PalmSens

Communication protocol for Nexus

Version 1.1, 2025-09-12

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`) . . . 5
4.3. Get register (`G`) . . . 6
4.4. Load MethodSCRIPT (`l`) . . . 7
4.5. Run loaded MethodSCRIPT (`r`) . . . 8
4.6. Execute (= load and run) MethodSCRIPT (`e`) . . . 9
4.7. Load MethodSCRIPT from file (`l_fs`) . . . 10
4.8. Execute (= load and run) MethodSCRIPT from file (`e_fs`) . . . 11
4.9. Get serial number (`i`) . . . 12
4.10. Get multi-channel serial number (`m`) . . . 13
4.11. Get MethodSCRIPT version (`v`) . . . 14
4.12. Enter bootloader (`dlfw`) . . . 15
4.13. Get directory listing (`fs_dir`) . . . 15
4.14. Read file (`fs_get`) . . . 17
4.15. Write file (`fs_put`) . . . 18
4.16. Delete file or directory (`fs_del`) . . . 19
4.17. Get file system information (`fs_info`) . . . 19
4.18. Format storage device (`fs_format`) . . . 20
4.19. Mount file system (`fs_mount`) . . . 21
4.20. Unmount file system (`fs_unmount`) . . . 21
4.21. Clear file system (`fs_clear`) . . . 21
4.22. Get runtime capabilities (`CC`) . . . 22
4.23. Get MethodSCRIPT capabilities (`CM`) . . . 22
4.24. Halt script execution (`h`) . . . 23
4.25. Resume script execution (`H`) . . . 24
4.26. Abort script execution (`Z`) . . . 24
4.27. Abort measurement loop (`Y`) . . . 24
4.28. Reverse CV sweep (`R`) . . . 27
5. Register summary . . . 31

5.1. Generic registers . . . 31
5.2. Nexus specific registers . . . 31
6. Register details . . . 32
6.1. Permission level (0x02) . . . 32
6.2. License register (0x04) . . . 32
6.3. Unique instrument ID (0x05) . . . 33
6.4. Device serial number (0x06) . . . 33
6.5. Reset instrument (0x0B) . . . 34
6.6. Multi-channel role (0x0D) . . . 34
6.7. System date and time (0x0E) . . . 35
6.8. System warning (0x10) . . . 35
6.9. Ip address (0x85) . . . 36
6.10. Mute (0x87) . . . 36
7. Error handling . . . 37
8. Version changes . . . 39
Version 1.0 . . . 39
Version 1.1 . . . 39
Appendix A: Error codes . . . 40
Appendix B: MethodSCRIPT capabilities bit fields . . . 48
Appendix C: Communication capabilities bit fields . . . 53

Communication protocol for Nexus
Last document update: 2025-09-12

Chapter 1. Introduction

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

Initial communication with the Nexus 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

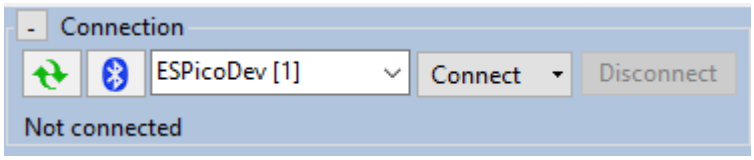
The Nexus features two communication interfaces: Ethernet (preferred) and USB. For USB, the CDC protocol is used, which means that the device identifies itself as a device with a (virtual) serial port. This is also called a Virtual COM Port (VCP). Although the virtual COM port has a number of options that can be configured, for the USB interface these options do not have any effect.

For the Ethernet connection, the IP address is shown on the display and the port is `49152`. This is just a raw TCP port, which can be easily connected to applications such as TeraTerm or Putty. There is no authentication required, so the instrument is available directly after opeing this port. It also allows to easily connect to from any programming or scripting language by opening this port.

2.1. Connection viewer

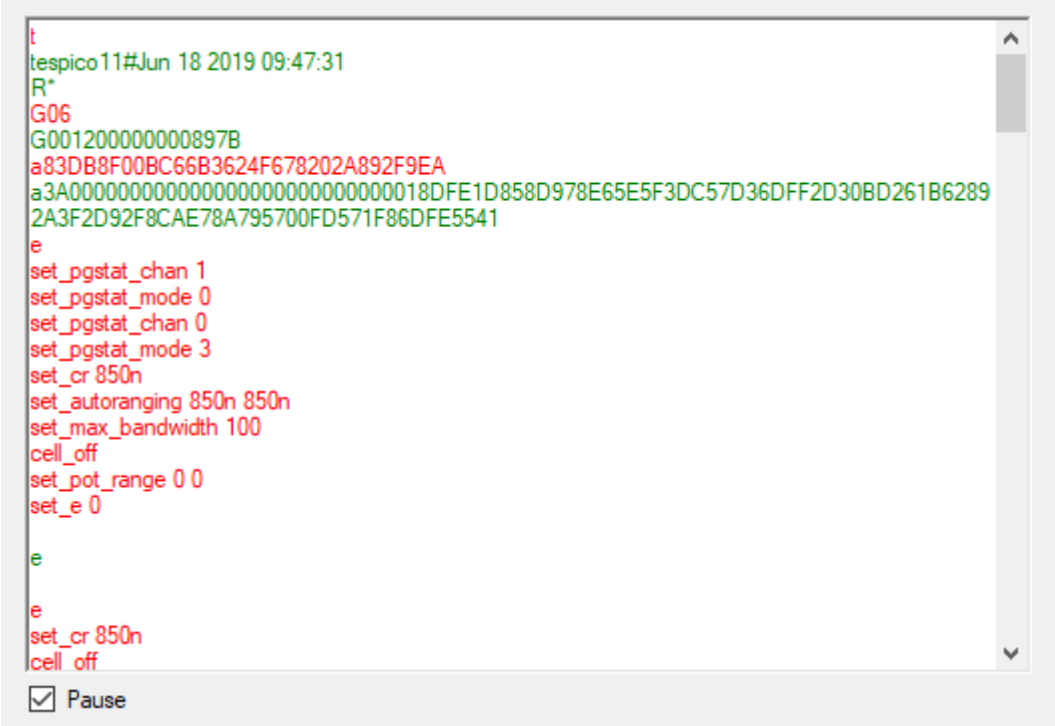
PSTrace version 5.6 or higher has a hidden feature that is useful when the communication protocol is used for development of software for the Nexus. PSTrace will open the *Connection viewer* window when you double-click on the "Not connected" label before connecting to the device.

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 7, *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
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
0x3E	`l_fs`	Idle	Load MethodSCRIPT from file
0x3F	`e_fs`	Idle	Execute (= load and run) MethodSCRIPT from file
0x60	`h`	Script	Halt script execution
0x61	`H`	Script	Resume script execution
0x62	`Z`	Script	Abort script execution
0x63	`Y`	Script	Abort measurement loop
0x65	`R`	Script	Reverse CV sweep

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*
```

Example

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

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 6.1, “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 6.1, "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

```
G001200000000899B
```

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.

```
1 l
2 mm
3 ..
4 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
```

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

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. Load MethodSCRIPT from file (`l_fs`)

Load a MethodSCRIPT stored on the filesystem into RAM. The end of the script is indicated by an empty line (i.e., a line containing only the newline character `\n`). 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

```
1 l_fs filename
```

Key	Type	Size	Description
`filename`	text	variable	The path of the MethodSCRIPT on the filesystem

Response format

```
l
```

Example

The following example loads a MethodSCRIPT from a file named `scripts/my_script`. If the script is loaded successfully, a subsequent `r` command would run it.

Example command

```
l_fs scripts/my_script
```

Example output

```
l
```

4.8. Execute (= load and run) MethodSCRIPT from file (`e_fs`)

Load and run a MethodSCRIPT from the filesystem (same as `l_fs` followed by `r`).

Command format

```
e_fs filename
```

Key	Type	Size	Description
`filename`	text	variable	The path of the MethodSCRIPT on the filesystem

Response format

```
e
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 example loads and runs a MethodSCRIPT from a file named `scripts/my_script`.

For this example, we presume that `scripts/my_script` contains the following:

Example script file contents (the newline characters are included here for clarity)

```
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 command

```
e_fs scripts/my_script
```

Example output

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

4.9. Get serial number (`i`)

Get the serial number of the instrument.

For some instruments, this is not the same as the serial printed on the housing.

Command format

```
i
```

Response format

```
ixx..xx
```

Key	Type	Size	Description
`xx..xx`	text	variable	The serial number.

Example

The following example queries the device serial.

Example command

```
i
```

Example output

```
iNEXS124C0020
```

4.10. 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.

Key	Type	Size	Description
`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
```

4.11. 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 Nexus firmware versions and the associated MethodSCRIPT versions is given below.

Nexus firmware version	MethodSCRIPT version
1.0.0	01.07.00
1.1.0	01.08.00

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 = 1.6.0)

```
v01.06.00
```

4.12. Enter bootloader (`dlfw`)

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

Command format

```
dlfw
```

Response format

```
d
```

4.13. Get directory listing (`fs_dir`)

Get a list of all files in the specified directory.

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

On EmStat4 and EmStatPico based devices `fs_dir` recursively shows files in subdirectories. On the Nexus, only the immediate directory contents are shown.

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 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 date†, year
`MM`	dec	2*	File date†, month (01-12)
`DD`	dec	2*	File date†, day (01-31)
`hh`	dec	2*	File time†, hours (00-23)
`mm`	dec	2*	File time†, minutes (00-59)
`ss`	dec	2*	File 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 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.

Depending on the device, the timestamp associated with a file may be its creation time or its last modification time. On the EmStatPico, Sensit Wearable, and EmStat4, it is the creation time. On the Nexus, it is the last modification time.

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
```

4.14. 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
```

4.15. 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
```

4.16. Delete file or directory (`fs_del`)

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

This can take a long time for file trees containing many elements.

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
```

4.17. 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. Because files are allocated in blocks the actual size of the file will always be a multiple of this block size. For example, by writing 100 bytes to a file, the used space could

increase with 8 kB, and the free size decrease accordingly.

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
```

4.18. 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. The `fs_clear` command is usually much faster than the `fs_format` command.

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

Command format

```
fs_format
```

Response format

```
f
```

4.19. Mount file system (`fs_mount`)

Mount the file system.

Command format

```
fs_mount
```

Response format

```
f
```

4.20. 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
```

4.21. 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
```

4.22. 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
```

4.23. Get MethodSCRIPT capabilities (`CM`)

Get the MethodSCRIPT capabilities. Return a list of MethodSCRIPT commands that are licensed and supported by 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

```
CYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYYY
```

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

Example

Example command

```
CM
```

Example response

```
C000000000000000000000000000000000000002BFFF87FFFFFFFFFFBFFFFE
```

4.24. Halt script execution (`h`)

Halt execution of the running MethodSCRIPT.

This "pauses" the script. Execution can be resumed using the `H` command (see Section 4.25, “Resume script execution (`H`)”).

Command format

```
h
```

Response format

```
h
```

Example

See the examples in Section 4.27.

4.25. Resume script execution (H)

Resume execution of the halted MethodSCRIPT.

Command format

```
H
```

Response format

```
H
```

Example

See the examples in Section 4.27.

4.26. 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 4.27.

4.27. 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.

4.28. Reverse CV sweep (`R`)

During a CV (but not fast CV) sweep, reverse the sweep direction. This has the same effect as the MethodSCRIPT command `set_scan_dir 0`. Depending on the exact location where the reversal occurs, this may end the current scan early and advance to the next, if present. This command has no effect if run outside of a CV sweep.

Command format

```
R
```

Response format

```
R
```

Example

The plots below show the behaviour of the CV reverse command.

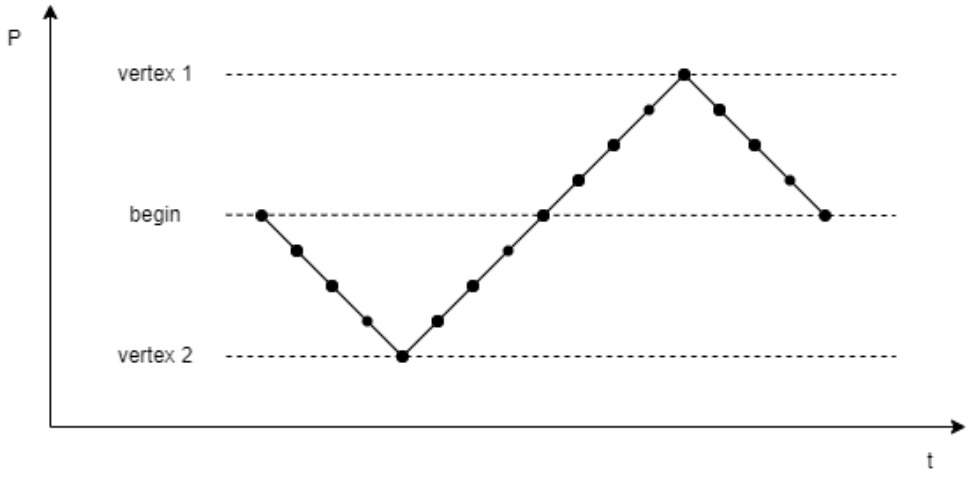


Figure 1. CV Sweep without reversal

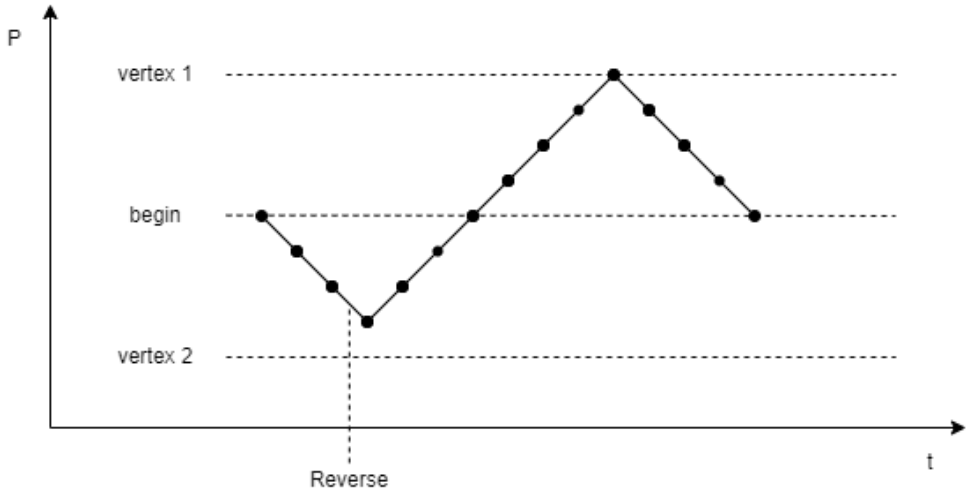


Figure 2. CV Sweep reversal short-cutting to next segment

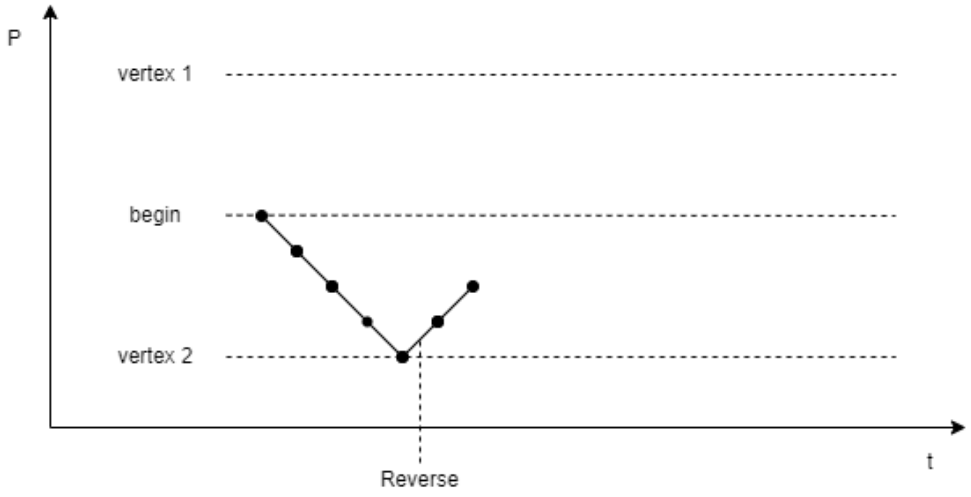


Figure 3. CV Sweep reversal ending the scan

The following MethodSCRIPT examples demonstrate the same behaviour.

This script performs a 3 vertex CV measurement, from 0 V to -1 V to 1 V, with steps of 250 mV and a scan rate of 1 V/s. Here only the potentials are sent back, for simplicity.

```
e
var c
var p
set_pgstat_chan 0
set_pgstat_mode 2
set_max_bandwidth 40
set_range ba 2100u
set_autoranging ba 210n 21m
set_e 0
cell_on
meas_loop_cv p c 0 -1 1 250m 1
pck_start
pck_add p
pck_end
endloop
on_finished:
cell_off
```

This results in 17 points.

```
e
M0005
Pda8000000
Pda7FC2F23u
Pda7F85E45u
Pda7F48D67u
Pda7F0BC8Au
Pda7F48D67u
Pda7F85E45u
Pda7FC2F23u
Pda8000000
Pda803D0DDu
Pda807A1BBu
Pda80B7299u
Pda80F4376u
Pda80B7299u
Pda807A1BBu
Pda803D0DDu
Pda8000000
*
```

Issuing the reverse command causes the sweep to change direction early. Below it can be seen that the sweep

only performs 3 steps in its initial direction instead of 4.

It can be seen that there is a delay between the `R` command being echo'd, and the data reversing. This occurs due to the sweep potentials being set in advance, and so it shouldn't be expected that the `R` command will take immediate effect.

```
e
M0005
Pda8000000
Pda7FC2F23u
Pda7F85E45u
R
Pda7F48D67u
Pda7F85E45u <--- Previously this was another step down to Pda7F0BC8Au
Pda7FC2F23u
Pda8000000
Pda803D0DDu
Pda807A1BBu
Pda80B7299u
Pda80F4376u
Pda80B7299u
Pda807A1BBu
Pda803D0DDu
Pda8000000
*
```

If the reversal would cause the sweep to change to a potential and direction that do not appear later in the current scan, then the sweep will advance to the next scan. If the CV is already in the final scan, it will end the CV instead. This can be seen in the below data.

```
e
M0005
Pda8000000
Pda7FC2F23u
Pda7F85E45u
Pda7F48D67u
Pda7F0BC8Au
R
Pda7F48D67u
Pda7F85E45u
*
```

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
0x02	Permission level	4	Write only	Write only
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
0x0B	Reset instrument	4	Write only	Write only
0x0D	Multi-channel role	1	Read only	Read / write
0x0E	System date and time	7	Read / write	Read / write
0x10	System warning	4	Read only	Read only

5.2. Nexus specific registers

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

ID	Description	Length (bytes)	Basic permission	Advanced permission
0x85	Ip address	4	Read only	Read only
0x87	Mute	1	Read / write	Read / write

Chapter 6. Register details

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

Some registers are write protected at startup and must be unlocked before use. The tables in Chapter 5, *Register summary* show which access rights each register has for each permission level. The Permission level (`0x02`) register can be used to set the 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.

6.1. 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 1, “Permission keys”.

Table 1. 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.

6.2. 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
```

6.3. 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
```

6.4. 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

```
G0012000000000899B
```

6.5. 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
```

6.6. 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.

Options are:

Value	Description
0x00	Stand-alone, no multi-instrument
0x4D	Master

Value	Description
0x53	Slave
0xFF	Standalone, no multi-instrument

Register format

```
mm
```

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

6.7. 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 file system and for the MethodSCRIPT command `rtc_get`. 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

6.8. 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
```

6.9. Ip address (`0x85`)

This returns the IP address of the instrument.

Register format

```
hhhhhhhh
```

Key	Size (bytes)	Description
`hh..hh`	4	The hexadecimal representation of the IP address

6.10. Mute (`0x87`)

Mute all sounds from the device.

Value	Setting
0	Unmuted
1	Muted

Register format

```
hh
```

Key	Size (bytes)	Description
`hh`	1	Either 1 or 0

Chapter 7. 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 8. Version changes

Version 1.0

- Initial release

Version 1.1

- Updated MethodSCRIPT to version 1.08.00
- Added load and execute MethodScript from file

Appendix A: Error codes

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

The error codes and their meaning are the same for all instruments and firmware versions. However, in some cases, the same error condition could result in a different error code when using another instrument or firmware version.

Table 2. 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
0x001B	This command or part of this command is not supported by the current device
0x001C	Step Potential must at least 1 DAC LSB for this technique

Error code	Description
0x001D	Pulse Potential must at least 1 DAC LSB for this technique
0x001E	Amplitude must at least 1 DAC LSB this technique
0x001F	Product is not licensed for this technique
0x0020	Cannot have more than one high speed and/or max range mode enabled
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
0x003B	An error has occurred while attempting to mount the filesystem
0x003C	An error has occurred while attempting to format the filesystem memory

Error code	Description
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	The multiplexer was not found.
0x0047	The filesystem has to be mounted to complete this action.
0x0048	This device is not a multi-device, no serial available.
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 selected GPIO pin mode is not allowed (by NVM config or device type).
0x0057	The on-board flash module has timed out.
0x0058	Timing error during fast measurement (possibly caused by communication).
0x005A	The instrument cannot meet the requested measurement timing.
0x005B	The variable type is already being measured.
0x006D	The COMM command expected an hexadecimal value, but received something else.
0x006E	The COMM command expected a decimal value, but received something else.
0x0071	The provided key does not fit the lock on this register.
0x0072	I2C port expander did not acknowledge a command

Error code	Description
0x0073	Filesystem module not supported
0x0074	The IP address is not available (yet).
0x007A	There is no measurement channel left for the requested measurement.
0x007B	Temperature measurements during EIS with > 8 kHz are not supported.
0x007C	The specified mode is unknown
0x007D	The ADXL367 did not acknowledge an I2C command
0x007E	An unexpected error occurred during an I2C operation.
0x007F	I2C bus timeout during I2C operation (probably caused by I2C target device).
0x0080	The CE is oscillating.
0x0082	Operation requires system warnings to be cleared.
0x0083	Filesystem operations are not supported on this device.
0x0084	The requested variable type does not support ranging.
0x0085	The selected GPIO pin does not support harware synchronisation.
0x0086	Hardware select must be disabled before the `role` pin can be disabled.
0x0087	The `role` pin must be configured before the hadware select can be enabled
0x0088	The instrument has reserved this GPIO pin to be controlled by hardware (e.g. file system or HW-sync).
0x0089	This GPIO pin cannot be unlocked, as it was not locked in the first place
0x008A	This GPIO pin can only be used for interfacing with a specific external memory
0x008B	The BiPot should be disabled.
0x008C	iR compensation should be disabled.
0x008D	The key provided for the reset command is incorrect.
0x008E	The SPI interface is not confgiured while it is required for the filesystem
0x008F	The SPI interface requires the SPI pins to be configured to 'peri 1'
0x0091	The GPIO is locked for a multiplexer (e.g. Mux8R2 or PicoMux)
0x0092	The GPIO is locked for external storage (e.g. SD-card or NAND flash)
0x0093	The GPIO is locked for an external LED
0x0094	The GPIO is locked for hardware synchronisation
0x0095	The GPIO is locked for external PGStat signals
0x0096	The GPIO is locked for some special purpose on this instrument
0x0097	An attempt was made to access a GPIO using a key while it is unlocked
0x0098	The configuration set using the Peripheral configuration (0x01) register is invalid

Error code	Description
0x0099	Filesystem file is corrupt
0x009A	Filesystem failed to format
0x009B	Filesystem I/O error
0x009C	Filesystem didn't have enough memory to perform the operation
0x009D	Filesystem path was too long to handle
0x009E	Filesystem the path was not valid
0x009F	Filesystem could not find the file specified
0x00A0	Filesystem FM not supported
0x00A1	Filesystem doesn't have a listing
0x00A2	Filesystem is not initialized
0x00A3	Filesystem file is open, but it should not have been
0x00A4	Filesystem file is not open
0x00A5	Filesystem does not support this feature
0x00A6	Filesystem expected something which is not true.
0x00A7	Filesystem could not find the path
0x00A8	Access denied due to prohibited access or directory full
0x00A9	The file/directory object is invalid
0x00AA	The physical drive is write protected
0x00AB	The logical drive number is invalid
0x00AC	There is no valid filesystem volume
0x00AD	The format operation was aborted due to any problem
0x00AE	The operation is rejected according to the file sharing policy
0x00AF	Working buffer could not be allocated
0x00B0	Too many files opened at once by filesystem
0x00B1	Parameter given to the filesystem is invalid
0x00B2	The file mode is invalid (should be readonly, new, append, overwrite).
0x00B3	The pin mode requred for the LED mapping is not allowed for this pin
0x00B4	The pin mode requred for the HW-sync role is not allowed for this pin
0x00B5	The pin mode requred for the HW-sync start mapping is not allowed for this pin
0x00B6	Use of the encrypted filesystem failed
0x00B7	User encryption key is already set

Error code	Description
0x00B8	The communications protocol is not in valid lock state for this command
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
0x4017	Arrays must be the same size but are not
0x4018	The script has ended unexpectedly.
0x4019	The script command is only valid for a multichannel (combined) device
0x401A	The script command 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.
0x401D	The file path is too long for the file system.
0x401E	Insufficient memory to store array index
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` command
0x4029	The technique requires that at least one step should be made
0x402A	The variable names do not fit in memory anymore, try using shorter names.
0x402B	The variable name did not start with 'a'-'z' or otherwise contained anything other than 'a'-'z', '0'-'9' and '_'.
0x402C	The variable name is too long to be processed.
0x402D	The file mode is invalid.
0x402E	The file mode does not support a counter in the file path.
0x402F	The file path with the maximum counter value already exists.
0x4030	There are too many files open already.
0x4031	The specified multi device type is not defined.
0x4032	Cannot set the potential (or potential range) within the active measurement loop.
0x4033	Cannot set the current (or current range) within the active measurement loop.
0x4034	The used feature is not licensed on this product.
0x4035	The given filter type is unknown or not supported.
0x4036	The given command is only allowed within measurement loops.
0x4037	A computation has resulted in an overflow
0x4038	The array access was not correctly formed
0x4039	The literal argument was not correctly formed
0x403A	The subarray declaration was out of bounds for the source array
0x403B	A file needs to be opened before it can be written to
0x403C	The MethodScript output mode is unknown
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

Error code	Description
0x4208	MScript argument reference was invalid (not 'a' - 'z')
0x4209	MScript argument variable type is invalid or 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
0x4210	An f-string contains an opening brace that is never closed
0x4211	MScript argument is an array element, which is not supported by this command
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 4.23, "Get MethodSCRIPT capabilities (`CM`)"

Table 3. 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	`mux_config`
82	`mux_get_channel_count`
83	`mux_set_channel`
84	`set_gpio_msk`
85	`get_gpio_msk`
86	`set_e_aux`
87	*RESERVED*
88	`set_ir_comp`

Bit number	Command string
89	*RESERVED*
90	`meas_fast_cv`
91	`set_acquisition_frac_autoadjust`
92	`alter_vartype`
93	`meas_loop_acv`
94	`meas_ms_eis`
95	`meas_fast_ca`
96	`mod_var`
97	`notify_led`
98	`set_scan_dir`
99	`meas_loop_ca_alt_mux`
100	`meas_loop_cp_alt_mux`
101	`meas_loop_ocp_alt_mux`
102	`smooth`
103	`peak_detect`
104	`set_bipot_mode`
105	`set_bipot_potential`
106	`meas_loop_eis_dual`
107	`rtc_get`
108	*RESERVED*
109	`beep`
110	`battery_perc`
111	`get_progress`
112	`pow_var`
113	`subarray`
114	`log_var`
115	`linear_fit`
116	`mean`
117	`trim_enable`
118	`meas_scp`
119	`display_text`

Bit number	Command string
120	`display_btns`
121	`display_clear`
122	`display_progress`
123	`display_icon`
124	`display_draw`
125	`display_inp_num`
126	`display_scroll_add`
127	`display_scroll_get`
128	`display_keyboard`
129	`qr_scan`

Appendix C: Communication capabilities bit fields

The following table lists all MethodSCRIPT commands and their respective bit field in the Section 4.22, "Get runtime capabilities (`CC`)".

Table 4. 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 - 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
58	`fs_unmount`	Unmount file system
59	`fs_clear`	Clear file system

Bit number	Command string	Description
60	`m`	Get multi-channel serial number
61		*RESERVED*
62	`l_fs`	Load MethodSCRIPT from file
63	`e_fs`	Execute (= load and run) MethodSCRIPT from file
64	`sfs_put`	Write file with cryptographic verification
65	`sfs_del`	Delete file with cryptographic verification
66	`sfs_format`	Format filesystem with cryptographic verification
67	`sfs_clear`	Clear filesystem with cryptographic verification
68	`comm_lock`	Comm Lock
69	`comm_unlock`	Comm Unlock
70 - 95		*RESERVED*
96	`h`	Halt script execution
97	`H`	Resume script execution
98	`Z`	Abort script execution
99	`Y`	Abort measurement loop
100		*RESERVED*
101	`R`	Reverse CV sweep