PalmSens
Compact Electrochemical Interfaces

MethodSCRIPT v1.0

Rev. 26

Contents

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 parameters that are similar to the parameters 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 PSTrace and then exported to MethodSCRIPT. This allows for convenient testing of different measurements in PSTrace. 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)
- 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 EmStat Pico. 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 '#'. Tabs and other extra whitespace besides argument separators are not allowed.

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
var a\n
\n
```

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 “7FFFFF6m”. 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.
,	= Metadata separator.
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 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 Chronoamperometry measurement loop:

```
#Run a measurement loop for the Chronoamperometry 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
end_loop
```

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 a parameter 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 a parameter. Table 5: 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 5: Variable Types

8 Script parameter types

8.1 var

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

8.2 literal

A literal is a constant value parameter, 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.

9 Script commands

9.1 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”.

9.2 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”.

9.3 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‘.

9.4 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’.

9.5 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’

9.6 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’

9.7 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’

9.8 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.

9.9 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.

9.10 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 i 1
endloop
```

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

9.11 endloop

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

Arguments

No arguments.

9.12 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’.

9.13 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 `end_loop` 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.

Example

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

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.

9.14 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 `end_loop` 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.

Example

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

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.

9.15 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 `end_loop` 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".

Example

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

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

9.16 meas_loop_swv

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 `end_loop` 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.

Example

```
meas_loop_swv p c f r -500m 500m 10m 15m 10
pck_start
pck_add p
pck_add c
pck_end
end_loop
```

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).

9.17 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 `end_loop` 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”.

Example

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

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.

9.18 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 `end_loop` 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.

Example

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

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.

9.19 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 `end_loop` command. Refer to section “Measurement loop” for more information.

Arguments

Name	Type	
Output potential	var	Output variable to store the measured RE potential 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.

Example

```
meas_loop_ocp p 500m 100m 2
pck_start
pck_add p
pck_end
end_loop
```

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.

9.20 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 parameters.
- set_autoranging: autoranging is not yet implemented for EIS

This is a measurement loop function and needs to be terminated with an `end_loop` 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
end_loop
```

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.

9.21 on_finished:

Code after this tag is always executed unless an error has occurred. This is useful to be able to set the potentiostat into a known state after a MethodSCRIPT has been aborted.

Arguments

No arguments.

Example

```
cell_on
meas_loop_ca p c 100m 100m 2
pck_start
pck_add p
pck_add c
pck_end
end_loop
on_finished:
cell_off
```

This script turns the cell on and performs a CA measurement. If the measurement is aborted halfway by sending an abort command during the script execution, the cell is turned off. This ensures the cell is always turned off, even if the measurement is aborted.

9.22 set_autoranging

Enable or disable autoranging for all meas_loop_* functions except EIS. 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 parameters. 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.

9.23 pck_start

Signal the start of a measurement data packet.

Arguments

No arguments.

Example

```
pck_start
```

Signal the start of a new measurement package.

9.24 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.

9.25 pck_end

Signal the end of a measurement data package.

Arguments

No arguments.

Example

```
pck_end
```

Signal the end of a measurement data package.

9.26 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_bandwith 1k
```

Set the max bandwidth to a frequency of 1 kHz.

9.27 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

9.28 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.

9.29 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.

9.30 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

PGStat Modes" for more information.

Arguments

Name	Type	
PGStat mode	uint8	Set pgstat mode: 0 = off 2 = low power 3 = high power

Example

```
set_pgstat_mode 3
```

Set hardware configuration to high power mode.

9.31 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.

Example

```
send_string hello world
```

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

9.32 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.

9.33 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.

9.34 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_pot_range 0 1200m
```

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

10 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”.

10.1 PGStat mode off

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

10.2 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.

10.3 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.

11 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.

11.1.1 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
#Set current range for currents of up to 1 mA
set_cr 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
```

11.1.2 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 power 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
```

11.1.3 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
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
```

12 Device specific information

12.1 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

Table 6: EmStat Pico PGStat mode properties (see EmStat Pico datasheet for more information)

12.2 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

Table 7: EmStat Pico current ranges

12.3 Other device specific properties

Property	EmStat Pico
Number of pgstat channels	2

Table 8: Other device specific properties