405, 0xa7db, 0xb7fa, 0x8799, 0x97b8,
0xe75f, 0xf77e, 0xc71d, 0xd73c, 0x26d3, 0x36f2,
0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9,
0xb98a, 0xa9ab, 0x5844, 0x4865, 0x7806, 0x6827,
0x18c0, 0x08e1, 0x3882, 0x28a3, 0xcb7d, 0xdb5c,
0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0,
0x2ab3, 0x3a92, 0xfd2e, 0xed0f, 0xdd6c, 0xcd4d,
0xbdaa, 0xad8b, 0x9de8, 0x8dc9, 0x7c26, 0x6c07,
0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba,
0x8fd9, 0x9ff8, 0x6e17, 0x7e36, 0x4e55, 0x5e74,
0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
};


//CRC16-CCITT(XMODEM), poly=0x1021
uint32_t UTILS_CalculateCrc(uint8_t *data, uint32_t len)
{
    uint32_t i;
    uint16_t crc = 0;

    while(len--)
    {
        i = (*data++ ^ (crc >> 8)) & 0xFF;
```

```
58            crc = crc_table[i] ^ (crc << 8);
59        }
60
61        return (crc & 0xFFFF);
62    }
```

A.2 Defines

```
1
2  #define FS 01
3  #define FE 04
4  #define DLE 16
5
6  // Commands
7  typedef enum
8  {
9    CMD_REQ_DISCOVERY = 0x20,
10     CMD_REQ_WRITE_NET = 0x27,
11   CMD_REQ_READ_USR = 0x40,
12   CMD_REQ_WRITE_USR = 0x41,
13   CMD_REQ_SAVE_USR = 0x42,
14   CMD_REQ_WRITE_CTRL = 0x44,
15
16     CMD_ACK_DISCOVERY = 0xA0,
17     CMD_ACK_WRITE_NET = 0xA7,
18     CMD_ACK_READ_USR = 0xC0,
19   CMD_ACK_WRITE_USR = 0xC1,
20   CMD_ACK_SAVE_USR = 0xC2,
21     CMD_ACK_WRITE_CTRL = 0xC4
22
23 }T_COMMANDS;
```

A.3 Generate Request

```
unsigned int GenerateRequestCmd(T_COMMANDS cmd, unsigned char *sn, unsigned int reg,
            unsigned int len, char *value, char *dest)
{
  char buff[sclib_MAX_CMD_MSG];
  unsigned int buffLen = 0;
  unsigned int destLen = 0;
  unsigned short crc;

  buff[buffLen++] = cmd;

  switch (cmd)
  {
    case CMD_REQ_WRITE_MAN:
    case CMD_REQ_WRITE_NET:
      if (sn) {
        memcpy(buff + buffLen, sn, 8);
        buffLen += 8;
      }
    case CMD_REQ_READ_USR:
    case CMD_REQ_WRITE_USR:
    case CMD_REQ_WRITE_CTRL:
      if (len) {
        memcpy(buff + buffLen, &reg, 4);
        buffLen += 4;
        memcpy(buff + buffLen, &len, 4);
        buffLen += 4;
      }
      break;
    case CMD_REQ_SAVE_MAN:
      if (sn) {
        memcpy(buff + buffLen, sn, 8);
        buffLen += 8;
      }
      break;

    default:
      break;
  }

  switch (cmd)
  {
    case CMD_REQ_WRITE_USR:
    case CMD_REQ_WRITE_NET:
    case CMD_REQ_SAVE_USR:
    case CMD_REQ_SAVE_MAN:
    case CMD_REQ_JMP_TO_APP:
    case CMD_REQ_ERASE_FLASH:
    case CMD_REQ_FLASH_VERIFY:
    case CMD_REQ_PROGRAM_FLASH:
    case CMD_REQ_WRITE_CTRL:
      if (len && value) {
        memcpy(buff + buffLen, value, len);
        buffLen += len;
      }
      break;
    case CMD_REQ_WRITE_MAN:
      if (len && value) {
        memcpy(buff + buffLen, value, 16 + len);
        buffLen += (16 + len);
```

```
60          }
61          break;
62        default:
63          break;
64      }
65
66      // Calculate CRC for the frame.
67      crc = CalculateCrc(buff, buffLen);
68      buff[buffLen++] = (char)crc;
69      buff[buffLen++] = (char)(crc >> 8);
70
71      dest[destLen++] = FS;
72
73      // Form TxPacket. Insert DLE in the data field whereever FS and FE are present.
74      for(unsigned int i = 0; i < buffLen; i++)
75      {
76        if((buff[i] == FE) || (buff[i] == FS)
77            || (buff[i] == DLE))
78        {
79          dest[destLen++] = DLE;
80        }
81        dest[destLen++] = buff[i];
82      }
83
84      // FE: End of transmission
85      dest[destLen++] = FE;
86      //dest[destLen++] = '\0';
87
88      return destLen;
89  }
```

A.4 Generate Response

```
unsigned int GenerateResponseCmd(T_COMMANDS cmd, unsigned int len, char *value, char *dest)
{
  char buff[sclib_MAX_CMD_MSG];
  unsigned int buffLen = 0;
  unsigned int destLen = 0;
  unsigned short crc;

  buff[buffLen++] = cmd;

  switch (cmd)
  {
  case CMD_ACK_WRITE_USR:
  case CMD_ACK_WRITE_NET:
  case CMD_ACK_SAVE_USR:
  case CMD_ACK_SAVE_MAN:
  case CMD_ACK_JMP_TO_APP:
  case CMD_ACK_ERASE_FLASH:
  case CMD_ACK_FLASH_VERIFY:
  case CMD_ACK_PROGRAM_FLASH:
  case CMD_ACK_WRITE_CTRL:
  case CMD_ACK_WRITE_MAN:
    if (len && value) {
      memcpy(buff + buffLen, value, len);
      buffLen += len;
    }
    break;
  default:
    break;
  }

  // Calculate CRC for the frame.
  crc = CalculateCrc(buff, buffLen);
  buff[buffLen++] = (char)crc;
  buff[buffLen++] = (char)(crc >> 8);

  dest[destLen++] = FS;

  // Form TxPacket. Insert DLE in the data field whereever FS and FE are present.
  for (unsigned int i = 0; i < buffLen; i++)
  {
    if ((buff[i] == FE) || (buff[i] == FS)
      || (buff[i] == DLE))
    {
      dest[destLen++] = DLE;
    }
    dest[destLen++] = buff[i];
  }

  // FE: End of transmission
  dest[destLen++] = FE;
  //dest[destLen++] = '\0';

  return destLen;
}
```

A.5 Parse Frame

```
bool BuildRxFrame(char *dest, unsigned int &destLen, char *buff,
          unsigned int &buffLen, unsigned int maxBuffSize)
{

  static bool Escape = false;
  unsigned short crc;
  bool frameValid = false;


  while((buffLen > 0) && (frameValid == false))
  {
    buffLen --;

    if(destLen >= (maxBuffSize - 2))
    {
      destLen = 0;
    }

    switch(*buff)
    {


      case FS: //Start of header
        if(Escape)
        {
          // Received byte is not FS, but data.
          dest[destLen++] = *buff;
          // Reset Escape Flag.
          Escape = FALSE;
        }
        else
        {
          // Received byte is indeed a FS which indicates start of new frame.
          destLen = 0;
        }
        break;

      case FE: // End of transmission
        if(Escape)
        {
          // Received byte is not FE, but data.
          dest[destLen++] = *buff;
          // Reset Escape Flag.
          Escape = FALSE;
        }
        else
        {
          // Received byte is indeed a FE which indicates end of frame.
          // Calculate CRC to check the validity of the frame.
          if(destLen > 1)
          {
            crc = (dest[destLen-2]) & 0x00ff;
            crc = crc | ((dest[destLen-1] << 8) & 0xFF00);
            if((CalculateCrc(dest, (destLen-2)) == crc) && (destLen > 2))
            {
              // CRC matches and frame received is valid.
              frameValid = true;
            }
          }
```

```
60          }
61          break;
62
63
64          case DLE: // Escape character received.
65          if(Escape)
66          {
67            // Received byte is not ESC but data.
68            dest[destLen++] = *buff;
69            // Reset Escape Flag.
70            Escape = FALSE;
71          }
72          else
73          {
74            // Received byte is an escape character. Set Escape flag to escape next byte.
75            Escape = TRUE;
76          }
77          break;
78
79        default: // Data field.
80            dest[destLen++] = *buff;
81            // Reset Escape Flag.
82            Escape = FALSE;
83          break;
84
85      }
86      // Increment the pointer.
87      buff++;
88
89    }
90
91    return frameValid;
92  }
```