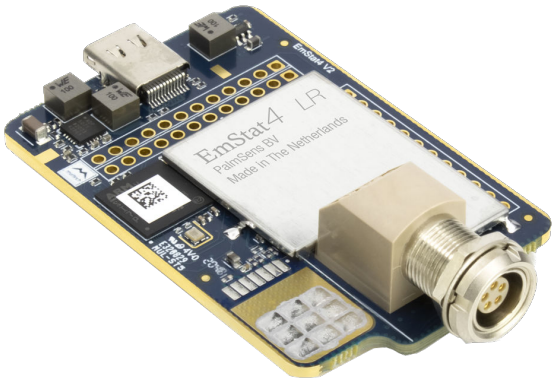
PalmSens
Compact Electrochemical Interfaces



Communication protocol for EmStat4

Version 1.2, 2023-01-24

Communication protocol for EmStat4

Last document update: 2023-01-24

Table of Contents

1. Introduction . . . 1
   1.1. Terminology . . . 1
2. Communication . . . 2
   2.1. Connection viewer . . . 2
   2.2. Communication protocol . . . 3
   2.3. Communication modes . . . 3
3. Command summary . . . 4
4. Command details . . . 5
   4.1. Get firmware version (`t`) . . . 5
   4.2. Set register (`S`) . . . 6
   4.3. Get register (`G`) . . . 7
   4.4. Load MethodSCRIPT (`l`) . . . 8
   4.5. Run loaded MethodSCRIPT (`r`) . . . 9
   4.6. Execute (= load and run) MethodSCRIPT (`e`) . . . 10
   4.7. Store loaded MethodSCRIPT to NVM (`Fmscr`) . . . 11
   4.8. Load MethodSCRIPT from NVM (`Lmscr`) . . . 12
   4.9. Hibernate (*deprecated*) (`s`) . . . 12
   4.10. Get serial number (`i`) . . . 13
   4.11. Get multi-channel serial number (`m`) . . . 14
   4.12. Get MethodSCRIPT version (`v`) . . . 15
   4.13. Enter bootloader (`dlfw`) . . . 16
   4.14. Get directory listing (`fs_dir`) . . . 16
   4.15. Read file (`fs_get`) . . . 17
   4.16. Write file (`fs_put`) . . . 18
   4.17. Delete file or directory (`fs_del`) . . . 19
   4.18. Get file system information (`fs_info`) . . . 20
   4.19. Format storage device (`fs_format`) . . . 21
   4.20. Mount file system (`fs_mount`) . . . 22
   4.21. Unmount file system (`fs_unmount`) . . . 22
   4.22. Clear file system (`fs_clear`) . . . 22
   4.23. Get runtime capabilities (`CC`) . . . 23
   4.24. Get MethodSCRIPT capabilities (`CM`) . . . 23
   4.25. Halt script execution (`h`) . . . 24
   4.26. Resume script execution (`H`) . . . 24
   4.27. Abort script execution (`Z`) . . . 25
   4.28. Abort measurement loop (`Y`) . . . 25
5. Register summary . . . 29

5.1. Generic registers . . . 29
5.2. EmStat4 specific registers . . . 29
6. Register details . . . 30
6.1. Peripheral configuration (0x01) . . . 30
6.2. Permission level (0x02) . . . 30
6.3. License register (0x04) . . . 31
6.4. Unique instrument ID (0x05) . . . 31
6.5. Device serial number (0x06) . . . 32
6.6. MethodSCRIPT autorun (0x08) . . . 32
6.7. Advanced options (0x09) . . . 33
6.8. UART data rate limit (0x0A) . . . 34
6.9. Reset instrument (0x0B) . . . 34
6.10. Multi-channel role (0x0D) . . . 35
6.11. System date and time (0x0E) . . . 36
6.12. Default GPIO config (0x0F) . . . 36
6.13. System warning (0x10) . . . 37
6.14. NVM commit (0x81) . . . 37
6.15. Multi-channel serial (0x87) . . . 38
6.16. AUX DAC gain (0x88) . . . 38
6.17. Baud rate configuration (0x89) . . . 38
7. CRC16 protocol extension . . . 40
7.1. Introduction . . . 40
7.2. Line format . . . 40
7.3. Acknowledge messages . . . 41
7.4. Other changes . . . 41
7.5. Examples . . . 41
8. Error handling . . . 43
9. Version changes . . . 45
Version 1.0 . . . 45
Version 1.2 . . . 45
Appendix A: Error codes . . . 46
Appendix B: MethodSCRIPT capabilities bit fields . . . 51
Appendix C: Communication capabilities bit fields . . . 55

Communication protocol for EmStat4
Last document update: 2023-01-24

Chapter 1. Introduction

This document describes the “online” communication protocol of the EmStat4.

Initial communication with an EmStat is always done using this online communication. Measurements and other scripts can be started by sending a MethodSCRIPT, more information about MethodSCRIPT can be found here: http://www.palmsens.com/methodscript

1.1. Terminology

PGStat	Potentiostat / Galvanostat
EmStat	PGStat device series by PalmSens
CE	Counter Electrode
RE	Reference Electrode
WE	Working Electrode
Technique	A standard electrochemical measurement technique
Iteration	A single execution of a loop
Int	Integer value
Float	Floating-point number (e.g. 3.14)
SI	International System of Units
Var	(MethodSCRIPT) variable (usually command input)
HEX	Hexadecimal (= base 16) number (e.g. 0xA1)
RAM	The (volatile) work memory of the instrument, which is lost after a power cycle
NVM	Non-Volatile Memory, i.e. memory that retains its contents after a power cycle
CRC	Cyclic Redunancy Check, an error-detecting code
CRC16	A 16-bit CRC

Chapter 2. Communication

The EmStat4 has two communication interfaces: USB and UART (Serial Port). For USB, the CDC protocol is used, which means that the device identifies itself as a device with a (virtual) serial port. This is also called a Virtual COM Port (VCP). Although the virtual COM port has a number of options that can be configured, for the USB interface these options do not have any effect. For the actual serial port (UART), the following settings should be used to connect with the instrument. Note that the bootloader uses slightly different settings than the application firmware. Normally the application firmware is used. The bootloader is only used for maintenance tasks such as firmware updates.

Table 1. EmStat4 UART connection details.

Property	Bootloader	Application
Signal level	3.3 V	
Baud rate	230400 bps	921600 bps[1]
Number of data bits	8	
Number of stop bits	1	
Parity	None	
Flow control	None	Hardware (RTS/CTS)

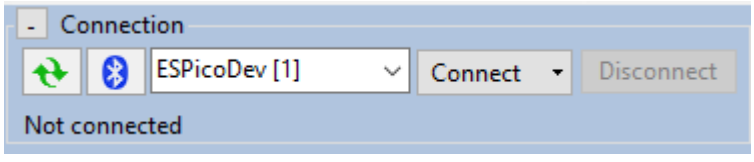
[1] Default baud rate. This can be configured.

The EmStat4 firmware uses RTS/CTS (hardware) flow control. It is highly recommended to enable RTS/CTS flow control on the host side as well. This ensures a reliable communication, even at high speeds and when the instrument or host is busy with other tasks. Connecting and using RTS/CTS is optional, but without flow control communication errors can occur on higher communication speed. If flow control is not used, the RTS/CTS lines must remain unconnected (high impedance).

2.1. Connection viewer

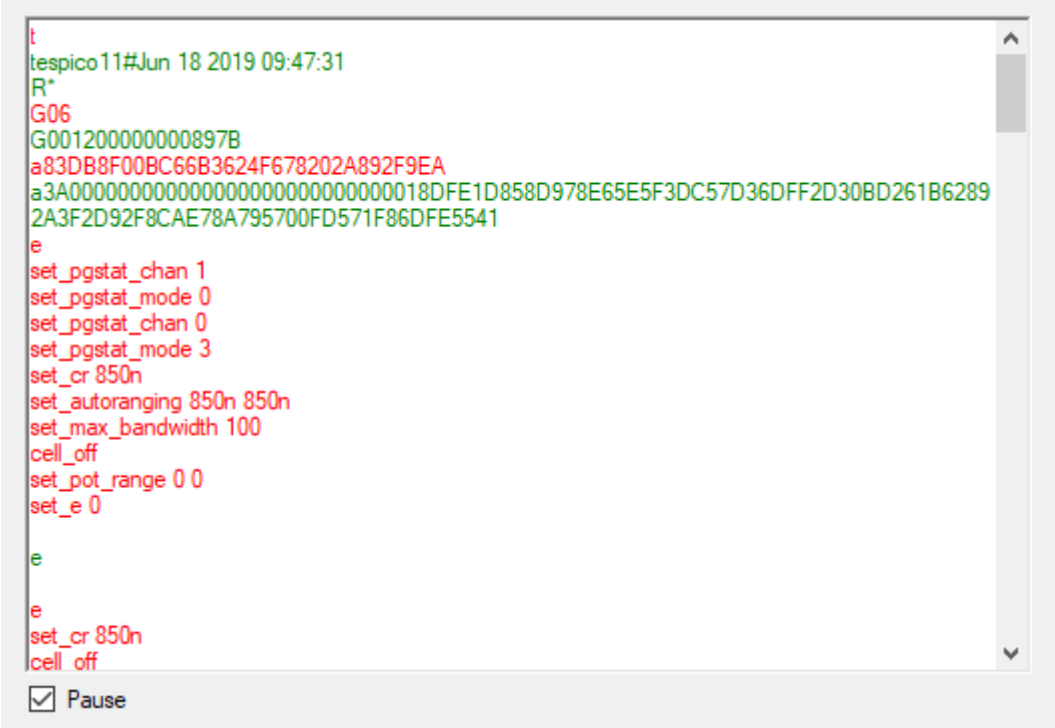
PSTrace version 5.6 or higher has a hidden feature that is useful when the communication protocol is used for development of software for the EmStat4. PSTrace will open the *Connection viewer* window when you double-click on the "Not connected" label before connecting to the device.

The "Not connected" label used to activate the Connection Viewer window.



Once connected, the connection viewer window will show all messages transmitted to the instrument (in red), and messages received from the instrument (in green). This can be helpful to understand the communication between the host and the instrument. Below is an example of the connection viewer window. Note that PSTrace is connected to an EmStat Pico in this example.

The connection viewer window.



2.2. Communication protocol

All commands and responses are terminated with a newline character. The used newline character is the Line Feed (`LF`) character (`'\n'`, ASCII code `10` or `0x0A`). The instrument never transmits a Carriage Return (CR) character (`'\r'`, ASCII code `13` or `0x0D`) and `CR` characters received by the instrument are ignored.

When a command is received by the instrument, it will echo the first character of the command and then respond with the command-specific data. After executing the command, a newline character is transmitted. If an error occurs during the execution of a command, the error is returned just before the newline character. See section Chapter 8, *Error handling* for more information about errors.

2.3. Communication modes

The device can be in two communication modes on which a subset of commands are available. These modes are listed below.

- Idle mode: for storing scripts and changing settings
- Script execution mode: during script execution

Chapter 3. Command summary

The following table gives an overview of all communication protocol commands.

ID	Command	Modes	Description
0x01	`t`	All modes	Get firmware version
0x20	`CC`	Idle	Get runtime capabilities
0x21	`CM`	Idle	Get MethodSCRIPT capabilities
0x22	`S`	Idle	Set register
0x23	`G`	Idle	Get register
0x24	`l`	Idle	Load MethodSCRIPT
0x25	`r`	Idle	Run loaded MethodSCRIPT
0x26	`e`	Idle	Execute (= load and run) MethodSCRIPT
0x27	`dlfw`	Idle	Enter bootloader
0x2B	`Fmscr`	Idle	Store loaded MethodSCRIPT to NVM
0x2C	`Lmscr`	Idle	Load MethodSCRIPT from NVM
0x2E	`s`	Idle	Hibernate (*deprecated*)
0x30	`i`	Idle	Get serial number
0x31	`v`	Idle	Get MethodSCRIPT version
0x33	`fs_dir`	Idle	Get directory listing
0x34	`fs_get`	Idle	Read file
0x35	`fs_put`	Idle	Write file
0x36	`fs_del`	Idle	Delete file or directory
0x37	`fs_info`	Idle	Get file system information
0x38	`fs_format`	Idle	Format storage device
0x39	`fs_mount`	Idle	Mount file system
0x3A	`fs_unmount`	Idle	Unmount file system
0x3B	`fs_clear`	Idle	Clear file system
0x3C	`m`	Idle	Get multi-channel serial number
0x60	`h`	Script	Halt script execution
0x61	`H`	Script	Resume script execution
0x62	`Z`	Script	Abort script execution
0x63	`Y`	Script	Abort measurement loop

Chapter 4. Command details

A list of all commands is given in the previous chapter. In this chapter, each commmand is described in more detail.

Some commands have one or more arguments. The format and meaning of such arguments is documented in those sections as well.



Commands are case-sensitive. For example, `s` (hibernate) is a different command than `S` (Set register).

4.1. Get firmware version (`t`)

Get the device firmware version. This includes the device type, firmware version, build date and release type.

Command format

```
t
```

Response format

Unlike most other commands, this command has a response consisting of multiple lines. The last line is terminated with an asterisk and a newline character (`'*\n'`). The format is as follows:

```
tddddddvv..vv#mmm dd yyyy hh:mm:ss
R*
```

Key	Type	Size	Description
`dddddd`	text	6	The device type. For the EmStat4 HR this is `es4_hr`. For the EmStat4 LR this is `es4_lr`.
`vv..vv`	text	2/4	The firmware version. This could be either: • a 2-digit version identifier `xy`, denoting firmware version `x.y` (e.g. `10` corresponds to firmware version 1.0) • a 4-digit version identifier `xyzz`, denoting firmware version `x.y.zz` (e.g. `1201` corresponds to firmware version 1.2.01).
`mmm dd yyyy hh:mm:ss`	text	20	The build date and time.
`R`	text	1	The release type: • `R` for Release versions. • `B` for Beta versions.
`*`	text	1	Marks the end of the response.

Example

Below are some example to demonstrate the format of the output.

Example output for an EmStat4 LR with firmware v1.0.00

```
tes4_lr1000#Jun  7 2021 16:51:38
R*
```

Example output for an EmStat4 HR with firmware v1.1.00

```
tes4_hr1100#Jan 28 2022 11:04:43
R*
```

4.2. Set register (`S`)

Sets the value of a register. Registers contain instrument specific configuration, settings and information that are accessible to the user. See Chapter 6, *Register details* for more information.



Some registers require a specific permission level to be accessed. See Section 6.2, "Permission level (`0x02`)" for more details.

Command format

```
Sxxyy...yy
```

Key	Type	Size	Description
`xx`	hex	2	Register identifier (see Chapter 6, *Register details*)
`yy⋯yy`	hex	variable	Value to write to the register, the number of digits depend on the register.

Response format

```
S
```

Example

The following example demonstrates writing the value `0xABCDEF12` to register `0x99` (= 153 decimal).

Example set register command

```
S99ABCDEF12
```

Example output

```
S
```

4.3. Get register (`G`)

Gets the value of a register. Registers contain instrument specific configuration, settings and information that are accessible to the user. See Chapter 6, *Register details* for more information.



Some registers require a specific permission level to be accessed. See Section 6.2, "Permission level (`0x02`)" for more details.

Command format

```
Gxx
```

Key	Type	Size	Description
`xx`	hex	2	Register identifier (see Chapter 6, *Register details*)

Response format

```
Gyy...yy
```

Key	Type	Size	Description
`yy..yy`	hex	variable	The value of the register when queried, the number of bytes depends on the register (see Chapter 6, *Register details*).

Example

The following example demonstrates how to get the device serial (register 0x06) from the instrument.

Example get register command

```
G06
```

Example output

```
G0012000000000899B
```

4.4. Load MethodSCRIPT (`l`)

Load a MethodSCRIPT into RAM. The end of the script is indicated by an empty line (i.e., a line containing only the newline character `\n`). The MethodSCRIPT is parsed during reception. Some script errors that can be detected during parsing, such as syntax errors, are reported directly. If an error is encountered during parsing, the script memory is cleared, so a new script must be loaded. If the script was loaded successfully (no error was returned during loading), then the script can be executed by the `r` command (see Section 4.5, "Run loaded MethodSCRIPT (`r`)").

Command format

This command consists of multiple lines. The first line contains only the `l` command. Then, the MethodSCRIPT is transmitted, line by line. After the last MethodSCRIPT line, an empty line must be transmitted to end the command.

```
l
mm
..
mm
```

Key	Type	Size	Description
`mm..mm`	text	variable	The MethodSCRIPT to load, terminated with an empty line. See the MethodSCRIPT documentation for more information.

Response format

```
l
```

Example

The following example loads a MethodSCRIPT that prints "Hello World" 5 times when executed. It can then be executed with the run command, see Section 4.5, "Run loaded MethodSCRIPT (`r`)",

Example command (the newline characters are included here for clarity)

```
l\n
var i\n
store_var i 0i ja\n
loop i < 3i\n
   send_string "Hello World"\n
   add_var i 1i\n
endloop\n
\n
```

Example output (the newline characters are included here for clarity)

```
l\n
```

4.5. Run loaded MethodSCRIPT (`r`)

Run (execute) loaded MethodSCRIPT from RAM.

Command format

```
r
```

Response format

The output of this command starts with `r\n` to denote the successful start of the script. This response is then followed by the output of the MethodSCRIPT, which depends on the actual script that is running. See the MethodSCRIPT documentation to see what type of responses can be expected. Note that a MethodSCRIPT does not have to transmit data, but most scripts do. When the MethodSCRIPT is finished (either successfully or with an error), an empty line is transmitted.

Summarized, the output format is:

```
r
pp..pp
...
pp..pp
```

Key	Type	Size	Description
`pp..pp`	text	variable	The MethodSCRIPT output. See the MethodSCRIPT documentation for more information.

Example

The following demonstrates running the MethodSCRIPT loaded in the example from Section 4.5, “Run loaded MethodSCRIPT (`r`)”.

Example command (the newline characters are included here for clarity)

```
r
```

Example output (the newline characters are included here for clarity)

```
r
L
THello World
THello World
THello World
+
```



`L` and `+` are MethodSCRIPT hints about entering and leaving a loop.

4.6. Execute (= load and run) MethodSCRIPT (`e`)

Load and run a MethodSCRIPT (same as `l` followed by `r`).

Command format

```
e
mm
..
mm
```

Key	Type	Size	Description
`mm..mm`	text	variable	The MethodSCRIPT to load, terminated with an empty line. See the MethodSCRIPT documentation for more information.

Response format

```
e
pp..pp
...
pp..pp
```

Example

The following demonstrates loading and running the same MethodSCRIPT as used in the example from Section 4.5, “Run loaded MethodSCRIPT (r)”.

Example command

```
e
var i
store_var i 0i ja
loop i < 3i
   send_string "Hello World"
   add_var i 1i
endloop
```

Example output

```
e
L
THello World
THello World
THello World
+
```

4.7. Store loaded MethodSCRIPT to NVM (`Fmscr`)

Store a loaded MethodSCRIPT to non-volatile memory (NVM).

Command format

```
Fmscr
```

Response format

```
F
```

Example

The following example demonstrates loading a script with `l` and storing it into the instrument's non-volatile memory.

Example command

```
l
send_string "Hello World!"

Fmscr
```

Example output

```
l
F
```

4.8. Load MethodSCRIPT from NVM (`Lmscr`)

Load a MethodSCRIPT from non-volatile memory (NVM). After the script has been loaded successfully, it can be executed by the `r` command (see Section 4.5, "Run loaded MethodSCRIPT (`r`)").

A MethodSCRIPT can only be loaded from NVM if it was written using the same MethodSCRIPT version as the current firmware supports.

Command format

```
Lmscr
```

Response format

```
L
```

Example

This example shows how to load a script from non-volatile memory (NVM) and execute it with an `r` command. The loaded script here was loaded in the example from Section 4.8, "Load MethodSCRIPT from NVM (`Lmscr`)"

Example command

```
Lmscr
r
```

Example output

```
L
r
THello World!
```

4.9. Hibernate (*deprecated*) (`s`)

Set the device into hibernate mode. This command has the same effect as the MethodSCRIPT command `hibernate 3i 0`.

On the EmStat Pico, the hibernate mode is a very low power mode. On the EmStat4, this command is only provided for compatibility reasons but has no special purpose.

The instrument can be woken up from hibernate mode either by sending any data or by setting the `WAKE` pin. Note that in hibernate mode, and shortly before and after it, no communication is possible with the device. Any data transmitted directly following a hibernate command, as well as any characters transmitted to wake up the device, or directly after the device is woken, will be lost.



This command is deprecated. The instrument can also be put into hibernate mode using MethodSCRIPT.

Command format

```
s
```

Response format

```
s
```

4.10. Get serial number (`i`)

Get the serial number of the instrument as printed on the serial sticker.

Command format

```
i
```

Response format

```
ixx..xx
```

Key	Type	Size	Description
`xx..xx`	text	variable	The serial number as printed on its sticker.

Example

The following example queries the device serial.

Example command

```
i
```

Example output

```
iES4LR21E0399
```

4.11. Get multi-channel serial number (`m`)

Get the device serial number from a multi-channel instrument.

Some instruments, such as the MultiEmStat4, consist of multiple devices internally, each with their own communication interface. In this case, each device will return a different serial number using the `i` command. However, the `m` command will return the same serial number on each connection, which is the serial number of the combined (multi-channel) instrument. The multi-channel serial number also contains the channel number (which is different for each instrument inside the multi-instrument) and the total number of channels. This allows the host software to determine if all channels are connected.

If the instrument is not in a multi-channel configuration, this will throw an error instead.

Command format

```
m
```

Response format

```
mss..ssCHiii-nnn
```

Key	Type	Size	Description
`ss..ss`	text	variable	The multi-channel serial number.
`CH`	text	2	A fixed delimiter.
`iii`	dec	3	Channel number (1..N).
`nnn`	dec	3	Total number of channels (N).

Example

The following example queries the multi-channel serial.

Example command

```
m
```

Example output for a 12-channel MultiEmStat4 HR

```
mMES4HR2106000310CH010-012
```

Example output for an EmStat Pico or a (stand-alone) EmStat4

```
m!0048
```

4.12. Get MethodSCRIPT version (`v`)

Get the MethodSCRIPT version. This number indicates the internal storage representation of a MethodSCRIPT rather than the version of MethodSCRIPT specification. The MethodSCRIPT version number is used to determine if the MethodSCRIPT stored in NVM (using the `Fmscr` command) can be loaded or not. A list of EmStat4 firmware versions and the associated MethodSCRIPT versions is given below.

EmStat4 firmware version	MethodSCRIPT version
1.0.0	0003
1.1.0	0006
1.2.0	01.04.00

Command format

```
v
```

Response format

```
vxx..xx
```

Key	Type	Size	Description
`xx..xx`	text	variable	The MethodSCRIPT version supported by the firmware.

Example

This example demonstrates reading the MethodSCRIPT version.

Example command

```
v
```

Example output (MethodSCRIPT version = 1.4.0)

```
v01.04.00
```

4.13. Enter bootloader (`dlfw`)

Resets the instrument into bootloader mode. The bootloader is mainly intended to perform firmware updates.

Command format

```
dlfw
```

Response format

```
d
```

4.14. Get directory listing (`fs_dir`)

Get a list of all files in the specified directory including sub directories.



It might take some time to find all files on the file system.

Command format

```
fs_dir [path]
```

Key	Type	Size	Description
`[path]`	text	variable	(Optional) Path of the directory to search. If no path is provided, all files on the file system are included.

Response format

The response will consist of one line of information for each file found. The information includes the file creation date and time, type (directory or normal file), size, and path. The response ends with an empty line.

```
f
YYYY-MM-DD hh-mm-ss;TTT;SS..SS;pp..pp
...
YYYY-MM-DD hh-mm-ss;TTT;SS..SS;pp..pp
```

Key	Type	Size	Description
`YYYY`	dec	4*	File creation date†, year
`MM`	dec	2*	File creation date†, month (01-12)
`DD`	dec	2*	File creation date†, day (01-31)

Key	Type	Size	Description
`hh`	dec	2*	File creation time†, hours (00-23)
`mm`	dec	2*	File creation time†, minutes (00-59)
`ss`	dec	2*	File creation time†, seconds (00-59)
`TTT`	text	3	File type (`FIL` for file, `DIR` for directory)
`SS..SS`	dec	variable (1-10)	File size in bytes
`pp..pp`	text	variable	Path to the file/directory

* Older firmware versions may print the decimal fields without padding, e.g: `0-0-0 0-0-0;FIL;0;empty.txt`

† The file creation date and time are based on the system date and time. In order to have a meaningful file date/time, make sure to set the system date and time before creating a file.

In case a file is not closed correctly, the file size will be reported as 4294967295 bytes. This can happen if an instrument is powered down while a file was still open. In this case, a small amount of data that was not flushed to the file storage yet might be lost. However, the file should still be readable, and the correct amount of data (that has been successfully written) will be returned.

Example

The following example lists the content of the `example/doc` directory.

Example command

```
fs_dir example/doc/
```

Example output

```
f
2022-02-22 20:22:02;FIL;4;example/doc/test.txt
2022-02-22 22:22:22;FIL;14;example/doc/measurement.txt
```

Example output in case no files are found

```
f
```

4.15. Read file (`fs_get`)

Read a file from the file system on the instrument.

Command format

```
fs_get <path>
```

Key	Type	Size	Description
`<path>`	text	variable	Path of the file to retrieve.

Response format

The command `fs_get <path>\n` prints `f\n`, followed by the contents of the requested file. The end of the file is indicated by an ASCII file separator character (`0x1C`). The output ends with an empty line (i.e., a newline character) if the file was read and transmitted successfully, otherwise it ends with an error code. The file separator character is always transmitted, even in case of a file error.

```
f
cc..cc
cc..cc
cc..cc
\x1C
```

Key	Type	Size	Description
`cc..cc`	text	variable	The file content in ASCII format.
`\x1C`	-	1	The file separator character (`0x1C`).

Example

This example requests the contents of the file *example/hello_world.txt*.

Example command

```
fs_get example/hello_world.txt
```

Example output

```
f
This is an example. Hello World!
The next line contains an file separator indicating end of transfer.
\x1C
```

4.16. Write file (`fs_put`)

Write a file to the file system of the instrument.

The file path must be unique. If a file with the same path already exists, an error is returned.

Command format

The command starts with `fs_put <path>\n`, where `path` is the path of the file to write. The following lines are the file contents, that are written to the file. The end of the file is indicated by an ASCII file separator character (`0x1C`).

```
fs_put <path>
xx..xx
\x1C
```

Key	Type	Size	Description
`<path>`	text	variable	The file path.
`xx..xx`	text	variable	The file content in ASCII format.
`\x1C`	-	1	The file separator character (`0x1C`).

Response format

The command returns a `\n` when it is accepted, as all commands do. It also returns an additional empty line (`\n`) when the command is finished.

```
f
```

Example

Example command

```
fs_put example/hello_world.txt
This is an example. Hello World!
The next line contains a file separator indicating end of transfer.
\x1C
```

Example output

```
f
```

4.17. Delete file or directory (`fs_del`)

Remove a file or directory (recursively) from the file system.

Command format

```
fs_del <path>
```

Key	Type	Size	Description
`<path>`	text	variable	Path of the file or directory to remove.

Response format

```
f
```

Example

The following example removes the file `/log.txt`.

Example command

```
fs_del /log.txt
```

Example output

```
f
```

4.18. Get file system information (`fs_info`)

Get information about the file system (free/used/total space).

The file system information consists of free space, used space and total space.



Due to file system overhead, the total space will be less than the nominal capacity of the storage medium. The exact amount of overhead depends of the type and size of storage medium.

Command format

Example command

```
fs_info
```

Response format

```
f
```

```
used:UU..UUkB free:FF..FFkB total:TT..TTkB
```

Key	Type	Size	Description
`UU..UU`	dec	variable	Used size in kB*
`FF..FF`	dec	variable	Free size in kB*
`TT..TT`	dec	variable	Total size in kB*

* 1 kB = 1024 bytes

Example

Example command

```
fs_info
```

Example response

```
used:192kB free:7878464kB total:7878656kB
```

4.19. Format storage device (`fs_format`)

Format the file storage medium. This prepares the storage medium to be used as file system. It also removes all existing data.

Formatting a (large) storage device can take some time.

Once the storage device is formatted, it is generally not necessary to use this command again. To only remove all files, it is recommended to use the `fs_clear` command instead.



Formatting the file storage erases all files. This operation cannot be undone.

Command format

```
fs_format
```

Response format

```
f
```

4.20. Mount file system (`fs_mount`)

Mount the file system.

Command format

```
fs_mount
```

Response format

```
f
```

4.21. Unmount file system (`fs_unmount`)

Unmount the file system. This can be used to re-mount the filesystem, in combination with `fs_mount`.

Command format

```
fs_unmount
```

Response format

```
f
```

4.22. Clear file system (`fs_clear`)

Remove all files and folders from the storage medium.



This operation cannot be undone.

Command format

```
fs_clear
```

Response format

```
f
```

4.23. Get runtime capabilities (`CC`)

Get the runtime capabilities. Return a list of supported commands for the instrument. Each bit represent one command, the mapping between bits and commands can be found in Appendix C, *Communication capabilities bit fields*.

Command format

```
CC
```

Response format

```
CXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXXX
```

Key	Type	Size	Description
`XX..XX`	hex	32	Bit fields for commands

Example

Example command

```
CC
```

Example response

```
C0000000000000000000000000000000000000F000000001FFFD8FF0000000E
```

4.24. Get MethodSCRIPT capabilities (`CM`)

Get the MethodSCRIPT capabilities. Return a list of supported MethodSCRIPT commands for the instrument as hexadecimal value. Each bit represent one command, the mapping between bits and commands can be found in Appendix B, *MethodSCRIPT capabilities bit fields*

Command format

```
CM
```

Response format

```
CYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

Key	Type	Size	Description
`YY..YY`	hex	32	Bit fields for MethodSCRIPT commands

Example

Example command

```
CM
```

Example response

```
C0000000000000000000000000000000000000000002BFFF87FFFFFFFFFFBFFFFE
```

4.25. Halt script execution (`h`)

Halt execution of the running MethodSCRIPT.

This "pauses" the script. Execution can be resumed using the `H` command (see Section 4.26, “Resume script execution (`H`)”).



The halt command will normally be handled at the end of every MethodSCRIPT command. However, while inside a loop, it is handled only when the `endloop` tag is reached.

Command format

```
h
```

Response format

```
h
```

Example

See the examples in Section 4.28.

4.26. Resume script execution (`H`)

Resume execution of the halted MethodSCRIPT.

Command format

```
H
```

Response format

```
H
```

Example

See the examples in Section 4.28.

4.27. Abort script execution (`Z`)

Abort execution of the current MethodSCRIPT. This has the same effect as the MethodSCRIPT command `abort`. It effectively stops the execution of the script as soon as possible. If an abort occurs during a (measurement) loop, all `endloop` commands are still executed. Consequently, the `*` and `+` characters that denote the end of a loop will still be transmitted. If the MethodSCRIPT contains an `on_finished:` tag, the commands after it will still be executed. MethodSCRIPT commands after the `on_finished:` tag cannot be aborted.

Unlike the MethodSCRIPT command `abort`, the command can also abort some long-running MethodSCRIPT commands, such as `await_int` and certain measurements.

Command format

```
Z
```

Response format

```
Z
```

Example

See the examples in Section 4.28.

4.28. Abort measurement loop (`Y`)

Abort the current measurement loop. This will break the execution of a MethodSCRIPT measurement loop command (i.e., a command starting with `meas_loop_`) after the current iteration. The current measurement iteration, i.e., all MethodSCRIPT commands between the start and the end of the measurement loop, will be executed, but no new iteration will be started. The script will then continue execution after the `endloop` command.

Command format

```
Y
```

Response format

```
Y
```

Example

Below is an example MethodSCRIPT that performs a linear sweep from -1 V to +1 V, with steps of 250 mV and a scan rate of 100 mV/s. This results in 9 measurements, each 2.5 second apart, with a total runtime of approximately 22.5 seconds. In our example setup, a 100 kΩ resistor was connected to the working electrode, so the measured current is expected to be between -10 µA and +10 µA, and the current range is set accordingly.

```
var c
var p
var i
var t
store_var i 0i ja
set_pgstat_mode 2
set_range ba 10u
cell_on
timer_start
meas_loop_lsv p c -1 1 250m 100m
  add_var i 1i
  pck_start
  pck_add i
  pck_add p
  pck_add c
  pck_end
endloop
timer_get t
meas 100m c ba
pck_start
pck_add t
pck_add c
pck_end
on_finished:
cell_off
send_string "Finished"
```

When the program is executed completely, the output will be something like this:

```
e
M0000
Pja8000001i;da7F0BDF9u;ba7678CD7p,10,20F,40
Pja8000002i;da7F48ED6u;ba78DBCE5p,10,20F,40
Pja8000003i;da7F85FB4u;ba7B3E948p,10,20F,40
Pja8000004i;da7FC3092u;ba7DA1200p,10,20F,40
Pja8000005i;da8059967n;ba8D7055Ef,14,20F,40
```

```
Pja8000006i;da803D24Cu;ba8265C17p,10,20F,40
Pja8000007i;da807A32Au;ba84C8C26p,10,20F,40
Pja8000008i;da80B7408u;ba872B4DDp,10,20F,40
Pja8000009i;da80F44E5u;ba898E141p,10,20F,40
*
Peb9570C36u;ba898E141p,10,20F,40
TFinished
```

The values in the data packages indicate that the measurement loop took approximately 22.5 seconds, and that the measured current after the measurement loop has the same value as during the last iteration of the loop.

However, if a `Y` command is send after the second iteration, the output will be something like this:

```
e
M0000
Pja8000001i;da7F0BDF9u;ba7679082p,10,20F,40
Pja8000002i;da7F48ED6u;ba78DB93Ap,10,20F,40
Y
Pja8000003i;da7F85FB4u;ba7B3E1F1p,10,20F,40
*
Peb872184Au;ba7D9E9A2p,10,20F,41
TFinished
```

…or, depending on the exact time the `Y` command is received, like this:

```
e
M0000
Pja8000001i;da7F0BDF9u;ba767942Ep,10,20F,40
Pja8000002i;da7F48ED6u;ba78DC43Cp,10,20F,40
Y
*
Peb84D7686u;ba7B3E948p,10,20F,40
TFinished
```

In this case, the values indicate that the measurement loop only took 5 seconds, and that the WE potential remained at the value it had at the end of the last iteration that was executed.

By halting the program after the second iteration, the output would be:

```
e
M0000
Pja8000001i;da7F0BDF9u;ba767942Ep,10,20F,40
Pja8000002i;da7F48ED6u;ba78DB93Ap,10,20F,40
h
```

If the program would now be continued and then aborted after three more iterations, the output would be:

```
H
Pja8000003i;da7F85FB4u;ba7B3E59Dp,11,20F,40
Pja8000004i;da7FC3092u;ba7DA0E54p,10,20F,40
Pja8000005i;da8059967n;ba8C8AFADf,14,20F,40
Z
*
TFinished
```

As can be seen in the above example, the metadata of the 3th iteration (the value `11`) indicates that a timing error occurred. It can also be seen that the code directly following the measurement loop is not executed when the script is aborted using the `Z` command, in contrast to the `Y` command, which only aborts the measurement loop but continues executing the remainder of the MethodSCRIPT.

Chapter 5. Register summary

5.1. Generic registers

The following table defines registers that are the same on all MethodSCRIPT instruments.

ID	Description	Length (bytes)	Basic permission	Advanced permission
0x01	Peripheral configuration	4	Read only	Read / write
0x02	Permission level	4	Read / write	Read / write
0x04	License register	8	Read only	Read only
0x05	Unique instrument ID	16	Read only	Read only
0x06	Device serial number	8	Read only	Read only
0x08	MethodSCRIPT autorun	1	Read only	Read / write
0x09	Advanced options	4	Read only	Read / write
0x0A	UART data rate limit	4	Read / write	Read / write
0x0B	Reset instrument	4	Write only	Write only
0x0D	Multi-channel role	1	Read only	Read only
0x0E	System date and time	7	Read / write	Read / write
0x0F	Default GPIO config	8	Read only	Read / write
0x10	System warning	4	Read only	Read only

5.2. EmStat4 specific registers

The table below lists all registers that are specific to the EmStat4.

ID	Description	Length (bytes)	Basic permission	Advanced permission
0x81	NVM commit	4	None	Write only
0x87	Multi-channel serial	8	Read only	Read only
0x88	AUX DAC gain	2	Read only	Read / write
0x89	Baud rate configuration	1	Read only	Read / write

Chapter 6. Register details

The internal registers are used to retrieve information, configure the device, or perform rarely used actions.

Some registers are write protected at startup and must be unlocked before use. The table in Section 6.2, “Permission level (0x02)” shows which access rights each register has depending for each permission level.

The data length of each register is given in bytes of represented data. This data is communicated in hexadecimal notation, using 2 characters per byte.

Some registers are stored in the non-volatile memory (NVM) of the instrument, meaning that the setting can be remembered even after a power cycle. On the EmStat4, writing to those register will immediately take effect, but the changes are only updated in volatile memory without actually updating the NVM. The NVM can be updated using the NVM commit register. This will write all changed values to NVM to make them permanent (until they are overwritten again using the same command).

6.1. Peripheral configuration (0x01)

Reads / writes the peripheral configuration as a bitmask from / to non-volatile memory. Support for external peripherals can be enabled here. Pins for peripherals that are not enabled can be used as GPIO pins. All peripherals default to GPIO. Multiple peripherals can be enabled at the same time by adding the hexadecimal values. For example: bit 1 is 0x01 and bit 5 is 0x20, combining them gives 0x21.



This setting is stored in NVM.

Register format

```
xxxxxxxx
```

Key	Size (bytes)	Description
xxxxxxxx	4	Peripheral configuration flags.

Table 2. EmStat4 peripheral configuration

Mask	Name	Description
0x0001	Disable power LED	Disable the (blue) power LED.
0x0040	Use external cell LED	When enabled, output the cell ON/OFF status on GPIO2 instead of the internal cell LED. The signal is active-high: Cell ON outputs a logic 1, cell OFF output a logic 0. GPIO2 can not be used for other purposes if this option is enabled.
Other	Reserved	Reserved for future use. Do not change!

6.2. Permission level (0x02)

By default, most registers are write protected to prevent accidental writes. This register can be used to disable the write protection. It is advised to turn the write protection back on when access to write protected registers is

no longer required.

Register format

```
kkkkkkkk
```

Key	Size (bytes)	Description
`kkkkkkkk`	4	Key to for switching to a specific permission mode. See Table 3, "Permission keys".

Table 3. Permission keys

Level	Key	Description
Basic	0x12345678	Default configuration at startup. Read-only access to non-volatile registers.
Advanced	0x52243DF8	Full access to all user changeable settings.

6.3. License register (`0x04`)

Request the licenses programmed into this instrument. For more information contact PalmSens.

Register format

```
xxxxxxxxxxxxxxxx
```

Key	Size (bytes)	Description
`xx..xx`	8	Instrument specific license key.

Example

Command to read the license register.

```
G04
```

6.4. Unique instrument ID (`0x05`)

Reads the unique ID for this instrument.



This is different than the device serial number.

Register format

```
xxxxxxxxxxxxxxxxxxxxxxxxxxxxxxxx
```

Key	Size (bytes)	Description
`xx..xx`	16	Unique hardware identifier.

Example

Command to read the instrument ID.

```
G05
```

6.5. Device serial number (`0x06`)

Contains the device serial number.

Register format

```
ttyybbbbnnnnnnnn
```

Key	Size (bytes)	Description
`tt`	1	A number specifying the device type.
`yy`	1	Production year.
`bbbb`	2	Production batch nr.
`nnnnnnnn`	4	Device ID, unique within all devices of the same type, year and batch.

Example

Command to read the serial number of the device.

```
G06
```

Example output

```
G001200000000899B
```

6.6. MethodSCRIPT autorun (`0x08`)

If set to 1, the MethodSCRIPT stored in non-volatile memory will be loaded and executed on startup. When the script ends, the EmStat4 returns to its normal behavior.



This setting is stored in NVM.

Register format

```
aa
```

Key	Size (bytes)	Description
`aa`	1	Autorun enable (00=disabled, 01=enabled).

Example

Command to read the autorun option.

```
G08
```

Command to enable the autorun option.

```
S0801
```

6.7. Advanced options (`0x09`)

The advanced options register is a bitmask of advanced options that can be enabled by the user.

Each option has a specific bit value (see table below). The value of this register is a *bitwise OR* of all option flags that are enabled. Writing to this register sets or clears all bits to the specified value. When writing to this register, make sure to set all required bits at once.



This setting is stored in NVM.

Table 4. Advanced option bits

Bit mask	Description
`0x80000000`	Enable CRC16 protocol extension.

Register format

```
aaaaaaaa
```

Key	Size (bytes)	Description
`aa..aa`	4	Advanced options

Example

Command to read the advanced options register.

```
G09
```

Command to clear the advanced options register.

```
S0900000000
```



If the CRC16 protocol extension is (accidentally) enabled, it can only be disabled using a command including valid CRC. In this case, the command `S0900000000AA9D43` can be used to clear the advanced options register, including the CRC16 protocol extension.

6.8. UART data rate limit (`0x0A`)

This register allows limiting the number of bytes per second that are sent by the device using UART. This is independent of the UART baud rate. This can be useful when no flow control mechanism is used with UART and the host cannot keep up with the data rate defined by the baud rate. A value of 0 disables data rate limiting, so the instrument will transmit at the maximum achievable speed.

Register format

```
dddddddd
```

Key	Size (bytes)	Description
`dd..dd`	4	Data rate limit in bytes per second

Example

Command to read the UART data rate limit.

```
G0A
```

Command to set the UART data rate limit to 5000 (=0x1388) bytes/sec.

```
S0A00001388
```

6.9. Reset instrument (`0x0B`)

Writing 0x93628ADE to this register will initiate a software reset of the device.



This command will not return a newline if the reset is successful.

Register format

```
93628ADE
```

Key	Size (bytes)	Description
`93628ADE`	4	Magic key to reset the instrument.

Example

Command to reset the instrument.

```
S0B93628ADE
```

6.10. Multi-channel role (`0x0D`)

Instrument role in a multi-channel setup.

When combining multiple instruments to create a multi-channel setup (as in, for example, the MultiEmStat4), it is sometimes necessary to synchronize all channels, so all measurements are performed at the same time. This can be achieved using the MethodSCRIPT command `set_channel_sync`. When using this feature, one instrument must be configured as master, and all others as slave. The multi-channel role determines how the instrument behaves when synchronization commands are used.

This role is assigned from the factory and depends on the physical layout.

Options are:

Value	Description
0x00	Stand-alone, no multi-instrument
0x4D	Master
0x53	Slave
0xFF	Standalone, no multi-instrument

Register format

```
mm
```

Key	Size (bytes)	Description
`mm`	1	Multi-instrument role

6.11. System date and time (`0x0E`)

The system date and time in hex format. This is used for the time/date shown on files in the filesystem. Depending in the instrument, the time may or may not be kept on a restart.

Register format

```
yyyymmddhhaass
```

Key	Size (bytes)	Description
`yyyy`	2	Year, in HEX format
`mm`	1	Month (1-12), in HEX format
`dd`	1	Day (1-31), in HEX format
`hh`	1	Hour (0-23), in HEX format
`aa`	1	Minute (0-59), in HEX format
`ss`	1	Second (0-59), in HEX format

6.12. Default GPIO config (`0x0F`)

Default GPIO settings at startup. Once set (and committed to NVM) the instrument will initialize it's GPIO to this state on startup.



This setting is stored in NVM.



The GPIO config is not checked when this register is updated. An incorrect configuration can cause a system warning during startup.

Register format

```
ccccccccoooopppp
```

Key	Size (bytes)	Description
`cc..cc`	4	GPIO configuration mode with 2 bits per pin. The values are: 0 = Mode 0 (Input) 1 = Mode 1 (Output) 2 = Mode 2 (alternate function 1) 3 = Mode 3 (alternate function 2)
`oooo`	2	Output level with 1 bit per pin (only applicable for output pins)
`pppp`	2	Pull-up enabled with 1 bit per pin (only applicable for input pins)

Example

Command to configure pin 0 as input pin with pull-up and pin 1 as output pin with output value 1.

```
S0F0000000400020001
```

6.13. System warning (`0x10`)

Read and clear the system warning.

If a problem occurred that can not be displayed or handled at that moment, a system warning is set. This is indicated with the blinking LED and available in this register. Reading this register will return the first error code that caused a system warning. This is usually the most meaningful error code, since any subsequent errors might be a consequence of the first error. This register is cleared when read.

Register format

```
wwwwwwww
```

Key	Size (bytes)	Description
`ww..ww`	4	Last encountered error code For a list of error codes, see Appendix A, *Error codes*.

Example

Command to read and clear the system warning.

```
G10
```

6.14. NVM commit (`0x81`)

Commit the current settings to non-volatile-memory to keep the settings across power cycles and resets. This requires a magic key to be provided

Register format

```
1234ABCD
```

Key	Size (bytes)	Description
`1234ABCD`	4	Magic key to commit.

Example

Command to commit the NVM settings.

```
S811234ABCD
```

6.15. Multi-channel serial (`0x87`)

Reads the multi-channel serial number for MultiEmStat configurations.

Register format

```
RRSSSSSSSSSSCCNN
```

Key	Size (bytes)	Description
`RR`	1	Reserved (`00`)
`SS..SS`	5	Serial
`CC`	1	Channel number
`NN`	1	Total number of channels

6.16. AUX DAC gain (`0x88`)

Gain for the auxiliary DAC. This can be useful to compensate for external circuitry. The register value gets divided by 1000 internally to make the actual gain.

Register format

```
gggg
```

Key	Size (bytes)	Description
`gggg`	2	Gain * 1000

6.17. Baud rate configuration (`0x89`)

Get or set the instrument's UART baud rate. This register expects an index, which is specified for each baud rate in the table below. The default baud rate can be found in the instrument's "description document".

Index	Baud rate
0	Default baud rate (see Chapter 2, *Communication*)
1	9600

Index	Baud rate
2	19200
3	38400
4	57600
5	115200
6	230400
7	460800
8	921600



Make sure to note which baud rate is set, because you can only connect to the device using the configured baud rate.

Register format

```
BB
```

Key	Size (bytes)	Description
`BB`	1	Baud rate index

Example

Command to set the baud rate to 230400 bits per second.

```
S8906
```

Chapter 7. CRC16 protocol extension

7.1. Introduction

For certain applications of the EmStat4, data validity is of critical importance. For such applications, all data communication from and to the instrument has to be verifiable. In order to make the communication verifiable, an extension of the protocol was implemented that adds a sequence number and a 16-bit CRC to each line. The CRC makes it possible to verify if received data is correct, i.e., if no part of the line was corrupted or lost during transmission. The sequence number allows the host to verify that no complete lines were missed.

The CRC16 protocol extension can be enabled in the instruments non-volatile configuration by setting the corresponding option bit (by issuing the command `S0980000000` in normal mode). See the Set register command and the Advanced options register for more details on how to enable this extension.

Enabling the CRC16 protocol has the following effects:

- All lines transmitted by the EmStat4 include a sequence number and CRC.
- All lines transmitted by the host software must include a sequence number and CRC.
- For each line correctly received by the EmStat4, an acknowledge message is transmitted.
- In case the received sequence number is different then expected, an error message (`!002C`) is transmitted. This can happen if a line is lost, but can also happen at the start of the communication, for example if the host application has been restarted. A sequence number error is treated as a warning and is not considered an error by the EmStat4. The received line will still be acknowledged and processed.
- For each corrupted line received by the EmStat4, an error message (`!002B` or `!002D`) is transmitted. In this case, the message is not processed by the firmware.
- Some commands have a slightly different response.

The following section describes the protocol extension details.

7.2. Line format

The CRC extension adds an 8-bit sequence number and 16-bit CRC to each line before the newline separator (`\n`). This applies to all data transmitted to and from the device.

Line format when CRC16 protocol extension is enabled

```
nnnnnnnnSSCCCC\n
```

Key	Type	Size	Description
`nnnnnnnn`	text	variable	The normal line that would be transmitted if the CRC16 protocol extension was disabled.
`SS`	hex	1 byte	The sequence number (0-255).
`CCCC`	hex	2 bytes	The 16-bit CRC, calculated over `nnnnnnnnSS`.

The sequence number allows the receiver to detect if there are missing lines. There are separate, independent, sequence numbers for data in both directions (from and to instrument). At startup, the EmStat4 initializes its

sequence number to 0 and also expects the host to start with sequence number 0. After every transmitted line, the corresponding sequence number is incremented with one. After sequence number 255, it rolls over to number 0.

The CRC allows the receiver to verify the integrity of the received data. The CRC is calculated over the full line, excluding the newline character, but including the sequence number. The used CRC is the CRC-CCITT polynomial $x^{16} + x^{12} + x^5 + 1$, often represented as 0x1021. The initial value is 0xFFFF.



When using Python, the standard libary function `binascii.crc_hqx()` can be used to calculate the CRC.

7.3. Acknowledge messages

To give the host more certainty that the data is actually received by the EmStat4, the instrument will acknowledge every received line with an acknowledge message. The acknowledge message simply contains the sequence number of the received line, between angle brackets, e.g. `<00>`. The message itself also contains a sequence number and CRC like any other message transmitted by the instrument. The acknowledge messages are only transmitted by the instrument and should not be transmitted by the host.

Acknowledge message format

```
<AA>SSCCCC\n
```

Key	Type	Size	Description
`AA`	hex	1 byte	The sequence number (0-255) of the received line.
`SS`	hex	1 byte	The sequence number (0-255) of the instrument.
`CCCC`	hex	2 bytes	The 16-bit CRC, calculated over `nnnnnnnnSS`.

7.4. Other changes

The EmStat4 will respond mostly in the same way as it does without the CRC16 protocol extension. An exception is with MethodSCRIPT related commands (`e` and `l`). These will normally return with just a letter without newline and a send the newline when the entire script is received. Since this would interfere with the acknowledge messages it was decided that when the CRC16 protocol extension is enabled it will add an additional newline directly after the command response letter.

7.5. Examples

Below are some examples to demonstrate the differences between communication with and without the CRC16 protocol extension.

Example command without CRC16 protocol extension

Host to instrument	Instrument to host
`t\n`	
	`tes4_lr1000#Jun 7 2021 16:51:38\n`
	`R*\n`

Example command with CRC16 protocol extension enabled

Host to instrument	Instrument to host
`t0A9524\n`	
	`<0A>454FBA\n`
	`tes4_lr1000#Jun 7 2021 16:51:38463321\n`
	`R*47D271\n`

Note: `\n` is the newline character, initial sequence IDs are `0x0A` for the host and `0x45` for the instrument.

MethodSCRIPT example without CRC16 protocol extension (Note that there's no `\n` after the `e` response from the instrument!)

Host to instrument	Instrument to host
`e\n`	
	`e`
`send_string "Hello World"\n`	
`\n`	
	`\n`
	`THello World\n`
	`\n`

MethodSCRIPT example with CRC16 protocol extension enabled

Host to instrument	Instrument to host
`e03BFA2\n`	
	`<03>4CFEF6\n`
	`e4D7D16\n`
`send_string "Hello World"04A94C\n`	
	`<04>4ECF1D\n`
`057E6C\n`	
	`<05>4F89CA\n`
	`50D13C\n`
	`THello World5142CE\n`
	`52F17E\n`

Chapter 8. Error handling

After sending a command to the device, the device may respond with an error code. This may occur if a command or parameter is not supported by the connected instrument or otherwise outside of its capabilities.

The general error format is an exclamation mark (`!`) followed by a 4-digit (hexadecimal) error code. However, when an error is encountered during reception (loading) of a MethodSCRIPT, the error response also contains the line and column number. When an error is encountered during *execution* of a MethodSCRIPT, the error response only contains the line number. Because a newline character has already been transmitted at the start of the script execution, the exclamation mark will be on the start of the line (not prepended by the `e`) in this case.

General error format of the device communication protocol

```
c!XXXX\n
```

Error format during MethodSCRIPT parsing (loading)

```
l!XXXX: Line LL, Col CC\n
```

Error format during MethodSCRIPT execution

```
!XXXX: Line LL\n
```

Key	Type	Size	Description
`c`	text	1	The first letter of the received command.
`XXXX`	hex	4	The error code (see Appendix A, *Error codes*).
`LL`	dec	variable	The line number of the MethodSCRIPT on which the error occurred.
`CC`	dec	variable	The column number (character position within the line) on which the error occurred.

For a full list of error codes, see Appendix A, *Error codes*



Error codes can be different on different instruments and firmware versions.

After an error occurred, the instrument will ignore further input for a short time (roughly 50-100 ms). It is recommended to wait for more than 100 ms before transmitting the next command, to make sure it will be received and processed normally.

Examples

Example of wrong communication protocol command

Host to instrument	Instrument to host
`wrong_command\n`	`w` `!0003\n`

Example of wrong MethodSCRIPT command (parsing error)

Host to instrument	Instrument to host

`e\n`
`wrong_methodscript_command\n`
	`e`
	`!4001: Line 1, Col 27\n`
	`\n`

Example of MethodSCRIPT runtime error (division by zero)

Host to instrument	Instrument to host

`e\n`
`var x\n`
`store_var x 0i ja\n`
`send_string "1"\n`
`div_var x 0i\n`
`send_string "2"\n`
`\n`
	`e`
	`\n`
	`T1\n`
	`!0028: Line 4\n`
	`\n`

Chapter 9. Version changes

Version 1.0

- Initial version.

Version 1.2

- Updated communication details (UART baudrate and flow control)
- Clarified error handling
- Added CRC16 protocol extension
- Added commands `CC`, `CM`, `s`, and `m`
- Added file system commands (`fs_*`)
- Added/updated registers
- Added register permissions
- Updated error codess
- Updated document format

Appendix A: Error codes

The following table lists all error codes that can be returned by MethodSCRIPT instruments.

Note that some of these error codes are part of the communication protocol (e.g. 0004–0006, 0008, 0009), while others are only returned during MethodSCRIPT loading (e.g. 0003, 4001) or MethodSCRIPT execution (e.g. 0028, 400F). Some more generic error codes (e.g. 0001) are applicable for both. In this table, no distinguishment is made between the source of the error codes. Instead, all codes are included and sorted by number so they can be quickly referenced when troubleshooting.



Error codes can be different on different instruments and firmware versions.

Table 5. Error code lookup table

Error code	Description
0x0001	An unspecified error has occurred
0x0002	An invalid *VarType* has been used
0x0003	The command was not recognized
0x0004	Unknown register
0x0005	Register is read-only
0x0006	Communication mode invalid
0x0007	An argument has an unexpected value
0x0008	Command exceeds maximum length
0x0009	The command has timed out
0x000B	Cannot reserve the memory needed for this var
0x000C	Cannot run a script without loading one first
0x000E	An overflow has occurred while averaging a measured value
0x000F	The given potential is not valid
0x0010	A variable has become either "NaN" or "inf"
0x0011	The input frequency is invalid
0x0012	The input amplitude is invalid
0x0014	Cannot perform OCP measurement when cell on
0x0015	CRC invalid
0x0016	An error has occurred while reading / writing flash
0x0017	The specified flash address is not valid for this device
0x0018	The device settings have been corrupted
0x0019	Authentication error
0x001A	Calibration invalid

Error code	Description
0x001B	This command or part of this command is not supported by the current device
0x001C	Step Potential cannot be negative for this technique
0x001D	Pulse Potential cannot be negative for this technique
0x001E	Amplitude cannot be negative for this technique
0x001F	Product is not licensed for this technique
0x0020	Cannot have more than one high speed and/or max range mode enabled (EmStat Pico)
0x0021	The specified PGStat mode is not supported
0x0022	Channel set to be used as Poly WE is not configured as Poly WE
0x0023	Command is invalid for the selected PGStat mode
0x0024	The maximum number of vars to measure has been exceeded
0x0025	The specified PAD mode is unknown
0x0026	An error has occurred during a file operation
0x0027	Cannot open file, a file with this name already exists
0x0028	Variable divided by zero
0x0029	GPIO pin mode is not known by the device
0x002A	GPIO configuration is incompatible with the selected operation
0x002B	CRC of received line was incorrect (CRC16-ext)
0x002C	ID of received line was not the expected value (CRC16-ext)
0x002D	Received line was too short to extract a header (CRC16-ext)
0x002E	Settings are not initialized
0x002F	Channel is not available for this device
0x0030	Calibration process has failed
0x0032	Critical cell overload, aborting measurement to prevent damage.
0x0033	FLASH ECC error has occurred
0x0034	Flash program operation failed
0x0035	Flash Erase operation failed
0x0036	Flash page/block is locked
0x0037	Flash write operation on protected memory
0x0038	Flash is busy executing last command.
0x0039	Operation failed because block was marked as bad
0x003A	The specified address is not valid

Error code	Description
0x003B	An error has occurred while attempting to mount the filesystem
0x003C	An error has occurred while attempting to format the filesystem memory
0x003D	A timeout has occurred during SPI communication
0x003E	A timeout has occurred somewhere
0x003F	The calibrations registers are locked, write actions not allowed.
0x0040	Memory module not supported.
0x0041	Flash memory format not recognized or supported.
0x0042	This register is locked for current permission level.
0x0043	Register is write-only
0x0044	Command requires additional initialization
0x0045	Configuration not valid for this command
0x0046	No mux was found during auto-detect.
0x0047	The filesystem has to be mounted to complete this action.
0x0048	This device is not a multi-device, no serial available.
0x0049	GPIO configuration is incompatible with the ES4X IO AUX port
0x004A	MCU register access is not allowed, only RAM and peripherals are accessible.
0x004B	Runtime (comm) command argument too short to be valid.
0x004C	Runtime (comm) command argument has an invalid format.
0x004E	Hibernate wake up source is invalid
0x004F	Hibernate requires at least one wake up source, none was given.
0x0050	Wake pin for hibernate not configured as `input`
0x0051	The code provided to the permission register was not valid.
0x0052	An overrun error occurred on a communication interface (e.g. UART).
0x0053	Argument length incorrect for this register.
0x0055	The GPIO pins requested to change do not exist on this instrument
0x0056	The GPIO pin is reserved for a special purpose (by NVM config or device type)
0x0057	The on-board flash module has timed out.
0x0200	COMM argument value cannot be negative for this command
0x0201	COMM argument value cannot be positive for this command
0x0202	COMM argument value cannot be zero for this command
0x0203	COMM argument value must be negative for this command (also not zero)

Error code	Description
0x0204	COMM argument value must be positive for this command (also not zero)
0x0205	COMM argument value is outside the allowed bounds for this command
0x0206	COMM argument value cannot be used for this specific instrument
0x0207	An unexpected additional COMM argument was provided
0x4001	The script command is unknown
0x4004	An unexpected character was encountered
0x4005	The script is too large for the internal script memory
0x4008	This optional argument is not valid for this command
0x4009	The stored script is generated for an older firmware version and cannot be run
0x400B	Measurement loops cannot be placed inside other measurement loops
0x400C	Command not supported in current situation
0x400D	Scope depth too large
0x400E	The command had an invalid effect on scope depth
0x400F	Array index out of bounds
0x4010	I2C interface was not initialized with i2c_config command
0x4011	This is an error, NAck flag not handled by script
0x4012	Something unexpected went wrong.
0x4013	I2C clock frequency not supported by hardware
0x4014	Non integer SI vars cannot be parsed from hex or binary representation
0x4016	RTC was selected as wake-up source and selected time is not supported
0x4018	The script has ended unexpectedly.
0x4019	The script command is only valid for a multichannel (combined) device
0x401A	Fast_cv cannot be called from within a measurement loop.
0x401B	the pck sequence is called wrong
0x401C	The maximum amounts of variables per packet has been exceeded.
0x4020	A timeout has occurred for one of the script commands
0x4021	The mux is not initialized/configured.
0x4022	Measurement loop timing is too fast to use with multiplexer
0x4023	The script command is only valid for a device with IR compensation
0x4024	The resistance value is to big for the whole autorange range
0x4025	The resistance value is to big for current current range

Error code	Description
0x4026	The variable already exists when declared
0x4027	This command requires the cell to be enabled with the `cell_on` command
0x4028	This command requires the cell to be disabled with the `cell_off` comma
0x4029	The technique requires that at least one step should be made
0x4200	MScript argument value cannot be negative for this command
0x4201	MScript argument value cannot be positive for this command
0x4202	MScript argument value cannot be zero for this command
0x4203	MScript argument value must be negative for this command (also not zero)
0x4204	MScript argument value must be positive for this command (also not zero)
0x4205	MScript argument value is outside the allowed bounds for this command
0x4206	MScript argument value cannot be used for this specific instrument
0x4207	MScript argument datatype (float/int) is invalid for this command
0x4208	MScript argument reference was invalid (not 'a' - 'z')
0x4209	MScript argument variable type is not supported for this command
0x420A	An unexpected, additional, (optional) MScript argument was provided
0x420B	MScript argument variable is not declared
0x420C	MScript argument is of type var, which is not supported by this command
0x420D	MScript argument is of type literal, which is not supported by this command
0x420E	MScript argument is of type array, which is not supported by this command
0x420F	MScript argument array size is insufficient
0x7FFF	A fatal error has occurred, the device must be reset

Appendix B: MethodSCRIPT capabilities bit fields

The following table lists all MethodSCRIPT commands and their respective bit field in the Section 4.24, "Get MethodSCRIPT capabilities (`CM`)"

Table 6. MethodSCRIPT capabilities lookup table

Bit number	Command string
0	*RESERVED*
1	`var`
2	`array`
3	`store_var`
4	`copy_var`
5	`add_var`
6	`sub_var`
7	`mul_var`
8	`div_var`
9	`set_e`
10	`set_int`
11	`await_int`
12	`wait`
13	`loop`
14	`endloop`
15	`breakloop`
16	`if`
17	`else`
18	`elseif`
19	`endif`
20	`get_time`
21	`meas`
22	*RESERVED*
23	`meas_loop_lsv`
24	`meas_loop_cv`
25	`meas_loop_dpv`
26	`meas_loop_swv`

Bit number	Command string
27	`meas_loop_npv`
28	`meas_loop_ca`
29	`meas_loop_pad`
30	`meas_loop_ocp`
31	`meas_loop_eis`
32	`set_autoranging`
33	`pck_start`
34	`pck_add`
35	`pck_end`
36	`set_max_bandwidth`
37	`set_cr`
38	`cell_on`
39	`cell_off`
40	`set_pgstat_mode`
41	`send_string`
42	`set_pgstat_chan`
43	`set_gpio_cfg`
44	`set_gpio_pullup`
45	`set_gpio`
46	`get_gpio`
47	`set_pot_range`
48	*RESERVED*
49	`set_poly_we_mode`
50	`file_open`
51	`file_close`
52	`set_script_output`
53	`array_get`
54	`array_set`
55	`i2c_config`
56	`i2c_read_byte`
57	`i2c_write_byte`

Bit number	Command string
58	`i2c_read`
59	`i2c_write`
60	`i2c_write_read`
61	`hibernate`
62	`abort`
63	`timer_start`
64	`timer_get`
65	`set_range`
66	`set_range_minmax`
67	`meas_loop_cp`
68	`set_i`
69	`meas_loop_lsp`
70	`meas_loop_geis`
71	`int_to_float`
72	`float_to_int`
73	`bit_and_var`
74	`bit_or_var`
75	`bit_xor_var`
76	`bit_lsl_var`
77	`bit_lsr_var`
78	`bit_inv_var`
79	`set_channel_sync`
80	`set_acquisition_frac`
81	*RESERVED*
82	*RESERVED*
83	*RESERVED*
84	`set_gpio_msk`
85	`get_gpio_msk`
86	`set_e_aux`
87	*RESERVED*
88	*RESERVED*

Bit number	Command string
89	*RESERVED*
90	`meas_fast_cv`
91	`set_acquisition_frac_autoadjust`

Appendix C: Communication capabilities bit fields

The following table lists all MethodSCRIPT commands and their respective bit field in the Section 4.23, "Get runtime capabilities (`CC`)".

Table 7. Communication capabilities look up table

Bit number	Command string	Description
0		*RESERVED*
1	`t`	Get firmware version
2 - 31		*RESERVED*
32	`CC`	Get runtime capabilities
33	`CM`	Get MethodSCRIPT capabilities
34	`S`	Set register
35	`G`	Get register
36	`l`	Load MethodSCRIPT
37	`r`	Run loaded MethodSCRIPT
38	`e`	Execute (= load and run) MethodSCRIPT
39	`dlfw`	Enter bootloader
40 - 42		*RESERVED*
43	`Fmscr`	Store loaded MethodSCRIPT to NVM
44	`Lmscr`	Load MethodSCRIPT from NVM
45		*RESERVED*
46	`s`	Hibernate (*deprecated*)
47		*RESERVED*
48	`i`	Get serial number
49	`v`	Get MethodSCRIPT version
50		*RESERVED*
51	`fs_dir`	Get directory listing
52	`fs_get`	Read file
53	`fs_put`	Write file
54	`fs_del`	Delete file or directory
55	`fs_info`	Get file system information
56	`fs_format`	Format storage device
57	`fs_mount`	Mount file system

Bit number	Command string	Description
58	`fs_unmount`	Unmount file system
59	`fs_clear`	Clear file system
60	`m`	Get multi-channel serial number
61 - 95		*RESERVED*
96	`h`	Halt script execution
97	`H`	Resume script execution
98	`Z`	Abort script execution
99	`Y`	Abort measurement loop