ICP Family Programmers

High-Level RS-232 Communication

Contents

1 History

- Version Mar-25:
  - standalone programming in paragraph 9.3: corrected item "Get programming results"
  - enum RES_STA renamed to STA_STATUS
  - structure ERR_RES renamed to RES_STAT
- Version Feb-25: initial version

2 Glossary

##	*Abbreviation*	*Meaning*
1.	PM	Program Memory (main flash, as in ICP for Windows)
2.	DM	Data Memory (data memory, as in ICP for Windows)
3.	Environment, sub-environment	See “Release Notes - Sub-Environments.pdf”
4.	RAM buffer	Every programmer channel has a volatile RAM buffer (256 bytes) which can be used for different purposes, for example to overwrite non-volatile contents of the environment. **WARNING:** be careful with RAM buffer manipulation since it changes your programming data. See paragraph 10 of “DLL Description.pdf” for more details

3 Installation

1.1 Software Installation

1.1.1 Software

- Visit our site and get the latest software: https://softlog.com/downloads/

1.1.2 ICP Family Standard Software

- The manual assumes that the ICP Family Windows software “ICP for Windows” is installed, minimum version 16.2.1a 20-Feb-25

1.2 Hardware Installation

1.2.1 Minimum firmware

- Firmware 38.2 or higher is required (comes with software 16.2.1a 20-Feb-25)

1.2.2 Activated Options

- DLL/Command Line Activation (D) is required

1.2.3 Follow ICP Family User’s Manual for hardware installation

4 Files

- The following file is required: fr_exp.h, it can be got from ICP for Windows - Help – Errorcodes or from the ICP DLL installed package

5 General

- Channel-by-channel operation is required for multi-channel programmers

6 Return Values (Errorcodes)

See enum AUTO_ERROR_LEVEL in file fr_exp.h

7 Communication Protocol

7.1 Parameters

##	*Parameter*	*Value*
1.	Signal levels for RS-232 connection	Standard RS-232 levels
2.	Baud rate	**115,200 for RS-232** (921.6K for USB, 460.8K for LAN) NOTE: RS-232 baud rate can be re-programmed to 460.8K for ICP2-COMBO and ICP2-ISO/LAN
3.	Flow control	Off
4.	Data bits	8
5.	Parity	No
6.	Stop bits	1
7.	Minimum delay between TX and RX	1ms (delay in firmware)

7.2 Communication Packets

- Host (Master) always initiates transmission
- ICP programmer (Slave) immediately answers

7.2.1 Master packet

Byte Position	***Mnemonics***	***Master Packet: Description***
0	QUAN	Packet length (in bytes) not including CS and itself
1	ADDR	Channel address 0x00…0x3F (physical channels 1…64) Note: 1-channel programmers have default address 0x01 (as physical channel 2)
2	HIGH-LEVEL ('H')	'H' (0x48=72dec) specifies high-level protocol
3	FUNC	High-level function
4…195	DATA_OUT	[BYTE 0]-Data byte 0 (optional) [BYTE 1]-Data byte 1 (optional) … [BYTE 191]-Data byte 191 (optional)
Last	CS	Checksum, calculated as 8-bit sum of all bytes excluding CS (MOD256)

7.2.2 Slave packet

Byte Position	***Mnemonics***	***Slave Packet: Description***
0	QUAN	Packet length (in bytes) not including CS and itself
1	ADDR	Channel address 0x00…0x3F (Slave address)
2	STAT	Status bits: 0x01 = BUSY 0x02…0x80 = Reserved
3	ERR_L	Result according to enum AUTO_ERROR_LEVEL, 0 means "OK"
4	ERR_H	
5…196	DATA_IN	[BYTE 0]-Data byte 0 (optional) [BYTE 1]-Data byte 1 (optional) … [BYTE 191]-Data byte 191 (optional)
Last	CS	Checksum, calculated as 8-bit sum of all bytes excluding CS (MOD256)

7.3 Function Summary (all Data Fields in HEX)

- Every function received by a programmer disables automatic background environment refresh (every hour). The environment refresh can be done by function H232_FUNC_ENV_SWITCH_START with required parameters

Function Number	***Mnemonics and Description***	***TX Data (DATA_OUT)***	***Possible Errorcodes***	***Status Bits to be Analyzed***	***RX Data (DATA_IN)***
	Base functions				
1	**H232_FUNC_STOP_CLEAR** Stop programming or clear error/result	[0]: 1=stop programming 2=clear programming result Other values are reserved	AUTO_H232_PROG_BUSY_ANOTHER AUTO_H232_PARAM_INVALID AUTO_STA_BUSY	BUSY	-
2	**H232_FUNC_GET_STATUS** Get busy status	-	-	BUSY	-
3	**H232_FUNC_GET_PROG_INFO** Get programmer info	[0]: number of info bytes to be read [1]: data offset. If the **both are 0** then entire structure is read	AUTO_H232_PARAM_INVALID	-	[0]…[191]: data according to structure H232_PROG_INFO
4	**H232_FUNC_GET_ENV_INFO_START** Start getting environment info	[0]: number of info bytes to be read [1]: data offset. If the **both are 0** then entire structure is read	AUTO_H232_PARAM_INVALID AUTO_PRJ_INVALID	BUSY	[0]…[191]: data according to structure H232_ENV_INFO
5	**H232_FUNC_ENV_SWITCH_START** Start environment switch/refresh process: - switch does nothing for the same number - use "refresh" for reloading the same	[0]: environment number [1]: sub-env. (L) [2]: sub-env. (H) [3]: 1=force refresh [4]: 1=use CRC	AUTO_H232_PROG_BUSY_ANOTHER AUTO_ENV_NUM_OUT_RANGE AUTO_H232_INVALID_THIS_PROG	BUSY	-

Function Number	*Mnemonics and Description*	*TX Data (DATA_OUT)*	*Possible Errorcodes*	*Status Bits to be Analyzed*	*RX Data (DATA_IN)*
	environment - sub-environment number must be 0 for ICP2-Portable	(additionally to CS)			
6	**H232_FUNC_STA_PROG_START** Start standalone programming	-	AUTO_PRJ_INVALID AUTO_H232_PROG_BUSY_ANOTHER AUTO_H232_PROG_CANT_START AUTO_STA_BUSY AUTO_SEC_CNT_INTEG AUTO_SEC_CNT_ZERO AUTO_SER_ERR	BUSY	-
7	**H232_FUNC_GET_PROG_RESULT** Get programming results	-	AUTO_H232_PROG_INVALID_RES, many erroneous results depending on MCU	BUSY	[0]…[191]: data according to structure RES_STAT
8	**H232_FUNC_FIRM_RESET_START** Start firmware reset (nothing to do with BUSY, wait at least 2sec)	-	-	BUSY	-
	Data manipulation functions				
9	**H232_FUNC_RAM_BUF_USAGE** RAM buffer usage. Note: use PM and DM buffer start (offset from 0, not from absolute addresses)	[0]…[3]-PM start address in MCU [4],[5]-number of bytes for PM [6]…[9]-DM start address in MCU [10],[11]-number of bytes for DM [12]-RAM buffer usage according to enum RAM_BUF_USAGE	AUTO_H232_PARAM_INVALID	-	-
10	**H232_FUNC_RAM_BUF_WRITE** Write data to RAM buffer (RAM buffer size is 256 bytes, see USER_RAM_SIZE)	[0],[1]: offset in buffer [2]: number of bytes to be written [3]…[130]: 128 data bytes max. in a portion	AUTO_H232_PARAM_INVALID	-	-
11	**H232_FUNC_RAM_BUF_READ** Read data from RAM buffer	[0],[1]: offset in buffer [2]: number of bytes to be read (128 max in a portion)	AUTO_H232_PARAM_INVALID	-	[0]…[127]: data from RAM buffer

8 Functions: Detailed Description

- Address field is shown for physical channel 2 (**address 0x01**)

8.1 Function 1 (parameter 0x01): Stop Programming (H232_FUNC_STOP_CLEAR)

- Operation: stop programming process
- Slave packet:
  - STAT: always BUSY
  - ERR: always 0
- BUSY polling is required

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	[5]
Value	0x04	0x01	0x48	0x01	**0x01**	0x4F

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x04	0x01	0x01	0x00	0x00	-	0x06

8.2 Function 1 (parameter 0x02): Clear Error/Result (H232_FUNC_STOP_CLEAR)

- Operation: clear programming error/result. Function "Get programming results" will return error after this function
- Slave packet:
  - STAT: always 0
  - ERR: always 0
- BUSY polling is not required

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	[5]
Value	0x04	0x01	0x48	0x01	**0x02**	0x50

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x04	0x01	0x00	0x00	0x00	-	0x05

8.3 Function 2: Get Busy Status (H232_FUNC_GET_STATUS)

- Operation: get BUSY status of a started operation
- Slave packet:
  - STAT: 0 (not busy = operation done) or 1 (busy = operation is in progress)
  - ERR: always 0

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	-	[4]
Value	0x03	0x01	0x48	0x02	-	0x4E

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x04	0x01	0x00 or 0x01	0x00	0x00	-	<CS>

8.4 Function 3: Get Programmer Info (H232_FUNC_GET_PROG_INFO)

- Operation: get programmer info of selected channel
- Parameters:
  - DATA_OUT[0]: number of data bytes to be read
  - DATA_OUT[1]: data offset.
  If the both are 0 then entire structure H232_PROG_INFO is read (Feb-25: 40 bytes)
- Slave packet:
  - STAT: always 0
  - ERR: 0 if parameters are OK
  - DATA_IN: entire structure H232_PROG_INFO or its part according to Master parameters
- BUSY polling is not required
- Example: get entire programmer info

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	[4], [5]	[6]
Value	0x05	0x01	0x48	0x03	0x00, 0x00	0x51

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	[5]…[0x2C]	[0x2D]
Value	0x2C	0x01	0x00	0x00	0x00	Structure H232_PROG_INFO	<CS>

8.5 Function 4: Get Environment Info (H232_FUNC_GET_ENV_INFO)

- Operation: get environment info of selected channel
- BUSY polling **before** this function is required
- Parameters:
  - DATA_OUT[0]: number of data bytes to be read
  - DATA_OUT[1]: data offset.
  If the both are 0 then entire structure H232_ENV_INFO is read (Feb-25: 64 bytes)
- Slave packet:
  - STAT: status (BUSY or not BUSY)
  - ERR: 0 if parameters are OK
  - DATA_IN: entire structure H232_PROG_INFO or its part according to Master parameters
- Example: get entire environment info

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	[4], [5]	[6]
Value	0x05	0x01	0x48	0x04	0x00, 0x00	0x52

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	[5]…[0x44]	[0x45]
Value	0x44	0x01	0x00	0x00	0x00	Structure H232_ENV_INFO	<CS>

8.6 Function 5: Start Environment Switch/Refresh (H232_FUNC_ENV_SWITCH_START)

- Operation: start environment switch process
- Parameters:
  - DATA_OUT[0]: environment number (0=env. 1, 1=env.2, …, 5=env.6)
  - DATA_OUT[1]: sub-environment (L) number (0=sub-env. 1, 1=sub-env.2, …)
  - DATA_OUT[2]: sub-environment (H) number (currently 0)
  - DATA_OUT[3]: 0=normal switch (no switch for same environment/sub-env. number), 1=refresh (force switch for same environment)
  - DATA_OUT[4]: 0=validate environment by checksum only, 1=validate by checksum and CRC
- Slave packet:
  - STAT: BUSY if no error and selected environment/sub-environment differs from the current one and no refresh
  - ERR: depends on many conditions
- BUSY polling is required
- Example: normal switch to physical environment 5.55 (previous environment differs from 5.55)

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]…[8]	[9]
Value	0x08	0x01	0x48	0x05	0x04, 0x36, 0x00, 0x00, 0x00	0x90

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x04	0x01	0x01	0x00	0x00	-	0x06

8.7 Function 6: Start Standalone Programming (H232_FUNC_STA_PROG_START)

- Operation: start standalone programming process
- Parameters: none
- Slave packet:
  - STAT: BUSY if no error
  - ERR: depends on many conditions
- BUSY polling is required
- Example: successful programming start

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	-	[4]
Value	0x03	0x01	0x48	0x06	-	0x52

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x04	0x01	0x01	0x00	0x00	-	0x06

8.8 Function 7: Get programming results (H232_FUNC_GET_PROG_RESULT)

- Operation: get programmer results
- BUSY polling before this function is required
- Parameters: none
- Slave packet:
  - STAT: BUSY or 0 if no error
  - ERR: many erroneous results depending on MCU
  - DATA_IN: entire structure RES_STAT (Feb-25: 19 bytes)
- Example: verification error (7)

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	-	[4]
Value	0x03	0x01	0x48	0x07	-	0x53

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	[5]…[0x17]	[0x18]
Value	0x17	0x01	0x00	0x07	0x00	Structure RES_STAT	<CS>

8.9 Function 8: Start Firmware Reset (H232_FUNC_FIRM_RESET_START)

- Operation: start firmware reset of selected channel
- **IMPORTANT**: physical channel 1 of multi-channel programmers selects communication interface, therefore don't operate other channels until reset of channel 1 is complete. Recommended sequence: send this command to all channels excluding channel 1 without delay, then send it to channel 1 and wait 3-4 seconds
- Slave packet:
  - STAT: always BUSY
  - ERR: always 0
- BUSY polling is not required, wait as specified above

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	-	[4]
Value	0x03	0x01	0x48	0x08	-	0x54

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x04	0x01	0x01	0x00	0x00	-	0x06

8.10 Function 9: RAM Buffer Usage (H232_FUNC_RAM_BUF_USAGE)

- Operation: define functionality of the RAM buffer
- Parameters:
  - DATA_OUT[0]…[3]: start address/offset (in bytes) of PM to be overwritten by the RAM buffer
  - DATA_OUT[4],[5]: length of overwritten block (PM) in bytes
  - DATA_OUT[6]…[9]: start address/offset (in bytes) of DM to be overwritten by the RAM buffer
  - DATA_OUT[10],[11]: length of overwritten block (DM) in bytes
  - DATA_OUT[12]: functionality of RAM buffer according to enum RAM_BUF_USAGE
- Slave packet:
  - STAT: always 0
  - ERR: 0 if RAM buffer length and usage parameters are OK. Note: start address/offset is not checked
- BUSY polling is not required
- Example: overwrite PM starting from address/offset 0x200, 5 bytes. In this case it will work as follows:
  - STM32U031C6 (MCU physical start address = 0x0800_0000): affected start address = 0x0800_0200
  - nRF52840-QFAA (MCU physical start address = 0): affected start address = 0x200

Master:

	QUAN	***ADDR***	***'H'***	***FUNC***	***DATA_OUT***	***CS***
Offset	[0]	[1]	[2]	[3]	[4]…[0x10]	[0x11]
Value	0x10	0x01	0x48	0x09	0x00, 0x02, 0x00, 0x00, 0x05, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x01	0x6A

Slave:

	QUAN	***ADDR***	***STAT***	***ERR_L***	***ERR_H***	***DATA_IN***	***CS***
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x04	0x01	0x00	0x00	0x00	-	0x05

8.11 Function 10 (0x0A): Write Data to RAM buffer (H232_FUNC_RAM_BUF_WRITE)

- Operation: write data to RAM buffer
- Parameters:
  - DATA_OUT[0],[1]: offset in RAM buffer
  - DATA_OUT[2]: number of bytes to be written (maximum portion 128 bytes)
  - DATA_OUT[3]…[130]: data bytes
- Slave packet:
  - STAT: always 0
  - ERR: 0 if parameters are OK
- BUSY polling is not required
- Example: write 5 bytes (0x01, 0x23, 0x45, 0x67, 0x89) to the beginning of the RAM buffer

Master:

	QUAN	***ADDR***	***'H'***	***FUNC***	***DATA_OUT***	***CS***
Offset	[0]	[1]	[2]	[3]	[4]…[0x0B]	[0x0C]
Value	0x0B	0x01	0x48	0x0A	0x00, 0x00, 0x05, 0x01, 0x23, 0x45, 0x67, 0x89	0xBC

Slave:

	QUAN	***ADDR***	***STAT***	***ERR_L***	***ERR_H***	***DATA_IN***	***CS***
Offset	[0]	[1]	[2]	[3]	[4]	-	[5]
Value	0x4	0x01	0x00	0x00	0x00	-	0x05

8.12 Function 11 (0x0B): Read Data from RAM buffer (H232_FUNC_RAM_BUF_READ)

- Operation: read data from RAM buffer
- Parameters:
  - DATA_OUT[0],[1]: offset in RAM buffer
  - DATA_OUT[2]: number of bytes to be read (maximum portion 128 bytes)
- Slave packet:
  - STAT: always 0
  - ERR: 0 if parameters are OK
  - DATA_IN: data from the RAM buffer
- BUSY polling is not required
- Example: read 5 bytes from the beginning of the RAM buffer

Master:

	QUAN	*ADDR*	*'H'*	*FUNC*	*DATA_OUT*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]…[6]	[7]
Value	0x06	0x01	0x48	0x0B	0x00, 0x00, 0x05	0x5F

Slave:

	QUAN	*ADDR*	*STAT*	*ERR_L*	*ERR_H*	*DATA_IN*	*CS*
Offset	[0]	[1]	[2]	[3]	[4]	[5]…[9]	[0x0A]
Value	0x09	0x01	0x00	0x00	0x00	0x01, 0x23, 0x45, 0x67, 0x89	0x63

9 How it Works

9.1 General

Most of operations are time-consuming, the following sequence is usually required:
- send and receive packets
- check errorcode for every packet and do related exception handling if the errorcode differs from 0
- if errorcode is 0 then check BUSY status by function H232_FUNC_GET_STATUS until operation is complete

9.2 Environment Switch

- No error and timeout analysis is shown

Step	*Operation*	

1.	**Optional: Validate that all channels are connected**
	
	for (all channels, one-by-one)
	{
	- Get programmer info H232_FUNC_GET_PROG_INFO
	- No analysis is required, it was done to check communication
	}	

2.	**Start environment switch process (select desired environment)**
	
	for (all channels, one-by-one)
	{
	Start environment switch H232_FUNC_ENV_SWITCH_START
	}	

3.	**Wait until the switch process is done (about 1 sec per 1MByte)**
	
	for (all channels, one-by-one)
	{
	- Check busy state by H232_FUNC_GET_STATUS
	- If all channels are not busy then continue to next step
	}	

4.	**Get environment info**
	for (all channels, one-by-one)
	{
	- Get environment info H232_FUNC_GET_ENV_INFO_START, no BUSY analysis is required due to previous BUSY analysis
	- Validate that environment is valid and contains expected data – analyze data in structure H232_ENV_INFO
	}	

9.3 Standalone Programming

Step	*Operation*	
1.	Assumption: environment switch was done, environments in all channels are valid	

2.	**Optional: Validate that all channels are connected**
	
	for (all channels, one-by-one)
	{
	- Get programmer info H232_FUNC_GET_PROG_INFO
	- No analysis is required, it was done to check communication
	}	

3.	**Start programming**
	
	for (all channels, one-by-one)
	{
	Start programming H232_FUNC_STA_PROG_START
	}	

4.	**Wait until the programming process is done (time depends on MCU)**
	
	for (all channels, one-by-one)
	{
	- Check busy state by H232_FUNC_GET_STATUS
	- If all channels are not busy then continue to next step
	}	

5.	**Get programming results**
	for (all channels, one-by-one)
	{
	- Get programming results H232_FUNC_GET_PROG_RESULT
	- Analyze results: . The results are valid if resStat == resSTA_DONE
	}	

10 Getting Started

10.1 Install ICP GUI software

- Install ICP GUI software “ICP for Windows” by running setup file “IcpWin_setup_X_XX.exe”

10.2 Run ICP for Windows

- Run ICP for Windows
- Execute firmware upgrade
- Create environment(s) and transfer to the programmer → programmer is ready for standalone operation

10.3 Install ICP DLL / Command Line (Optional)

- Install ICP Family DLL Software
- Copy fr_exp.h to your project

10.4 Test communication with ICP programmer using Docklight

- See Appendix A below
- You can keep ICP for Windows running, just close the COM: Communication – Close COM
- Run Docklight or other communication software
- Send and receive individual packets

10.5 Make your project

- Make your own project

11 Appendix A: Structures

- See file fr_exp.h for other structures

- enum STA_STATUS

```
enum STA_STATUS { //status of standalone operation
  resSTA_IDLE = 0, //idle state (result is not valid)
  resSTA_BUSY = 1, //busy (operation in-progress)
  resSTA_DONE = 2  //done
};
```

- Structure RES_STAT, use #pragma pack(1)

```
typedef struct { //***don't change, only add***
  unsigned char  resStat;      //status, see RES_STA
  unsigned short resAutoErr;   //result (auto error)
  unsigned long  resErrPmQuan; //number of PM errors
  unsigned short resIdQuan;    //number of ID errors
  unsigned short resDmQuan;    //number of DM errors
  unsigned short resFuQuan;    //number of FU errors
  unsigned short resPmParQuan; //number of PM parity errors
  unsigned short resBmQuan;    //number of BM errors
  unsigned short resOtpQuan;   //number of OTP errors
} RES_STAT;
```

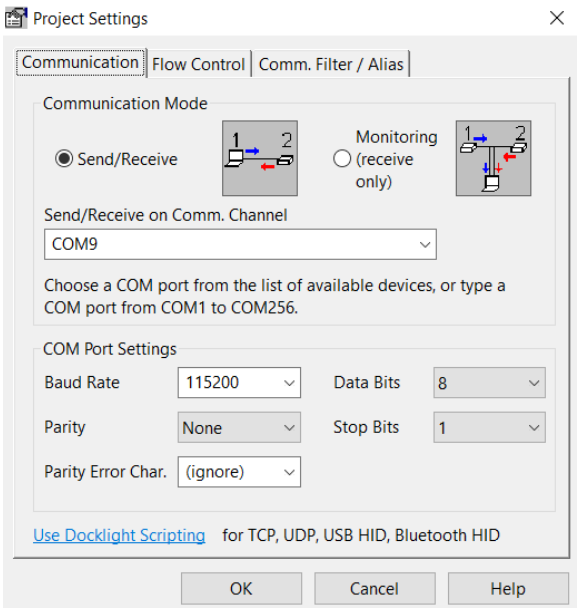
12 Appendix B: Docklight Example

12.1 Software

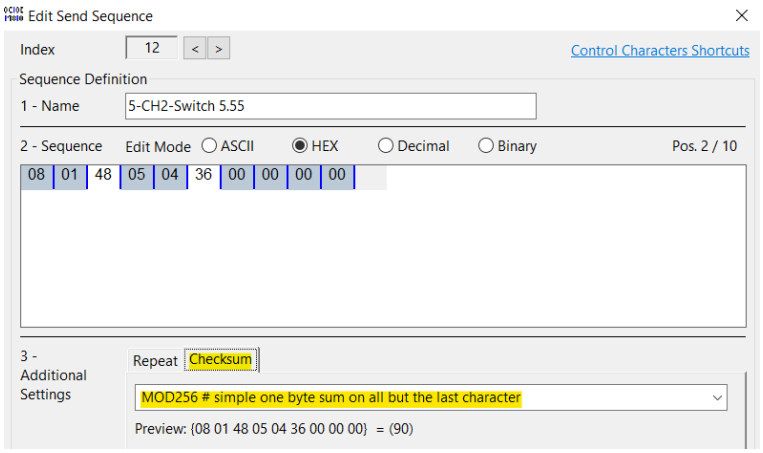
- Download and install Docklight base software from https://docklight.de/downloads/

12.2 Setup

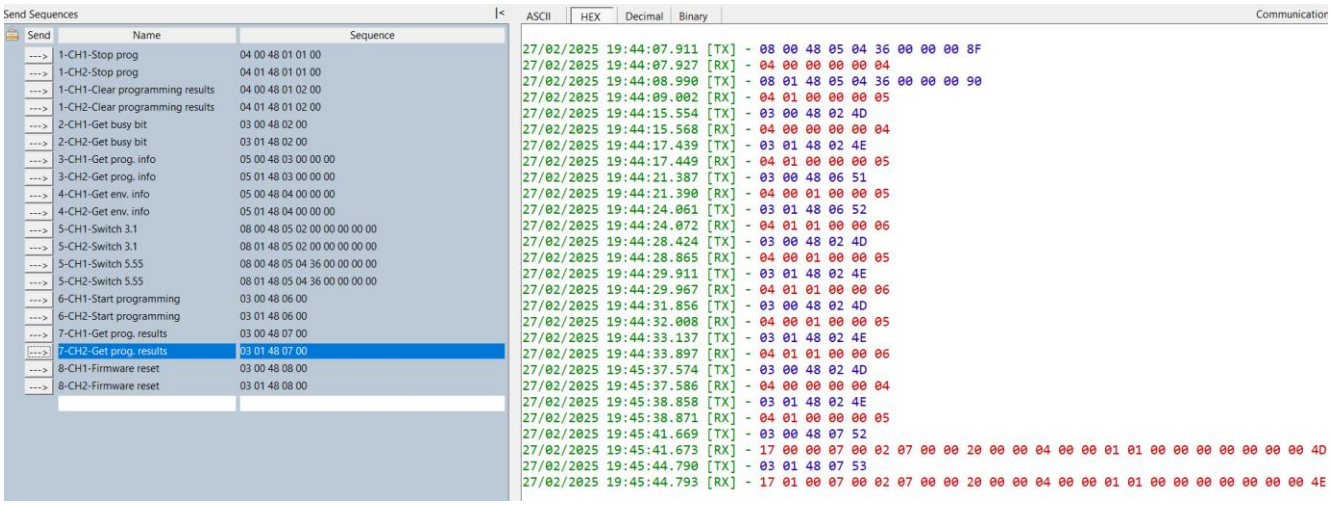
- Programmer: ICP2-GANG(G3)
- Communication interface: RS-232
- Settings (Tools – Project Settings):



- Packet (Send Sequence), example for function 5 (switch to environment 5.55), channel 2.
  Note: the last 00 byte is replaced with MOD256 checksum



- Example for sending different commands to physical channels 1 and 2:



13 Warranty

Softlog Systems (2006) Ltd. warrants this product against defects in materials and workmanship for a period of 1 (one) year. This warranty will not cover programmers that, in the opinion of Softlog Systems, have been damaged due to abuse, improper use, disassembly, replacement of parts or attempted repair by anyone other than an authorized Softlog Systems service technician.

This product must be returned to the supplier for warranty service within the stated period. The buyer shall pay all shipping costs and other charges or assessments for the product by the supplier.

Softlog Systems shall not be liable for any indirect, incidental, or consequential damages, regardless of whether liability is based upon breach of warranty, negligence, strict liability in tort, or any other theory, Softlog Systems will never be liable in an amount greater than the purchase price of the products described by this express warranty. No agent, distributor, salesperson, or wholesale or retail dealer has the authority to bind Softlog Systems to any other affirmation, representation, or warranty concerning these goods.

14 Contact

Softlog Systems (2006) Ltd.

6 Hayotzrim St. Or-Yehuda 6021820 Israel

Phone: 972-3-9515359
Fax: 972-3-9527520
Web: www.softlog.com
E-mail: sales@softlog.com, support@softlog.com

15 Copyright Notice