



User Guide
RAW Commands
for HPSC1 v2 HPSC2 HPSC4

Contents

1	Connection Settings	1
1.1	Ethernet Connection Parameters	1
2	Data Frame Description	2
2.1	Message	3
2.1.1	Discovery	3
2.1.2	Write Network Parameters	5
2.1.3	Read User Parameters	6
2.1.4	Write User Parameters	7
2.1.5	Save User Parameters	9
2.1.6	Write Control Parameters	10
3	Register Descriptions	11
3.1	Discovery	11
3.2	Network Parameters	13
3.3	User Parameters	14
3.4	Control Parameters	17
4	Revision History	18
	Appendices	19
A	Code Examples	20
A.1	CRC-16	20
A.2	Defines	22
A.3	Generate Request	23
A.4	Generate Response	25
A.5	Parse Frame	26

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 demonstrates 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 (54) 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 ($n \times 1$ byte(s)), written sequentially (uint8[0] represents first byte)
- **uint8[n] (string)** - uint8 array like uint8[n], containing ASCII coded text with NULL termination
- **uint16** - uint with 2 bytes, little-endian
- **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: FS REQ CRC-16 FE

- REQ = 0x20 (uint8)

RESPONSE: FS ACK LEN DISCOVERY_PAYLOAD CRC-16 FE

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

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.



- 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



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

Example - Change device name to "DEVICE1"

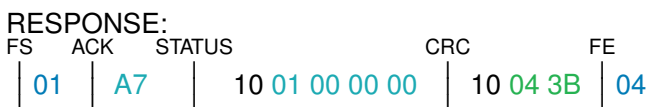
Legend:

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



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.



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)



Note When reading from User Registers depending on number of channels, user should read parameters only from actual number of channels.

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:

FS	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)



Note

When writing into User Registers depending on number of channels, user should write parameters only into actual number of channels.

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
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

CRC	FE
24 7A	04

RESPONSE:

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

Example 4 - Internal Trigger Mode

After Strobe Controller was set to *Internal Trigger* mode with user defined Voltage and Current, command for Trigger must be sent. For Trigger to start, '1' must be sent to respective register. For Trigger to stop, '0' must be sent to respective register. Example below shows command for starting Triggers on channel 1 and on channel 3.

Values:

CH1 = 1
 CH3 = 1

REQUEST:

FS	REQ	ADDR	LEN	0x68
01	41	68 00 00 00	10 10 00 00 00	10 01 00 00 00 00 00 00 00 10 01 00 00 00 00 00 00

CRC	FE
00 F2 97	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. After Save Register request, parameters will be restored on power cycle or on reset of the device.



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:



- REQ = 0x42 (uint8)

RESPONSE:



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

Example - Save User Parameters

Legend:

FS, FE
MESSAGE
CRC-16
Escape Character

REQUEST:



RESPONSE:



2.1.6 Write Control Parameters

The Strobe Controllers provide additional control registers for firing up Triggers in respective channels in *Software Trigger* mode. To fire up Trigger, '1' must be sent to respective register. After successful single strobe pulse, Strobe Controller will set registers to '0' automatically. The control registers are described in Chapter 3.4, Table 6.



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

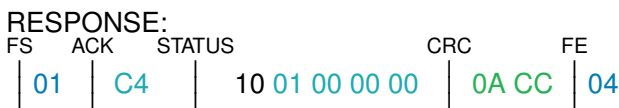
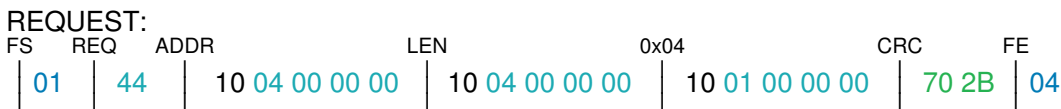


- ACK = 0xC4 (uint8)
- STATUS - OK=1, NOK=0 (uint32)

Example - Fire CH2 trigger

Legend:

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



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 [μ s]
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 [μ s]
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 [μ s]
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 [μ s]
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 [μ s]
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	1 - 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
1.1.0	All	Added support for HPSC1 v2, HPSC2	2018-04-11
	2.1.3	Added note for reading parameters from User Registers depending on number of channels	
	2.1.4	Added note for writing parameters into User Registers depending on number of channels, added example for <i>Internal Trigger</i>	
	2.1.5	Added note for Save Parameters function	
	2.1.6	Updated description	
	3.4	Updated description of Control Register	

Table 7: Revision History

Appendices

Listings

1	CRC-16	20
2	Defines	22
3	Generate Request	23
4	Generate Response	25
5	Parse Frame	26

A Code Examples

A.1 CRC-16

```

1 static const uint16_t crc_table[256] = {
2
3 0x0000, 0x1021, 0x2042, 0x3063, 0x4084, 0x50a5,
4 0x60c6, 0x70e7, 0x8108, 0x9129, 0xa14a, 0xb16b,
5 0xc18c, 0xd1ad, 0xe1ce, 0xf1ef, 0x1231, 0x0210,
6 0x3273, 0x2252, 0x52b5, 0x4294, 0x72f7, 0x62d6,
7 0x9339, 0x8318, 0xb37b, 0xa35a, 0xd3bd, 0xc39c,
8 0xf3ff, 0xe3de, 0x2462, 0x3443, 0x0420, 0x1401,
9 0x64e6, 0x74c7, 0x44a4, 0x5485, 0xa56a, 0xb54b,
10 0x8528, 0x9509, 0xe5ee, 0xf5cf, 0xc5ac, 0xd58d,
11 0x3653, 0x2672, 0x1611, 0x0630, 0x76d7, 0x66f6,
12 0x5695, 0x46b4, 0xb75b, 0xa77a, 0x9719, 0x8738,
13 0xf7df, 0xe7fe, 0xd79d, 0xc7bc, 0x48c4, 0x58e5,
14 0x6886, 0x78a7, 0x0840, 0x1861, 0x2802, 0x3823,
15 0xc9cc, 0xd9ed, 0xe98e, 0xf9af, 0x8948, 0x9969,
16 0xa90a, 0xb92b, 0x5af5, 0x4ad4, 0x7ab7, 0x6a96,
17 0x1a71, 0x0a50, 0x3a33, 0x2a12, 0xdbfd, 0xcbdc,
18 0xfbbf, 0xeb9e, 0x9b79, 0x8b58, 0xbb3b, 0xab1a,
19 0x6ca6, 0x7c87, 0x4ce4, 0x5cc5, 0x2c22, 0x3c03,
20 0x0c60, 0x1c41, 0xedae, 0xfd8f, 0xcdcc, 0xddcd,
21 0xad2a, 0xbd0b, 0x8d68, 0x9d49, 0x7e97, 0x6eb6,
22 0x5ed5, 0x4ef4, 0x3e13, 0x2e32, 0x1e51, 0x0e70,
23 0xff9f, 0xefbe, 0xdfdd, 0xcffc, 0xbf1b, 0xaf3a,
24 0x9f59, 0x8f78, 0x9188, 0x81a9, 0xb1ca, 0xa1eb,
25 0xd10c, 0xc12d, 0xf14e, 0xe16f, 0x1080, 0x00a1,
26 0x30c2, 0x20e3, 0x5004, 0x4025, 0x7046, 0x6067,
27 0x83b9, 0x9398, 0xa3fb, 0xb3da, 0xc33d, 0xd31c,
28 0xe37f, 0xf35e, 0x02b1, 0x1290, 0x22f3, 0x32d2,
29 0x4235, 0x5214, 0x6277, 0x7256, 0xb5ea, 0xa5cb,
30 0x95a8, 0x8589, 0xf56e, 0xe54f, 0xd52c, 0xc50d,
31 0x34e2, 0x24c3, 0x14a0, 0x0481, 0x7466, 0x6447,
32 0x5424, 0x4405, 0xa7db, 0xb7fa, 0x8799, 0x97b8,
33 0xe75f, 0xf77e, 0xc71d, 0xd73c, 0x26d3, 0x36f2,
34 0x0691, 0x16b0, 0x6657, 0x7676, 0x4615, 0x5634,
35 0xd94c, 0xc96d, 0xf90e, 0xe92f, 0x99c8, 0x89e9,
36 0xb98a, 0xa9ab, 0x5844, 0x4865, 0x7806, 0x6827,
37 0x18c0, 0x08e1, 0x3882, 0x28a3, 0xcb7d, 0xdb5c,
38 0xeb3f, 0xfb1e, 0x8bf9, 0x9bd8, 0xabbb, 0xbb9a,
39 0x4a75, 0x5a54, 0x6a37, 0x7a16, 0x0af1, 0x1ad0,
40 0x2ab3, 0x3a92, 0xfd2e, 0xed0f, 0xdd6c, 0xcd4d,
41 0xbdaa, 0xad8b, 0x9de8, 0x8dc9, 0x7c26, 0x6c07,
42 0x5c64, 0x4c45, 0x3ca2, 0x2c83, 0x1ce0, 0x0cc1,
43 0xef1f, 0xff3e, 0xcf5d, 0xdf7c, 0xaf9b, 0xbfba,
44 0x8fd9, 0x9ff8, 0x6e17, 0x7e36, 0x4e55, 0x5e74,
45 0x2e93, 0x3eb2, 0x0ed1, 0x1ef0
46 };
47
48
49 //CRC16-CCITT(XMODEM), poly=0x1021
50 uint32_t UTILS_CalculateCrc(uint8_t *data, uint32_t len)
51 {
52     uint32_t i;
53     uint16_t crc = 0;
54
55     while(len--)
56     {
57         i = (*data++ ^ (crc >> 8)) & 0xFF;

```

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

Listing 1: CRC-16

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 | }T_COMMANDS;
```

Listing 2: Defines

A.3 Generate Request

```

1  unsigned int GenerateRequestCmd(T_COMMANDS cmd, unsigned char *sn, unsigned int reg,
2      unsigned int len, char *value, char *dest)
3  {
4      char buff[sclib_MAX_CMD_MSG];
5      unsigned int buffLen = 0;
6      unsigned int destLen = 0;
7      unsigned short crc;
8
9      buff[buffLen++] = cmd;
10
11     switch (cmd)
12     {
13         case CMD_REQ_WRITE_NET:
14             if (sn) {
15                 memcpy(buff + buffLen, sn, 8);
16                 buffLen += 8;
17             }
18         case CMD_REQ_READ_USR:
19         case CMD_REQ_WRITE_USR:
20         case CMD_REQ_WRITE_CTRL:
21             if (len) {
22                 memcpy(buff + buffLen, &reg, 4);
23                 buffLen += 4;
24                 memcpy(buff + buffLen, &len, 4);
25                 buffLen += 4;
26             }
27             break;
28         default:
29             break;
30     }
31
32     switch (cmd)
33     {
34         case CMD_REQ_WRITE_USR:
35         case CMD_REQ_WRITE_NET:
36         case CMD_REQ_SAVE_USR:
37         case CMD_REQ_WRITE_CTRL:
38             if (len && value) {
39                 memcpy(buff + buffLen, value, len);
40                 buffLen += len;
41             }
42             break;
43         default:
44             break;
45     }
46
47     // Calculate CRC for the frame.
48     crc = CalculateCrc(buff, buffLen);
49     buff[buffLen++] = (char)crc;
50     buff[buffLen++] = (char)(crc >> 8);
51
52     dest[destLen++] = FS;
53
54     // Form TxPacket. Insert DLE in the data field wherever FS and FE are present.
55     for(unsigned int i = 0; i < buffLen; i++)
56     {
57         if((buff[i] == FE) || (buff[i] == FS)
58             || (buff[i] == DLE))
59     {

```

```
60     dest[destLen++] = DLE;
61   }
62   dest[destLen++] = buff[i];
63 }
64
65 // FE: End of transmission
66 dest[destLen++] = FE;
67 //dest[destLen++] = '\0';
68
69 return destLen;
70 }
```

Listing 3: Generate Request

A.4 Generate Response

```

1 unsigned int GenerateResponseCmd(T_COMMANDS cmd, unsigned int len, char *value, char *dest)
2 {
3     char buff[sclib_MAX_CMD_MSG];
4     unsigned int buffLen = 0;
5     unsigned int destLen = 0;
6     unsigned short crc;
7
8     buff[buffLen++] = cmd;
9
10    switch (cmd)
11    {
12    case CMD_ACK_WRITE_USR:
13    case CMD_ACK_WRITE_NET:
14    case CMD_ACK_SAVE_USR:
15    case CMD_ACK_WRITE_CTRL:
16        if (len && value) {
17            memcpy(buff + buffLen, value, len);
18            buffLen += len;
19        }
20        break;
21    default:
22        break;
23    }
24
25    // Calculate CRC for the frame.
26    crc = CalculateCrc(buff, buffLen);
27    buff[buffLen++] = (char)crc;
28    buff[buffLen++] = (char)(crc >> 8);
29
30    dest[destLen++] = FS;
31
32    // Form TxPacket. Insert DLE in the data field wherever FS and FE are present.
33    for (unsigned int i = 0; i < buffLen; i++)
34    {
35        if ((buff[i] == FE) || (buff[i] == FS)
36            || (buff[i] == DLE))
37        {
38            dest[destLen++] = DLE;
39        }
40        dest[destLen++] = buff[i];
41    }
42
43    // FE: End of transmission
44    dest[destLen++] = FE;
45    //dest[destLen++] = '\0';
46
47    return destLen;
48 }

```

Listing 4: Generate Response

A.5 Parse Frame

```

1  bool BuildRxFrame(char *dest, unsigned int &destLen, char *buff,
2      unsigned int &buffLen, unsigned int maxBuffSize)
3  {
4
5      static bool Escape = false;
6      unsigned short crc;
7      bool frameValid = false;
8
9
10     while((buffLen > 0) && (frameValid == false))
11     {
12         buffLen --;
13
14         if(destLen >= (maxBuffSize - 2))
15         {
16             destLen = 0;
17         }
18
19         switch(*buff)
20         {
21
22
23             case FS: //Start of header
24                 if(Escape)
25                 {
26                     // Received byte is not FS, but data.
27                     dest[destLen++] = *buff;
28                     // Reset Escape Flag.
29                     Escape = FALSE;
30                 }
31                 else
32                 {
33                     // Received byte is indeed a FS which indicates start of new frame.
34                     destLen = 0;
35                 }
36                 break;
37
38             case FE: // End of transmission
39                 if(Escape)
40                 {
41                     // Received byte is not FE, but data.
42                     dest[destLen++] = *buff;
43                     // Reset Escape Flag.
44                     Escape = FALSE;
45                 }
46                 else
47                 {
48                     // Received byte is indeed a FE which indicates end of frame.
49                     // Calculate CRC to check the validity of the frame.
50                     if(destLen > 1)
51                     {
52                         crc = (dest[destLen-2]) & 0x00ff;
53                         crc = crc | ((dest[destLen-1] << 8) & 0xFF00);
54                         if((CalculateCrc(dest, (destLen-2)) == crc) && (destLen > 2))
55                         {
56                             // CRC matches and frame received is valid.
57                             frameValid = true;
58                         }
59                     }
60                 }
61             }
62         }
63     }
64 }

```

```
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 }
```

Listing 5: Parse Frame