

MethodSCRIPT v1.1

Contents

| | | |
|-------|---|----|
| 1 | Introduction | 4 |
| 2 | Features | 5 |
| 2.1 | Release features | 5 |
| 2.2 | Planned future features | 5 |
| 2.3 | Supported devices | 5 |
| 3 | Script format | 6 |
| 4 | MethodSCRIPT variables | 6 |
| 4.1 | Script command variables | 7 |
| 4.2 | Measurement data package variables | 7 |
| 5 | Interpreting measurement data packages | 8 |
| 5.1 | Package format | 8 |
| 5.2 | Variable sub package format | 8 |
| 5.3 | Package parsing example | 9 |
| 6 | Measurement loop commands | 9 |
| 6.1 | Measurement loop example | 9 |
| 6.2 | Measurement loop output | 10 |
| 7 | Variable Types | 11 |
| 8 | Script argument types | 12 |
| 8.1 | var | 12 |
| 8.2 | literal | 12 |
| 8.3 | var_type | 12 |
| 8.4 | integer (int8, int16, int32, uint8, uint16, uint32) | 12 |
| 8.5 | comparator | 12 |
| 8.6 | string | 12 |
| 8.7 | Optional arguments | 12 |
| 9 | Optional arguments | 13 |
| 9.1 | poly_we | 13 |
| 10 | Tags | 14 |
| 10.1 | Supported tags | 14 |
| 10.2 | on_finished: | 14 |
| 11 | Script commands | 15 |
| 11.1 | var | 15 |
| 11.2 | store_var | 15 |
| 11.3 | copy_var | 16 |
| 11.4 | add_var | 16 |
| 11.5 | sub_var | 16 |
| 11.6 | mul_var | 17 |
| 11.7 | div_var | 17 |
| 11.8 | set_e | 17 |
| 11.9 | wait | 17 |
| 11.10 | loop | 18 |

| | | |
|--------|---------------------------------------|----|
| 11.11 | endloop..... | 18 |
| 11.12 | meas..... | 18 |
| 11.13 | meas_loop_lsv | 19 |
| 11.14 | meas_loop_cv | 20 |
| 11.15 | meas_loop_dpv | 21 |
| 11.16 | meas_loop_svv..... | 22 |
| 11.17 | meas_loop_npv | 23 |
| 11.18 | meas_loop_ca | 24 |
| 11.19 | meas_loop_pad..... | 25 |
| 11.20 | meas_loop_ocp | 26 |
| 11.21 | meas_loop_eis | 27 |
| 11.22 | set_autoranging..... | 28 |
| 11.23 | pck_start..... | 28 |
| 11.24 | pck_add..... | 28 |
| 11.25 | pck_end..... | 28 |
| 11.26 | set_max_bandwidth | 29 |
| 11.27 | set_cr..... | 29 |
| 11.28 | cell_on | 29 |
| 11.29 | cell_off | 29 |
| 11.30 | set_pgstat_mode..... | 31 |
| 11.31 | send_string..... | 31 |
| 11.32 | set_gpio | 31 |
| 11.33 | set_pot_range | 32 |
| 11.34 | set_pgstat_chan | 33 |
| 11.35 | set_poly_we_mode | 33 |
| 12 | PGStat Modes | 34 |
| 12.1 | PGStat mode off..... | 34 |
| 12.2 | PGStat mode low speed | 34 |
| 12.3 | PGStat mode high speed | 34 |
| 12.4 | PGStat mode max range..... | 34 |
| 12.5 | PGStat mode poly_we | 34 |
| 13 | Script examples..... | 35 |
| 13.1.1 | EIS example | 35 |
| 13.1.2 | LSV example | 36 |
| 13.1.3 | SWV example..... | 38 |
| 14 | Error handling..... | 40 |
| 15 | Device specific information | 42 |
| 15.1 | PGStat mode properties..... | 42 |
| 15.2 | Current ranges | 43 |
| 15.3 | Other device specific properties..... | 44 |
| 16 | Version changes | 45 |

1 Introduction

The MethodSCRIPT scripting language is designed to improve the flexibility of the PalmSens potentiostat and galvanostat devices for OEM users. It allows users to start measurements with arguments that are similar to the arguments in PSTrace.

PalmSens provides libraries and examples for handling low level communication with the EmStat Pico and generating scripts for supported devices.

Terminology

PGStat: Potentiostat / Galvanostat

CE: Counter Electrode

RE: Reference Electrode

WE: Working Electrode

RHS: Right hand side

LHS: Left hand side

Technique: A standard electrochemical technique

Iteration: A single execution of a loop

2 Features

2.1 Release features

- Measurements can be tested in PStace and then exported to MethodSCRIPT. This allows for convenient testing of different measurements in PStace. The resulting MethodSCRIPT can then be easily imported as a text file and executed from within the user application.
- Support for the following electrochemical techniques:
 - Linear Sweep Voltammetry (LSV)
 - Cyclic Voltammetry (CV)
 - Differential Pulse Voltammetry (DPV)
 - Square Wave Voltammetry (SWV)
 - Normal Pulse Voltammetry (NPV)
 - Chronoamperometry (CA)
 - Open Circuit Potentiometry (OCP)
 - Electrochemical Impedance Spectroscopy (EIS)
 - Pulsed Amperometric Detection (PAD)
- Support for BiPot / Poly WE
- Different measurements can be chained after one another in the same script, making it possible to combine multiple measurements without communication overhead.
- Variables can be stored and referenced to from within the script.
- Up to 26 variables can be declared. This enables temporary storage of measurement data to be sent later.
- Simple math can be performed on variables (add,sub,mul,div).
- Support for loops.
- Support for user code during a measurement step.
- Exact timing control.
- Scripts will be verified for syntax and whether or not they can be executed by the device. If there is an error, the location of the error will be communicated.

2.2 Planned future features

- Low power modes (sleep, hibernate) (already available outside of script).
- Storing of measuring data to an SD card (if SD card is available).
- Autorun script at certain time intervals or at start up.
- In script support for an external RTC (if RTC is available).
- Conditional statements (if, else, elseif, endif)
- Checksum for measurement data packages to check their validity.
- Variables can also be declared as arrays. Up to 4000 variables can be used. This allows for fast burst measurements that are not slowed down by communication.

2.3 Supported devices

- EmStat Pico

3 Script format

The script consists of a series of pre-defined commands. Each command starts with the command string, followed by a pre-defined number of arguments. Arguments are separated by a ' ' (space) character. Each command is terminated by a '\n' character. The '\n' is omitted in most examples. Each line is limited to a maximum of 128 characters. Comments can be added by having the first character on the line be '#'.

To send a script to the device, first send "e\n". This sets the device into MethodSCRIPT mode. To terminate the script, add a line containing only a '\n'.

The following example shows a short script that simply declares a variable, including the '\n' characters:

```
e\n
#This is a comment\n
send_string "hello world"\n
\n
```

The response to this script will be:

```
e\n <- Ack of the execute script cmd 'e'
Thello world\n <- Reply of the "send_string "hello world"" cmd
\n <- End of script
```

4 MethodSCRIPT variables

MethodSCRIPT variables are represented by a 28 bit integer number with a SI prefix from "Table 1: SI prefix conversion table". Only SI prefixes available in this table can be used. A variable with a value of "100" and a prefix of "m" translates to a floating point value of 0.1. Variables are not explicitly linked to a unit; instead the unit is implied by the associated "Variable Type". Refer to section "Variable Types" for more information. Representation of MethodSCRIPT variables changes depending on whether the variable is part of a script command or part of a measurement data package.

| SI prefix | Text | Factor |
|-----------|-------|-------------------|
| 'a' | atto | 10 ⁻¹⁸ |
| 'f' | femto | 10 ⁻¹⁵ |
| 'p' | pico | 10 ⁻¹² |
| 'n' | nano | 10 ⁻⁹ |
| 'u' | micro | 10 ⁻⁶ |
| 'm' | milli | 10 ⁻³ |
| '' | none | 10 ⁰ |
| 'k' | kilo | 10 ³ |
| 'M' | Mega | 10 ⁶ |
| 'G' | Giga | 10 ⁹ |
| 'T' | Tera | 10 ¹² |
| 'P' | Peta | 10 ¹⁵ |
| 'E' | Exa | 10 ¹⁸ |

Table 1: SI prefix conversion table

4.1 Script command variables

MethodSCRIPT variables that are part of the MethodSCRIPT sent to the device are represented as a signed integer followed by a prefix. For example, a value of -0.01 would be represented as “-10m”.

4.2 Measurement data package variables

MethodSCRIPT variables that are part of a measurement data package are represented as 28 bit unsigned hexadecimal values with an offset of $0x8000000$ (2^{27}). Each variable has one of the SI prefixes shown in Table 1: SI prefix conversion table.

This format looks as follows:

```
HHHHHHHp
```

Where:

HHHHHHH = Hexadecimal value.

p = Prefix character.

For example, a value of 0.01 would be represented as “800000Am” and a value of -0.01 would be represented as “7FFFFFF6m”. PalmSens provides source code examples that showcase how to parse measurement data.

To convert a MethodSCRIPT variable to a floating point value, the following pseudocode can be used:

```
(HexToUint32(HHHHHHH) - 2^27) * SIFactorFromPrefix(p)
```

To convert a floating point value to a MethodSCRIPT variable, the following pseudocode can be used:

```
Uint32ToHex(value) / SIFactorFromPrefix(p) + 2^27
```

Most programming languages have a built in way of converting a HEX string to an integer. The function SIFactorFromPrefix can use a hash table lookup or a switch case to translate the prefix character to its corresponding factor.

5 Interpreting measurement data packages

5.1 Package format

Measurement packages consist of a header, followed by any amount of “variable” packages (each with their own “variable type”), followed by a terminating ‘\n’ character. “Table 2: Measurement data package format” shows this format. Section “Variable sub package format” explains the format of the variable fields.

| Header | Var 1 | Var separator | Var 2 | Var separator | Var X | Term |
|--------|----------|---------------|----------|---------------|----------|------|
| P | Variable | ; | Variable | ; | Variable | \n |

Table 2: Measurement data package format

5.2 Variable sub package format

The format for a variable sub package is:

| Var 1 | Var 1 metadata 1 | Var 1 metadata X |
|------------|------------------|------------------|
| ttHHHHHHHp | ,MV..V | ,MV..V |

Table 3: Variable sub package format

Where:

- tt = Variable Type, represented as a base26 identifier that ranges from “aa” to “zz”. Variable Types are always lower case. See section “Variable Types” for more information.
- HHHHHHHp = MethodSCRIPT package variable. See section “Measurement data package variables” for more information.
- ,
- M = Metadata type ID, see “Table 4: Metadata types”.
- V..V = Metadata value as a hexadecimal value, length is determined by metadata type.

Metadata fields contain extra information about the variable. Each variable can have multiple metadata fields. See “Table 4: Metadata types” for the possible metadata types.

| ID | Name | Length | Content |
|----|---------------|--------|---|
| 1 | Status | 1 | 0 = OK 1 = timing not met (custom commands in the measurement loop took too long for the specified interval of the measurement) 2 = overload (>95% of max ADC value) 4 = underload (<2% of max ADC value) 8 = overload warning (>80% of max ADC value) If an overload or underload is detected, the measured data can be unreliable. |
| 2 | Current range | 2 | Index of current range (device specific, see “Current ranges”). This current range is just intended for diagnostic purposes, and is not used in any calculations during parsing. |

Table 4: Metadata types

5.3 Package parsing example

An EmStat Pico sends the following measurement data package:

```
Pda8000800u;ba8000800u,10,201\n
```

This package contains two variables: "da8000800u" and "ba8000800u,10,201".

The package "da8000800u" can be broken down as follows:

- The Variable Type is "da", this is variable type "VT_CELL_POTENTIAL".
- The value is "08000800 – 0x8000000" = 0x800 = 2048. The prefix is "u" which stands for "micro". This makes the final value 2048 uV (or 2.048 mV).
- This variable has no metadata.

The package "ba8000800u,10,201" can be broken down as follows:

- The Variable Type is "ba", this corresponds to Variable Type "VT_CURRENT_WE".
- The value is "08000800 – 0x8000000" = 0x800 = 2048. The prefix is "u" which stands for "micro". This makes the final value 2048 uA (or 2.048 mA).
- This variable has two metadata packages, the first has an ID of "1" and a value of 0, indicating it is a status package with the value "OK". The second metadata package has an ID of "2" and a value of 01. This indicates that it is a current range with the current range "1". For the EmStat Pico, this refers to the "1.95 uA" current range. This current range is just for diagnostic purposes, and is not used in any calculations during parsing.

6 Measurement loop commands

All techniques are implemented as "measurement loop commands". This means that the command will execute one iteration of the measurement technique. After this, all MethodSCRIPT commands within the measurement loop are executed. When all commands have been executed, the device waits for the correct timing to start the next iteration of the measurement technique and the process begins again for the next iteration.

It is possible that the script steps in the loop take more time than is available between each iteration. If this happens, the next measurement iteration is delayed. It is the responsibility of the user to ensure there is enough time between measurement iterations to execute the user commands in the loop.

6.1 Measurement loop example

The following example shows a typical Chrono Amperometry measurement loop :

```
#Run a measurement loop for the Chrono Amperometry technique
meas_loop_ca p c 100m 100m 2
#These user commands are executed after one measurement
#iteration has been done
pck_start
pck_add p
pck_add c
pck_end
#At "end loop", the script execution halts until it is time for the
#next measurement loop iteration
endloop
```

6.2 Measurement loop output

At the start of each measurement loop, the following line is sent from the device:

```
MXXXX
```

Where:

M = The header for a measurement loop start package.

XXXX = The technique ID of the measurement loop, see "Table 5: Measurement technique ID's"

| ID | Name |
|------|--|
| 0001 | Linear Sweep Voltammetry |
| 0002 | Differential Pulse Voltammetry |
| 0003 | Normal Pulse Voltammetry |
| 0005 | Cyclic Voltammetry |
| 0007 | Chrono Amperometry |
| 0008 | Pulsed Amperometric Detection |
| 000B | Open-Circuit Chrono Potentiometry |
| 000D | Electrochemical Impedance Spectroscopy |

Table 5: Measurement technique ID's

When a measurement loop is completed the following line is sent:

```
*
```

The following example shows the output of a EIS measurement loop command:

```
M000D  
... data packages ...  
*
```

7 Variable Types

Variable Types offer context to MethodSCRIPT variables. They communicate the unit and the origin of the variable. They are also used as an argument to some functions to measure a specific type of variable. For example, when the “meas” command is used, the type of variable to measure must be passed as an argument. Table 6: Variable Types shows the available variable types.

| Measurable types | ID | Description |
|-----------------------------------|----|--|
| VT_UNKNOWN | aa | Unknown (not initialized) |
| VT_POTENTIAL_RE | ab | Measured RE voltage |
| VT_POTENTIAL_CE | ac | Measured CE voltage |
| VT_POTENTIAL_WE | ad | Measured WE voltage |
| | | |
| VT_POTENTIAL_AUX1_IN | as | Measured aux1 voltage |
| VT_POTENTIAL_AUX2_IN | at | Measured aux2 voltage |
| | | |
| VT_CURRENT_WE | ba | Measured WE current |
| | | |
| VT_PHASE | cp | Measured phase |
| VT_IMP | ci | Measured impedance |
| VT_ZREAL | cc | Measured real part of complex impedance |
| VT_ZIMAG | cd | Measured imaginary part of complex impedance |
| Appliable types | ID | Description |
| VT_CELL_POTENTIAL | da | Set control value for cell potential |
| VT_CELL_CURRENT | db | Set control value for cell current |
| VT_CELL_FREQUENCY | dc | Set value for frequency |
| VT_CELL_AMPLITUDE | dd | Set value for ac amplitude |
| Generic types (reserved for user) | ID | Description |
| VT_CURRENT_GENERIC1 | ha | |
| VT_CURRENT_GENERIC2 | hb | |
| VT_CURRENT_GENERIC3 | hc | |
| VT_CURRENT_GENERIC4 | hd | |
| VT_POTENTIAL_GENERIC1 | ia | |
| VT_POTENTIAL_GENERIC2 | ib | |
| VT_POTENTIAL_GENERIC3 | ic | |
| VT_POTENTIAL_GENERIC4 | id | |
| VT_MISC_GENERIC1 | ja | |
| VT_MISC_GENERIC2 | jb | |
| VT_MISC_GENERIC3 | jc | |
| VT_MISC_GENERIC4 | jd | |

Table 6: Variable Types

8 Script argument types

8.1 var

The argument “var” is a reference to a MethodSCRIPT variable. Variables can be changed during runtime.

8.2 literal

A literal is a constant value argument, it cannot change during runtime.

8.3 var_type

See section “Variable Types”

8.4 integer (int8, int16, int32, uint8, uint16, uint32)

These are integer constants, these cannot be changed and do not accept SI prefixes.

8.5 comparator

Comparator operator for Boolean logic, these include:

- The equals operator “==”
- The not equals operator “!=”
- The greater than operator “>”
- The greater than or equal to operator “>=”
- The smaller than operator “<”
- The smaller than or equal to operator “<=”

8.6 string

A string constant argument, a string is always encapsulated in double quotes (“”).

8.7 Optional arguments

Some commands can have optional arguments to extend their functionality. For example most techniques support the use of a second working electrode (bipot or poly_we). See chapter “Optional arguments” for detailed information.

9 Optional arguments

Optional arguments are added after the last mandatory argument. The format is "cmd_name(arg1 arg2 arg3 ..)"

9.1 poly_we

Arguments

| Name | Type | |
|----------------|-------|---|
| Channel | uint8 | Channel of the additional working electrode |
| Output current | var | Output variable to store the measured current in. |

Example

```
e
#declare variable for output potential
var p
#declare variable for output current of main WE
var c
#declare variable for output current of secondary WE
var b
#enable bipot on ch 1
set_pgstat_chan 1
#set the selected channel to bipot mode
set_pgstat_mode 5
#set bp mode to offset or constant
set_poly_we_mode 1
#set offset or constant voltage
set_e 100m
#set the current-range of the secondary WE
set_cr 1u
#switch back to do actual measurement on ch 0
set_pgstat_chan 0
#set the main WE channel to low speed mode
set_pgstat_mode 2
set_cr 1u
set_pot_range 0m 0m
set_max_bandwidth 500
set_e -500m
cell_on
wait 1
#LSV measurement using channel 0 as WE1 and channel 1 as WE2
#WE2 current is stored in var b
meas_loop_lsv p c -500m 500m 5m 100m poly_we(1 b)
    pck_start
    pck_add p
    pck_add c
    pck_add b
    pck_end
endloop
cell_off
```

Perform an LSV measurement and send a data packet for every iteration. The data packet contains the set potential (p), the measured current of the main WE (c) and the measured current of the secondary WE (b). The LSV performs a potential scan from -500 mV to 500 mV with steps of 10 mV at a rate of 100 mV/s. This results in a total of 101 data points at a rate of 10 points per second.

10 Tags

A script can have optional tags (or labels) to direct the execution flow in case of an event like aborting a running script.

10.1 Supported tags

| Name | Description |
|--------------|--|
| on_finished: | The commands after this this tag will be execute when the script is aborted, or normal script execution reaches the tag. |

10.2 on_finished:

Example

```
meas_loop_eis h r j 10m 200k 100 17 0
pck_start
pck_add h
pck_add r
pck_add j
pck_end
endloop
on_finished:
cell_off
```

The cell will be switched off when the EIS loop is finished or the script is aborted during the EIS loop. If a runtime script error occurs, these commands will not be executed.

11 Script commands

11.1 var

Declare a variable. All variables must be declared before use. Currently only names that consist of 1 lower case character are allowed.

Arguments

| Name | Type | |
|---------------|------|---------------------------|
| Variable name | var | Variable reference (a-z). |

Example

```
var a
```

Declare variable with name "a".

11.2 store_var

Store a value in a variable. This value can be referenced in following commands.

Arguments

| Name | Type | |
|---------------|----------|---|
| Variable name | var | Variable reference. |
| Value | literal | Literal value to store in the variable. |
| Variable Type | var_type | The type identifier for this value, see section "Variable Types". |

Example

```
store_var i 200 ja
```

Store a value of 200 in the variable 'i'. This value is of type: "VT_MISC_GENERIC1".

11.3 copy_var

Copies value from the source address to the destination address.

Arguments

| Name | Type | |
|----------------------|------|----------------------------------|
| Source variable | var | Variable reference to copy from. |
| Destination variable | var | Variable reference to copy to. |

Example

```
copy_var i j
```

Copies the variable 'i' to 'j'.

11.4 add_var

Add "lhs" to "rhs" and store the result in "lhs". Metadata of lhs is maintained.

Arguments

| Name | Type | |
|------|---------------|--|
| Lhs | var | The lhs variable, the result is stored here. |
| Rhs | var / literal | Literal or variable to add to lhs var. |

Example

```
add_var i 1
```

Adds 1 to variable 'i' and stores it to 'i'.

11.5 sub_var

Subtract "rhs" from "lhs" and store the result in "lhs". Metadata of lhs is maintained.

Arguments

| Name | Type | |
|------|---------------|---|
| Lhs | var | The lhs variable, the result is stored here. |
| Rhs | var / literal | Literal or variable to subtract from lhs var. |

Example

```
sub_var i 1
```

Subtracts 1 from the variable 'i' and stores it to 'i'

11.6 mul_var

Multiply “lhs” with “rhs” and store the result in “lhs”. Metadata of lhs is maintained.

Arguments

| Name | Type | |
|------|---------------|--|
| Lhs | var | The lhs variable, the result is stored here. |
| Rhs | var / literal | Literal or variable to multiply lhs by. |

Example

```
mul_var i 1500m
```

Multiplies the variable ‘i’ with 1.5 and stores it to ‘i’

11.7 div_var

Divide “lhs” by “rhs” and store the result in “lhs”. Metadata of lhs is maintained.

Arguments

| Name | Type | |
|------|---------------|--|
| Lhs | var | The lhs variable, the result is stored here. |
| Rhs | var / literal | Literal or variable to divide lhs by. |

Example

```
div var i 1500m
```

Divides the variable ‘i’ by 1.5 and stores it to ‘i’

11.8 set_e

Apply a variable or literal as the cell potential. This determines the potential (WE vs RE). The potential is limited by the potential range of the currently active “pgstat mode” see section “PGStat mode properties”.

Arguments

| Name | Type | |
|-----------|---------------|---------------------------------------|
| Potential | var / literal | The cell potential to apply in volts. |

Example

```
set_e 100m
```

Sets control value for the potentiostat loop to 0.1V.

11.9 wait

Wait for the specified amount of time.

Arguments

| Name | Type | |
|------|---------------|--|
| Time | var / literal | The amount of time to wait in seconds. |

Example

```
wait 100m
```

Wait 100 milliseconds.

11.10 loop

Repeat all commands up to the next “endloop” until the specified condition is matched. All loops must be terminated with an “endloop”.

Arguments

| Name | Type | |
|---------------------------|---------------|---|
| Stop condition lhs | var / literal | Literal or variable to be compared with the rhs variable. |
| Stop condition comparator | comparator | Literal value to store in the variable. |
| Stop condition rhs | var / literal | Literal or variable to be compared with the lhs variable. |

Example

```
var i
store_var i 0 aa
loop i < 10
add_var i 1
endloop
```

Add 1 to i until variable “i” reaches 10.

11.11 endloop

Signals the end of a loop, see “loop” command.

Arguments

No arguments.

11.12 meas

Measure a datapoint of the specified type and store the result as a variable. The datapoint will be averaged for the specified amount of time at the maximum available sampling rate.

Arguments

| Name | Type | |
|-----------------|---------------|--|
| Time to measure | var / literal | The amount of time to spend averaging measured data. |
| Destination | var | Variable to store the measured data in. |
| Var type | var_type | The type of variable to measure, see section “Variable Types”. |

Example

```
meas 100m c ba
```

Measure the signal with the var_type: ba (VT_CURRENT_WE) for 100ms and store the result in the variable ‘c’.

11.13 meas_loop_lsv

Perform a Linear Sweep Voltammetry (LSV) measurement and store the resulting current in a variable. An LSV measurement scans a potential range in small steps and measures the current at each step.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section "Measurement loop" for more information.

Arguments

| Name | Type | |
|------------------|---------------|--|
| Output potential | var | Output variable to store the set potential for this iteration. |
| Output current | var | Output variable to store the measured current in. |
| Begin potential | var / literal | The begin potential for the LSV technique. |
| End potential | var / literal | The end potential for the LSV technique. |
| Step potential | var / literal | The potential increase for each step. Affects the amount of data points per second, together with the scan rate. This is an absolute step. The direction of the scan is determined by "Begin potential" and "End potential". |
| Scan rate | var / literal | The scan rate of the LSV technique. This is the speed at which the applied potential is ramped in V/s. Can only be positive. |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_lsv p c -500m 500m 10m 100m
pck_start
pck_add p
pck_add c
pck_end
endloop
```

Perform an LSV measurement and send a data packet for every iteration. The data packet contains the set potential and measured current. The LSV performs a potential scan from -500 mV to 500 mV with steps of 10 mV at a rate of 100 mV/s. This results in a total of 101 data points at a rate of 10 points per second.

11.14 meas_loop_cv

Perform a Cyclic Voltammetry (CV) measurement. In a CV measurement, the potential is stepped from the begin potential to the vertex 1 potential, then the direction is reversed and the potential is stepped to the vertex 2 potential and finally the direction is reversed again and the potential is stepped back to the begin potential. The current is measured at each step.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section "Measurement loop" for more information.

Arguments

| Name | Type | |
|--------------------|---------------|---|
| Output potential | var | Output variable to store the set potential for this iteration. |
| Output current | var | Output variable to store the measured current in. |
| Begin potential | var / literal | The begin potential for the CV technique. |
| Vertex 1 potential | var / literal | The vertex 1 potential. First potential where direction reverses. |
| Vertex 2 potential | var / literal | The vertex 2 potential. Second potential where direction reverses. |
| Step potential | var / literal | The potential increase for each step. Affects the amount of data points per second, together with the scan rate. This is an absolute step that does not affect the direction of the scan. |
| Scan rate | var / literal | The scan rate of the CV technique. This is the speed at which the applied potential is ramped in V/s. Can only be positive. |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_cv p c 0 500m -500m 10m 100m
pck_start
pck_add p
pck_add c
pck_end
endloop
```

Perform a CV measurement and send a data packet for every iteration. The data packet contains the set potential and measured current. The CV performs a potential scan from 0 mV to 500 mV to -500 mV to 0 mV. The steps of 10 mV at a rate of 100 mV/s. This results in a total of 201 data points at a rate of 10 points per second.

11.15 meas_loop_dpv

Perform a Differential Pulse Voltammetry (DPV) measurement. In a DPV measurement, the potential is stepped from the begin potential to the end potential. At each step, the current (reverse current) is measured, then a potential pulse is applied and the current (forward current) is measured. The forward current minus the reverse current is stored in the “Output current” variable.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section “Measurement loop” for more information.

Arguments

| Name | Type | |
|------------------|---------------|---|
| Output potential | var | Output variable to store the set potential for this iteration. |
| Output current | var | Output variable to store “forward current – reverse current” in. |
| Begin potential | var / literal | The begin potential for the potential scan. |
| End potential | var / literal | The end potential for the potential scan. |
| Step potential | var / literal | The potential increase for each step. Affects the amount of data points per second, together with the scan rate. This is an absolute step that does not affect the direction of the scan. |
| Pulse potential | var / literal | The potential of the pulse. This is added to the currently applied potential during a step. |
| Pulse time | var / literal | The time the pulse should be applied. |
| Scan rate | var / literal | The speed at which the applied potential is ramped in V/s. Can only be positive. Scan rate must be lower than “Step potential / Pulse time / 2”. |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_dpv p c -500m 500m 10m 20m 5m 100m
pck_start
pck_add p
pck_add c
pck_end
endloop
```

Perform a DPV measurement and send a data packet for every iteration. The data packet contains the set potential and “forward current – reverse current”. The DPV performs a potential scan from -500 mV to 500 mV with steps of 10 mV at a rate of 100 mV/s. This results in a total of 101 data points at a rate of 10 points per second. At every step a pulse of 20mV is applied for 5ms

11.16 meas_loop_svw

Perform a Square Wave Voltammetry (SWV) measurement. In a SWV measurement, the potential is stepped from the begin potential to the end potential. At each step, the current (reverse current) is measured, then a potential pulse is applied and the current (forward current) is measured. The forward current minus the reverse current is stored in the “Output current” variable. The pulse length is “1 / Frequency / 2”.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section “Measurement loop” for more information.

Arguments

| Name | Type | |
|------------------------|---------------|--|
| Output potential | var | Output variable to store the set potential for this iteration. |
| Output current | var | Output variable to store “forward current – reverse current” in. |
| Output forward current | var | Output variable to store forward current in. |
| Output reverse current | var | Output variable to store reverse current in. |
| Begin potential | var / literal | The begin potential for the potential scan. |
| End potential | var / literal | The end potential for the potential scan. |
| Step potential | var / literal | The potential increase for each step. This is an absolute step that does not affect the direction of the scan. |
| Amplitude potential | var / literal | The amplitude of the pulse. This value times 2 is added to the currently applied potential during a step. |
| Frequency | var / literal | The frequency of the pulses. |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_svw p c f r -500m 500m 10m 15m 10
pck_start
pck_add p
pck_add c
pck_end
endloop
```

Perform a SWV measurement and send a data packet for every iteration. The data packet contains the set potential and “forward current – reverse current”. The SWV performs a potential scan from -500 mV to 500 mV with steps of 10 mV at a frequency of 10 Hz. This results in a total of 101 data points at a rate of 10 points per second. At every step a pulse of 30mV (2*15mV) is applied for 50ms (1/Frequency/2).

11.17 meas_loop_npv

Perform a Normal Pulse Voltammetry (NPV) measurement. In an NPV measurement, the pulse potential is stepped from the begin potential to the end potential. At each step the pulse potential is applied and the current is measured at the top of this pulse. The potential is then set back to the begin potential until the next step. The measured current is stored in the “Output current” variable.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section “Measurement loop” for more information.

Arguments

| Name | Type | |
|------------------|---------------|---|
| Output potential | var | Output variable to store the set potential for this iteration. |
| Output current | var | Output variable to store the measured current in. |
| Begin potential | var / literal | The begin potential for the potential scan. |
| End potential | var / literal | The end potential for the potential scan. |
| Step potential | var / literal | The pulse potential increase for each step. Affects the amount of data points per second, together with the scan rate. This is an absolute step that does not affect the direction of the scan. |
| Pulse time | var / literal | The time the pulse should be applied. |
| Scan rate | var / literal | The speed at which the applied potential is ramped in V/s. Can only be positive. Scan rate must be lower than “Step potential / Pulse time / 2”. |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_npv p c -500m 500m 10m 20m 5m 100m
pck_start
pck_add p
pck_add c
pck_end
endloop
```

Perform an NPV measurement and send a data packet for every iteration. The data packet contains the set potential and measured pulse current. The NPV performs a potential scan from -500 mV to 500 mV with steps of 10 mV at a rate of 100 mV/s. This results in a total of 101 data points at a rate of 10 points per second. At every step a potential pulse of “step index * step potential” mV is applied for 5ms.

11.18 meas_loop_ca

Perform a Chrono Amperometry (CA) measurement. In a CA measurement, a DC potential is applied the current is measured at the specified interval. The measured current is stored in the “Output current” variable.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section “Measurement loop” for more information.

Arguments

| Name | Type | |
|------------------|---------------|--|
| Output potential | var | Output variable to store the set potential for this iteration. |
| Output current | var | Output variable to store the measured current in. |
| DC potential | var / literal | The DC potential to be applied. |
| Interval time | var / literal | The interval between measured data points. |
| Run time | var / literal | The total run time of the measurement. |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_ca p c 100m 100m 2
pck_start
pck_add p
pck_add c
pck_end
endloop
```

Perform a CA measurement and send a data packet for every iteration. The data packet contains the set potential and measured current. A DC potential of 100 mV is applied. The current is measured every 100 ms for a total of 2 seconds. This results in a total of 20 data points at a rate of 10 points per second.

11.19 meas_loop_pad

Perform a Pulsed Amperometric Detection (PAD) measurement. In a PAD measurement, potential pulses are applied to a DC potential. Each iteration starts at the DC potential, the current is measured before the pulse (idc). Then the pulse potential is applied, and the current is measured at the end of the pulse (ipulse). The output current returns a current value depending of one the 3 modes: dc (idc), pulse (ipulse) or differential (ipulse – idc).

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section “Measurement loop” for more information.

Arguments

| Name | Type | |
|------------------|---------------|---|
| Output potential | var | Output variable to store the set potential for this iteration. |
| Output current | var | Output variable to store “forward current – reverse current” in. |
| DC potential | var / literal | The begin potential for the potential scan. |
| Pulse potential | var / literal | The potential of the pulse. This is the potential that is set during a pulse. It is not referenced to the DC potential. |
| Pulse time | var / literal | The time the pulse should be applied. |
| Interval time | var / literal | The time of the pulse interval |
| Run time | var / literal | Total run time of the measurement |
| mode | uint8 | PAD mode : 1= dc , 2 = pulse , 3 = differential |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_pad p c 500m 1500m 10m 50m 10050m 2
pck_start
pck_add p
pck_add c
pck_end
endloop
```

Perform a PAD measurement and send a data packet for every iteration. The data packet contains the set potential and measured current. A DC potential of 500 mV is applied. A pulse potential of 1500mV is applied every 50 ms for 10 ms and the current is measured on the pulse (mode = pulse). The measurement is 10.05 seconds in total. This results in a total of 201 data points at a rate of 20 points per second.

11.20 meas_loop_ocp

Perform an Open Circuit Potentiometry (OCP) measurement. In an OCP measurement, the CE is disconnected so that no potential is applied. The open circuit RE potential is measured at the specified interval. The measured potential is stored in the “Output potential” variable.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section “Measurement loop” for more information.

Arguments

| Name | Type | |
|------------------|---------------|--|
| Output potential | var | Output variable to store the measured RE potential in. |
| Interval time | var / literal | The interval between measured data points. |
| Run time | var / literal | The total run time of the measurement. |
| <opt. argument> | Optional arg. | See chapter 0 for detailed information |

Example

```
meas_loop_ocp p 100m 2
pck_start
pck_add p
pck_end
endloop
```

Perform an OCP measurement and send a data packet for every iteration. The data packet contains the set measured RE potential. The RE potential is measured every 100 ms for a total of 2 seconds. This results in a total of 20 data points at a rate of 10 points per second.

11.21 meas_loop_eis

Perform an EIS frequency scan and store the resulting Z-real and Z-imaginary in the given variables. EIS does not currently support autoranging. High speed PGStat mode is required for EIS. The following commands currently have no effect on EIS measurements:

- `set_max_bandwidth`: bandwidth is taken from frequency scan ranges.
- `set_pot_range`: pot range is taken from amplitude and DC potential arguments.

This is a measurement loop function and needs to be terminated with an `endloop` command. Refer to section "Measurement loop" for more information.

Arguments

| Name | Type | |
|--------------------|---------------|--|
| Output frequency | var | Output variable to store the set frequency for this iteration. |
| Output Z-real | var | Output variable to store the measured phase in. |
| Output Z-imaginary | var | Output variable to store the measured impedance in. |
| Amplitude | var / literal | Amplitude of the applied sinewave. |
| Start frequency | var / literal | Start frequency of the scan. |
| End frequency | var / literal | End frequency of the scan. |
| Nr of points | var / literal | Number of frequency points to be scanned. |
| DC potential | var / literal | DC potential offset of the applied sinewave. |

Example

```
meas_loop_eis f r i 10m 100k 100 11 0
pck_start
pck_add f
pck_add r
pck_add i
pck_end
endloop
```

Perform an EIS measurement a frequency `f` with 10mV amplitude and stores the Z-real result in `r` and the Z-imaginary result in `j`. 11 points will be measured at frequencies between 100 kHz and 100 Hz, divided on a logarithmic scale.

11.22 set_autoranging

Enable or disable autoranging for all meas_loop_* functions. Autoranging selects the most appropriate current range for the current measured in the last measurement loop iteration. The selected current range is limited by the min and max current arguments. If min expected current and max expected current are the same value, autoranging is disabled.

Arguments

| Name | Type | |
|-------------|---------|--------------------------------------|
| Min current | literal | The min current in this measurement. |
| Max current | literal | The max current in this measurement. |

Example

```
set_autoranging 1u 1m
```

Enable autoranging for currents between 1 uA and 1 mA.

11.23 pck_start

Signal the start of a measurement data packet.

Arguments

No arguments.

Example

```
pck_start
```

Signal the start of a new measurement package.

11.24 pck_add

Add a stored variable to be sent in this data packet.

Arguments

| Name | Type | |
|----------|------|---|
| Variable | var | The variable to add to the data packet. |

Example

```
pck_add i
```

Add variable 'i' to the data packet.

11.25 pck_end

Signal the end of a measurement data package.

Arguments

No arguments.

Example

```
pck_end
```

Signal the end of a measurement data package.

11.26 set_max_bandwidth

Set maximum bandwidth of the signal being measured. Any signal of significant higher frequency than the set bandwidth will be filtered out. There is no defined lower bound to the bandwidth. At max bandwidth the signal is attenuated by up to 1% of the potential or current step.

Arguments

| Name | Type | |
|---------------|---------------|--|
| Max bandwidth | var / literal | The maximum expected bandwidth expected. Anything below this frequency will not be filtered out. |

Example

```
set_max_bandwidth 1k
```

Set the max bandwidth to a frequency of 1 kHz.

11.27 set_cr

Set the current range for the given maximum current. The device will select the lowest current range that can measure this current without overloading.

Arguments

| Name | Type | |
|-------------|---------------|-------------------------------|
| Max current | var / literal | The maximum expected current. |

Example

```
set_cr 500n
```

Set current range to be able to measure a current of 500nA

11.28 cell_on

Turn the cell on, any settings set when the cell was off will be applied here.

Arguments

No arguments.

Example

```
cell_on
```

Turn the cell on. The potentiostat will start applying the configured potential.

11.29 cell_off

Turn the cell off.

Arguments

No arguments.

Example

```
cell_off
```

Turn the cell off. This stops the potentiostat from applying a potential to the cell.

11.30 set_pgstat_mode

Set the pgstat hardware configuration to be used for measurements. Setting the pgstat mode initializes all channel settings to the default values for that mode. See section "PGStat Modes" for more information.

Arguments

| Name | Type | |
|-------------|-------|--|
| PGStat mode | uint8 | Set pgstat mode: 0 = off 2 = low speed 3 = high power 4 = max range 5 = poly_we |

Example

```
set_pgstat_mode 3
```

Set hardware configuration to high power mode.

11.31 send_string

Send an arbitrary string as output of the MethodSCRIPT. This string is prepended by a 'T', this is the "text" package identifier. Avoid sending a '\n' character or non-ASCII characters.

Arguments

| Name | Type | |
|--------|--------|--|
| String | string | An arbitrary string. Surrounded by quotes ("") |

Example

```
send_string "hello world"
```

Sends string "hello world" as output of the MethodSCRIPT.

11.32 set_gpio

Set GPIO pins. Pins with multiple roles that are not configured as GPIO pins are ignored.

Arguments

| Name | Type | |
|----------|--------|---|
| Pin mask | uint32 | Bitmask that represents the state of the bits. Bit 0 is for GPIO0, bit 1 for GPIO1, etc. Bits that are high correspond with a high output signal. |

Example

```
set_gpio 5
```

5 translates to 101 in binary, so pin 2 and 0 are set high, the rest of the pins is set low.

11.33 set_pot_range

Set the expected potential range for this script. Some devices cannot apply their full potential range in one measurement, but need to be set up to reach these potentials beforehand. This function lets you communicate to the device what the voltage range is you expect in your measurement. The device will automatically configure itself to be able to reach these potentials. This function will return an error if the expected voltage range is greater than the dynamic potential range of the device, or if the expected voltage range exceeds the maximum potential limits of the device.

This is a device specific command. The following devices require this command to reach their full potential range:

- EmStat Pico

For these devices the voltage range that can be applied without changing the expected potential range is defined in section “PGStat Modes” as the “dynamic potential range”.

Arguments

| Name | Type | |
|-------------|---------------|---|
| Potential 1 | var / literal | Bound 1 of the expected voltage range for this measurement. |
| Potential 2 | var / literal | Bound 2 of the expected voltage range for this measurement. |

Example

```
set_pot_range 0 1200m
```

Ensure that the next measurement can apply potentials between 0 V and 1.2 V.

11.34 set_pgstat_chan

Select a potentiostat channel. If the device has multiple parallel potentiostat channels, they can be selected with this command. In the future it will be possible to use these two channels parallel to each other, but this feature is not yet available. Refer to section “Other device specific properties” to see how many channels each device has.

Arguments

| Name | Type | |
|---------------|-------|-------------------------------------|
| Channel index | uint8 | The pgstat channel index to select. |

Example

```
set_pgstat_chan 0
```

Selects pgstat channel 0.

11.35 set_poly_we_mode

Selects the mode of the additional working electrode.

Arguments

| Name | Type | |
|--------------|-------|---|
| Poly_we_mode | uint8 | The mode of the additional working electrode: 0 = fixed mode (WEx is relative to RE) 1= offset mode (WEx is relative to WE) |

Example

```
set_poly_we_mode 1
```

The additional working electrode mode is set to offset mode.

12 PGStat Modes

PGStat modes are device wide configurations that affect which hardware is used during measurements. This is necessary for devices that have a choice between multiple measurement hardware with different properties. PGStat modes are device specific, more information can be found in "PGStat mode properties".

12.1 PGStat mode off

All hardware is turned off to save power, no measurements can be done.

12.2 PGStat mode low speed

The hardware configuration that has the best properties for low speed measurements is picked. Usually this means it is less sensitive to high frequency noise and consumes less power. However the maximum bandwidth is limited.

12.3 PGStat mode high speed

The hardware configuration that has the best properties for high speed measurements is used. In general, this will consume more power and be more sensitive to noise. However, it will allow higher frequencies measurements to be done.

12.4 PGStat mode max range

This mode uses a hardware configuration having the highest possible potential range by combining the high and low speed mode. In general, this will consume more power and be more sensitive to noise. The bandwidth is limited to the bandwidth of the low speed mode.

12.5 PGStat mode poly_we

This mode combines the two channels forming a poly_we (bipot) device. In this mode one channel is setup as the main potentiostat and the other as an additional working electrode (bipot).

13 Script examples

Note: The command terminators (\n) are not shown in the examples. These examples can be used on any device that supports MethodSCRIPT, but they contain some commands that are device specific for the EmStat Pico. These commands will be ignored on devices that do not use them.

13.1.1 EIS example

The following example script runs an EIS scan from 200 kHz down to 200 Hz over 11 points. After each point a data packet will be sent containing the: frequency, Z-real, Z-imaginary variables. The amplitude of the sine is set to 10m and no DC potential is applied.

```
e
var h
var r
var j
#Select channel 0
set_pgstat_chan 0
#High speed mode is required for EIS
set_pgstat_mode 3
#Autorange starting at 1mA down to 10uA
set_autoranging 10u 1m
#Cell must be on to do measurements
cell_on
#Run actual EIS measurement
meas_loop_eis h r j 10m 200k 200 11 0
  #Send measurement package containing frequency, Z-real and Z-imaginary
  pck_start
  pck_add h
  pck_add r
  pck_add j
  pck_end
endloop
#Turn cell off when finished or aborted
on_finished:
cell_off
```

Example output:

```
e <- ack of 'e' cmd
M000D <- start of measurement loop
Pdc8030D40 ;ccAAE483Fm,14,288;cd7FD3127 ,14,288 <- data package
... more data packages ...
Pdc8030D3Fm;cc80EDA04 ,14,287;cd9751491m,14,287 <- data package
* <- end of measurement loop
<- newline indicating end of script
```

13.1.2 LSV example

The following example script runs an LSV from -0.5 V to 1.5 V with steps of 10 mV in 201 steps. The scan rate is set to 100 mV/s. After each step, a data packet will be sent containing the set cell potential and the measured WE current. The measured WE current will be used to autorange.

```
e
var c
var p
#Select channel 0
set_pgstat_chan 0
#Low speed mode is fast enough
set_pgstat_mode 2
#Select bandwidth of 40 for 10 points per second
set_max_bandwidth 40
#Set up potential window between -0.5 V and 1.5 V, otherwise
#the max potential would be 1.1 V for low speed mode
set_pot_range -500m 1500m
#Set current range to 1 mA
set_cr 1m
#Enable autoranging, between current of 100 uA and 5 mA
set_autoranging 100u 5m
#Turn cell on for measurements
cell_on
#equilibrate at -0.5 V for 5 seconds, using a CA measurement
meas_loop_ca p c -500m 500m 5
  pck_start
  pck_add p
  pck_add c
  pck_end
endloop
#Start LSV measurement from -0.5 V to 1.5 V, with steps of 10 mV
#and a scan rate of 100 mV/s
meas_loop_lsv p c -500m 1500m 10m 100m
  #Send package containing set potential and measured WE current.
  pck_start
  pck_add p
  pck_add c
  pck_end
endloop
#Turn off cell when done or aborted
on_finished:
cell_off
```

Example output:

```
e <- ack of 'e' cmd
M0007 <- start of measurement loop (CA)
Pda7F85E36u;ba7F77484p,14,20B <- data package
... more data packages ...
Pda7F85E36u;ba7F77484p,14,20B <- data package
* <- end of measurement loop (CA)
M0000 <- start of measurement loop (LSV)
Pda816E55Fu;ba816DB89p,14,207 <- data package
... more data packages ...
Pda816E55Fu;ba816DB89p,14,207 <- data package
* <- end of measurement loop (LSV)
<- newline indicating end of script
```

13.1.3 SWV example

The following example script runs a SWV from -0.5V to 0.5V with steps of 10 mV in 101 steps. After each step, a data packet will be sent containing the cell potential for that step and current resulting from the SWV measurement.

```
e
var c
var p
var f
var g
set_pgstat_chan 0
set_pgstat_mode 2
#Set maximum required bandwidth based on frequency * 4,
#however since SWV measures 2 datapoints, we have to multiply the
#bandwidth by 2 as well
set_max_bandwidth 80
#Set potential window.
#The max expected potential for SWV is EEnd + EAmp * 2 - EStep.
#This measurement would also work without this command since it
#stays within the default potential window of -1.1 V to 1.1V
set_pot_range -500m 690m
#Set current range for a maximum expected current of 2 uA
set_cr 2u
#Disable autoranging
set_autoranging 2u 2u
#Turn cell on for measurement
cell_on
#Perform SWV
meas_loop_swv p c f g -500m 500m 10m 100m 10
  #Send package with set potential,
  #"forward current - reverse current",
  #"forward current"
  #"reverse current"
  pck_start
  pck_add p
  pck_add c
  pck_add f
  pck_add g
  pck_end
endloop
#Turn off cell when done or aborted
on_finished:
cell_off
```

Example output:

```
e <- ack of 'e' cmd
M0002 <- start of measurement loop
Pda7F85E36u;ba8030DDCp,10,202;ba7FB6915p,10,202;ba7F85B39p,10,202 <- data package
... more data packages ...
Pda807A1CAu;ba8030EB6p,10,202;ba80AB012p,10,202;ba807A15Cp,10,202 <- data package
* <- end of measurement loop
<- newline indicating end of script
```

14 Error handling

Errors can occur that prevent the execution of the MethodSCRIPT. These errors can occur either during the parsing of the script or during the execution of the script (runtime). If the error occurs during parsing, the line nr and character nr where the error occurred will be reported. During runtime, only the line nr will be reported. A command that returns an error will not return an extra newline '\n' after the newline of the error message.

Parsing error format:

| |
|------------------------|
| !XXXX: Line L, Col C\n |
|------------------------|

Runtime error format:

| |
|-----------------|
| !XXXX: Line L\n |
|-----------------|

Where:

XXXX = The error code, see "Table 7: Error codes"

L = Line nr, starting at 1

C = Line character nr, starting at 1

| Code (Hex) | Name | Description |
|------------|--------------------------------|---|
| 0001 | STATUS_ERR | An unspecified error has occurred |
| 0002 | STATUS_INVALID_VT | An invalid Value Type has been used |
| 0003 | STATUS_UNKNOWN_CMD | The command was not recognized |
| 0004 | STATUS_REG_UNKNOWN | Not applicable for MethodSCRIPT |
| 0005 | STATUS_REG_READ_ONLY | Not applicable for MethodSCRIPT |
| 0006 | STATUS_WRONG_COMM_MODE | Not applicable for MethodSCRIPT |
| 0007 | STATUS_BAD_ARG | An argument has an unexpected value |
| 0008 | STATUS_CMD_BUFF_OVERFLOW | Command exceeds maximum length |
| 0009 | STATUS_CMD_TIMEOUT | The command has timed out |
| 000A | STATUS_REF_ARG_OUT_OF_RANGE | A var has a wrong identifier |
| 000B | STATUS_OUT_OF_VAR_MEM | Cannot reserve the memory needed for this var |
| 000C | STATUS_NO_SCRIPT_LOADED | Cannot run a script without loading one first |
| 000D | STATUS_INVALID_TIME | The given (or calculated) time value is invalid for this command |
| 000E | STATUS_OVERFLOW | An overflow has occurred while averaging a measured value |
| 000F | STATUS_INVALID_POTENTIAL | The given potential is not valid |
| 0010 | STATUS_INVALID_BITVAL | A variable has become either "NaN" or "inf" |
| 0011 | STATUS_INVALID_FREQUENCY | The input frequency is invalid |
| 0012 | STATUS_INVALID_AMPLITUDE | The input amplitude is invalid |
| 0013 | STATUS_NVM_ADDR_OUT_OF_RANGE | Not applicable for MethodSCRIPT |
| 0014 | STATUS_OCP_CELL_ON_NOT_ALLOWED | Cannot perform OCP measurement when cell on |
| 0015 | STATUS_INVALID_CRC | Not applicable for MethodSCRIPT |
| 0016 | STATUS_FLASH_ERROR | An error has occurred while reading / writing flash |
| 0017 | STATUS_INVALID_FLASH_ADDR | An error has occurred while reading / writing flash |
| 0018 | STATUS_SETTINGS_CORRUPT | The device settings have been corrupted |
| 0019 | STATUS_AUTH_ERR | Not applicable for MethodSCRIPT |
| 001A | STATUS_CALIBRATION_INVALID | Not applicable for MethodSCRIPT |
| 001B | STATUS_NOT_SUPPORTED | This command or part of this command is not supported by the current device |

| | | |
|------|--------------------------------|--|
| 001C | STATUS_NEGATIVE_ESTEP | Step Potential cannot be negative for this technique |
| 001D | STATUS_NEGATIVE_EPULSE | Pulse Potential cannot be negative for this technique |
| 001E | STATUS_NEGATIVE_EAMP | Amplitude cannot be negative for this technique |
| 001F | STATUS_TECH_NOT_LICENCED | Product is not licenced for this technique |
| 0020 | STATUS_MULTIPLE_HS | Cannot have more than one high speed and/or max range mode enabled (EmStat Pico) |
| 0021 | STATUS_UNKNOWN_PGS_MODE | The specified PGStat mode is not supported |
| 0022 | STATUS_CHANNEL_NOT_POLY_WE | Channel set to be used as Poly WE is not configured as Poly WE |
| 0023 | STATUS_INVALID_FOR_PGSTAT_MODE | Command is invalid for the selected PGStat mode |
| 0024 | STATUS_TOO_MANY_EXTRA_VARS | The maximum number of vars to measure has been exceeded |
| 0025 | STATUS_UNKNOWN_PAD_MODE | The specified PAD mode is unknown |
| 0026 | STATUS_FILE_ERR | An error has occurred during a file operation |
| 0027 | STATUS_FILE_EXISTS | Cannot open file, a file with this name already exists |
| 4000 | STATUS_SCRIPT_SYNTAX_ERR | The script contains a syntax error |
| 4001 | STATUS_SCRIPT_UNKNOWN_CMD | The script command is unknown |
| 4002 | STATUS_SCRIPT_BAD_ARG | An argument was invalid for this command |
| 4003 | STATUS_SCRIPT_ARG_OUT_OF_RANGE | An argument was out of range |
| 4004 | STATUS_SCRIPT_UNEXPECTED_CHAR | An unexpected character was encountered |
| 4005 | STATUS_SCRIPT_OUT_OF_CMD_MEM | The script is too large for the internal script memory |
| 4006 | STATUS_SCRIPT_UNKNOWN_VAR_TYPE | The variable type specified is unknown |
| 4007 | STATUS_SCRIPT_VAR_UNDEFINED | The variable has not been declared |
| 4008 | STATUS_SCRIPT_INVALID_OPT_ARG | This optional argument is not valid for this command |
| 4009 | STATUS_SCRIPT_INVALID_VERSION | The stored script is generated for an older firmware version and cannot be run |
| 7FFF | STATUS_FATAL_ERROR | A fatal error has occurred, the device must be reset |

Table 7: Error codes

15 Device specific information

15.1 PGStat mode properties

EmStat Pico

| Low speed mode | Value min | Value max |
|--------------------------|-----------|-----------|
| Bandwidth | 0.016 Hz | 100 Hz |
| Potential range | -1.25 V | 2.0 V |
| Dynamic potential window | 2.2 V | 2.2 V |
| | | |
| High speed mode | Value min | Value max |
| Bandwidth | 0.016 Hz | 200 kHz |
| Potential range | -1.7 V | 2.0 V |
| Dynamic potential window | 1.214 V | 1.214 V |
| | | |
| Max range mode | Value min | Value max |
| Bandwidth | 0.016 Hz | 100 Hz |
| Potential range | -1.7 V | 2.0 V |
| Dynamic potential window | 2.6 V | 2.6 V |

Table 8: EmStat Pico PGStat mode properties (see EmStat Pico datasheet for more information)

15.2 Current ranges

EmStat Pico

| Low speed mode current ranges | Current follower resistor | Current range index |
|--------------------------------|---------------------------|---------------------|
| 100 nA | 10 MOhm | 0x0 |
| 1.95 uA | 512 kOhm | 0x1 |
| 3.91 uA | 256 kOhm | 0x2 |
| 7.81 uA | 128 kOhm | 0x3 |
| 15.63 uA | 64 kOhm | 0x4 |
| 31.25 uA | 32 kOhm | 0x5 |
| 62.5 uA | 16 kOhm | 0x6 |
| 125 uA | 8 kOhm | 0x7 |
| 250 uA | 4 kOhm | 0x8 |
| 500 uA | 2 kOhm | 0x9 |
| 1 mA | 1 kOhm | 0xA |
| 5 mA | 200 Ohm | 0xB |
| High speed mode current ranges | Current follower resistor | Current range index |
| 100 nA | 10 MOhm | 0x80 |
| 1 uA | 1 MOhm | 0x81 |
| 6.25 uA | 160 kOhm | 0x82 |
| 12.5 uA | 80 kOhm | 0x83 |
| 25 uA | 40 kOhm | 0x84 |
| 50 uA | 20 kOhm | 0x85 |
| 100 uA | 10 kOhm | 0x86 |
| 200 uA | 5 kOhm | 0x87 |
| 1 mA | 1 kOhm | 0x88 |
| 5 mA | 200 Ohm | 0x89 |
| Max range mode current ranges | Current follower resistor | Current range index |
| 100 nA | 10 MOhm | 0x80 |
| 1 uA | 1 MOhm | 0x81 |
| 6.25 uA | 160 kOhm | 0x82 |
| 12.5 uA | 80 kOhm | 0x83 |
| 25 uA | 40 kOhm | 0x84 |
| 50 uA | 20 kOhm | 0x85 |
| 100 uA | 10 kOhm | 0x86 |
| 200 uA | 5 kOhm | 0x87 |
| 1 mA | 1 kOhm | 0x88 |
| 5 mA | 200 Ohm | 0x89 |

Table 9: EmStat Pico current ranges

15.3 Other device specific properties

| Property | EmStat Pico |
|---------------------------|-------------|
| Number of pgstat channels | 2 |

Table 10: Other device specific properties

16 Version changes

Version 1.1 Rev 1

- Added support for EmStat Pico firmware v1.1
- Added “Tags” chapter
- Added Max range pgstat mode for the EmStat Pico
- Added BiPot / Poly WE support
- Added PAD technique
- The ‘e’ command now replies with an extra ‘\n’ to separate the script response from the ‘e’ command response
- Added ability to use whitespace in script (tabs and spaces)
- Added error code documentation

Version 1.1 Rev 2

- Corrected EIS auto ranging information
- Added information about loop command output

Version 1.1 Rev 3

- Corrected OCP parameters, does not have set potential
- Corrected set_pgstat_chan command example
- Corrected SWV example comment about bandwidth
- Correct loop example “add” command should be “add_var”
- Corrected inconsistent names for low power / low speed mode

Version 1.1 Rev 4

- Corrected endloop command was sometimes called end_loop